ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Д И П Л О М Н А Р А Б О Т А

Тема: Проектиране и изграждане на управляващ софтуер за бордови компютър за Mercedes-Benz Actros MPII 1844 LS, базиран на развоен хардуер OLIMEX AVR TLCD128CAN

Дипломант:
Ивайло Стоянов

Научен ръководител:
Марин Балканджиев

С О Ф И Я

2 0 0 9

ТЕХНОЛОГИЧНО УЧИЛИЩЕ
ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Дата на заданието: 20.11.2008 г. Утвърждавам:.....
Дата на предаване: 28.02.2009 г. / проф. д-р инж. Р. Иванов /

ЗАДАНИЕ
за дипломна работа

на ученика Ивайло Николаев Стоянов

1.Тема: Проектиране и изграждане на управляващ софтуер за бордови компютър за Mercedes-Benz Actros MPII 1844 LS, базиран на развоен хардуер OLIMEX AVR TLCD128CAN

2.Изисквания:
- Визуализиране на работните параметри на автомобила
- Реализиране на функция бордови компютър
- Реализиране на съветник за превключване на предавките

3.Съдържание 3.1 Обзор

3.2 Същинска част

3.3 Приложение

Дипломант :.....
Ръководител:.....
/ Марин Балканджиев /

Директор:.....
/ ст.пр. д-р А. Френски /

Увод

Автомобилите стават все по-модерни и е неизбежна нуждата от създаването на все по-мощен, по-сложен и по-ефективен управляващ софтуер, който да контролира всяка част от съвременния автомобил. Текущият стремеж е всяка част от него да се управлява електронно, за да бъде работата ѝ по-прецизна, по-надеждна и по-сигурна, а експлоатацията ѝ по-дългосрочна и по-функционална. Внедрява се по-голям брой електронни блокове, които да отговарят за реализирането на определена част или функционалност на автомобила. Например електронен блок за управление на двигателя, електронен блок за осветлението, електронен блок за контрол на спирачната система и т.н. Тенденцията е цялата тази електроника в автомобилите да изпълнява вграден управляващ програмен продукт - Embedded Software, който да реализира съответната задача.

Ако функционалността на определен електронен блок не е реализирана с използването на вграден управляващ софтуер, а чисто хардуерно, без процесор, който да управлява целия блок, то той може да изпълнява точно определен брой задачи – няма гъвкавост и възможност за бързо модифициране на функционалността, изпълнявана от електрониката. Ако се използва хардуер, който може да изпълнява вграден управляващ софтуер, то тази единствен хардуер може да реализира множество функционалности. При необходимост от промяна на функционалността,

изпълнявана от въпросния електронен блок, не е нужно да се проектира нов хардуер, който да отговаря на новите изисквания , а само управляващият го софтуер да бъде модифициран, за да срещне новите изисквания към функционалността. Казано с други думи - добавя се нов слой в автомобилната електроника - софтуерен, който да надгради хардуерния, за да може да бъдат реализирани софтуерно задачи, чието създаване преди не е било възможно.

Такъв е и предметът на текущата дипломна работа – да бъде създаден управляващ софтуер за такъв електронен блок– бордови компютър - устройство, чиято основна функция е да доставя на водача на превозното средство информация за пътуването и за текущото състояние на превозното средство.

Целите на дипломната работа са:

- Създаване на вграден софтуер, който да управлява развойния хардуер OLIMEX AVR TLCD128CAN;
- Управляващият софтуер да реализира доставянето на водача на информация за текущото състояние на автомобила;
- Управляващият софтуер да обработва получената информация за текущото състояние на автомобила и да предоставя нова такава на водача – функция бордови компютър;
- Управляващият софтуер да реализира функция съветник за смяна на предавките;

- Управляващият софтуер да може да бъде конфигуриран така, че да може да работи и за други автомобили освен посочения Mercedes-Benz Actros MPII 1844 LS.

За постигане целите на дипломната работа трябва да се извършат множество задачи, които са описани в по-нататъшните глави, като най-важните от тях са усвояване на програмирането на микропроцесорни системи, както и проучването на целевия автомобил.

Глава I

Проучването е неизменна част от разработването на всеки програмен продукт. В тази глава ще бъде разгледана точно тази фаза от създаването на дипломната работа.

Първата част от създаването на дипломната работа включва:

- Проучване на подобни решения;
- Проучване на целевия автомобил;
- Проучване на хардуерни платформи
- Проучване на среди за разработка на вграден софтуер;
- Проучване на езици за разработване на вграден софтуер;
- Проучване на RTOS.

1.1.Проучване на подобни решения

Множество съвременни автомобили имат бордови компютри. Тяхната полезност и функционалност са безспорни, поради голямото количество информация, което дават на водача за превозното средство. Различните бордови компютри дават различна информация на водача. Според начина на получаване тази информация може да бъде два вида:

- Измерена информация;
- Изчислена информация.

Първият тип е измерената информация. Тоест, това е информация, която се получава чрез отчитане и преобразуване на показанието от измерването на датчик, който преобразува измерваната величина в електрическа такава. Например величина, която се показва на бордовия компютър и от този първи тип е оборотите на двигателя. За целта се използва датчик, монтиран между двигателя и скоростната кутия на автомобила и отчитащ по определен начин броя на завъртанията на коляновия вал. Възможна реализация е датчикът да преобразува отчетения брой зъбци на маховика в правоъгълни импулси. Например един зъб от маховика се отчита от датчика с един правоъгълен импулс. Когато за оборотите е използван този метод за следенето им и се знае броя на зъбците на маховика е лесно да се преобразува получената поредица от правоъгълни импулси в брой на импулсите в минута. Това преобразуване се извършва от бордовия компютър или от друг блок, който му предоставя тази информация. Така се извършва преходът от обектът на измерваната величина – двигателят до показанието на бордовият компютър на измерваната величина. Това е една примерна поредица за измерването на оборотите на целевия автомобил.

Другият вид информация е изчисленият тип информация. Тоест, това е информация, която не се измерва директно от датчик, както в предишния пример, а е резултат от изчисление от няколко величини. Тези

величини, на базата на които се извършва изчислението могат да бъдат от всеки от двата типа информация. Пример за изчислена информация е разстояние, което може да бъде пропътувано с текущото количество гориво в резервоара и текущия разход на гориво. Двете величини, на базата на които се извършва изчислението са количеството на горивото в резервоара и разходът на гориво. Количеството на горивото се измерва най-често чрез резистивен датчик, намиращ се в резервоара на автомобила и показанието му се отчита от бордовия компютър или от друг електронен блок (количеството на гориво е величина от тип едно). Това най-често се реализира чрез измерване на пада на напрежение върху резистивният датчик, който измерва съпротивлението си в зависимост от количеството гориво в резервоара. Този пад на напрежение е аналогова величина, която трябва да се преобразува в цифрова. Това се извършва чрез аналогово-цифров преобразувател. Моментният разход е величина, чието измерване е тясно зависимо с горивната система на целевия автомобил и измерването и може да става по множество начини. Затова за примера се приема, че има цифрова информация за текущия разход на гориво за 100 км. Така при наличието на тези две величини може да изчислим какво е разстоянието, което може да бъде пропътувано с текущата големина на тези две величини. Изчислението е просто и се получава желаното разстояние. Това е пример на информация от тип две – изчислена информация. Тя се изчислява от две величини, които се

измерват, т.е. са от тип едно. Може величина от тип две да бъде получена от величина от тип едно и времето като величина (времето също е от тип едно). Пример за това е изминатото разстояние: измерената величина е скоростта и чрез използване на времето, през което автомобилът се е движил с тази скорост се получава величината от втори тип изминато разстояние.

Бордовият компютър дава разнообразна информация. Той може само да събира информацията от различни блокове на автомобила и да я предоставя на водача на превозното средство. Възможно е и тази информация да бъде пряко измервана от него.

Например оборотите на двигателя – това е основен параметър на двигателя, който е необходим не само за използването му от бордовия компютър, която визуализира на водача на автомобила оборотите на двигателя – този параметър е нужен на електронния блок на двигателя. Информацията от датчика за оборотите постъпва в този блок, той я преобразува и използва, за да изчислява количеството гориво, което трябва да бъде подадено за поддържането, увеличаването или намаляването на оборотите например. При такава реализация бордовият компютър може да извлича преобразуваната информация от този блок и да я визуализира на водача като обороти на двигателя. Но съществува и друга реализация, при която електронния блок на бордовият компютър е свързан директно с датчика и процесите на преобразуване на този сигнал се извършват от самия електронен блок на

бордовия компютър. След преобразуването сигналът се визуализира на индикацията и водачът получава информация за измервания параметър.

Ето кратък списък на някои от величините, които обикновено се визуализират от бордовите компютри на различните автомобили:

- Скорост на автомобила;
- Обороти на двигателя;
- Моментен разход на гориво;
- Количество гориво в резервоара;
- Изминато разстояние - временно;
- Изминато разстояние – общо;
- Разстояние, което може да бъде пропътувано с количеството гориво в резервоара;
- Текуща предавка;
- Съветник за превключване на предавките;
- Температура в купето;
- Външна температура;
- Индикация за отворена врата.

Бордовият компютър дава информация за много величини от състоянието на автомобила, както и за пътуването. Но цялата тази информация трябва да бъде визуализирана, чрез определен вид индикация. Най-често това е екран, на който се визуализират данните. Този екран може да бъде изграден от сегменти, от които се образуват буквите и цифрите. Друго решение (по-скъпо) е екран ,

изграден от масив или матрица от точки (пиксели), при който освен букви и цифри могат да бъдат визуализирани и картинки и иконки, тоест се разширява функционалността на екрана като индикация. Използването на цветен екран също разширява функционалността на индикацията, но също така увеличава и цената на бордовия компютър.

1.2.Проучване на целевия автомобил

Когато трябва да се създаде бордови компютър е много важен въпросът за какъв автомобил ще се реализира електронния блок. Различните автомобили измерват отделните параметри на пътуването по различен начин и за да се направи бордови компютър за съответния автомобил трябва тези спецификации да бъдат известни.

Необходимо е да бъдат известни технически данни относно автомобилът, за който ще бъде създаден бордовият компютър. Освен това са необходими и данни за това как се измерват работните параметри от автомобила. Как работят датчиците, които измерват параметрите на автомобила, къде са монтирани, в какви електрически величини се преобразува измервания параметър, дали изходът на датчика е в цифров или в аналогов вид, тоест необходимо ли е преобразуването му от аналогов в цифров вид. Технически данни за машината са също необходими – като например вместимостта на резервоара, минимални и максимални обороти на двигателя и др. Тази информация

ще бъде подробно изложена в точка 2.3. на настоящата дипломна работа.

1.3. Проучване на хардуерни платформи

За създаване на какъвто и да е електронен блок е необходима хардуерна платформа, която да реализира функционалността на електронния блок. Тенденцията е тази хардуерна платформа, да има процесор, която да изпълнява управляващия вграден софтуер, тоест цялата функционалност да бъде реализирана софтуерно. Управляващият софтуер се създава и компилира на персонален компютър, на който работи и програмистът, но след това се въвежда на FLASH паметта за програмен код на микропроцесора и бива изпълняван от него. Този похват се нарича „кръстосано компилиране” (cross-compiling). Развойната среда трябва да позволява извършването на такова програмиране – това се случва с т.н. **TAP** – **T**est **A**ccess **P**ort, разработен от JTAG (**J**oint **T**est **A**ction **G**roup), чрез който се извършва още настройката и търсенето на грешки на програмния код, изпълняван от процесора, както и други функции. Този метод е използван главно в нискобюджетни проекти. Друг начин е използването на **In-Circuit Emulator**, който напълно емулира функциите на целевия процесор, и позволява пълен контрол както над ядрото, така и на прилежащата периферия; в допълнение към емулатора има и допълнителен инструмент: **Trace**,

който може да запаметява поредицата от инструкции изпълнени от процесора.

В програмирането на вградени микропроцесорни системи се използват множество вградени периферни устройства, които реализират разнообразни функционалности. Такива например са:

- IC (**I**nput **C**apture)
- ADC (**A**nalog to **D**igital **C**onverter)
- TWI (**T**wo **W**ire **I**nterface)
- SPI (**S**erial **P**eripheral **I**nterface)
- CAN (**C**ontroller **A**rea **N**etwork)

IC (**I**nput **C**apture) позволява на микропроцесора да измерва честотата на входен сигнал от правоъгълни импулси. Измерването на честота може да се извършва чрез директно измерване на честотата или чрез измерване на периода на входния сигнал и намирането на честотата чрез изчисления.

ADC (**A**nalog to **D**igital **C**onverter) – Аналогово-Цифров Преобразовател - се използва за преобразуване на аналогови величини в цифрови такива. Такава величина е например напрежение. Използването на този метод се налага например при измерването на пада на напрежение на резистивни датчици.

Вграденият модул за **TWI** (**T**wo **W**ire **I**nterface) - (наречена още **I2C** (**I**nter-**I**ntegrated **C**ircuit)) позволява комуникация на външни периферни устройства и микропроцесора. Устройствата работят в “master-slave”

режим и за комуникацията са необходими 2 проводника. Начинът на комуникация е „**half-duplex**". Технологията е стандартизирана от Philips. Използва се в много датчици, които използват шината за данни, за да изпращат информация към микропроцесора.

SPI (**S**erial **P**eripheral **I**nterface) – е начин за комуникация на периферни устройства и микропроцесора. Устройствата работят в “master-slave” режим и за комуникацията са необходими 4 проводника. Начинът на комуникация е „**full-duplex**". Изобретена и стандартизирана е от Motorola.

CAN (**C**ontroller **A**rea **N**etwork) – мрежа, която се използва за свързване на отделните електронни блокове на автомобилите помежду си. Устройствата, свързани към CAN мрежата работят в “**multi-master**” режим. Използва се още и за свързване на датчици към електронните блокове. Създадена е от Bosch през 80-те години. Актуалната версия на CAN протокола е CAN 2.0, представена от Bosch през 1991 година.

1.4. Проучване на среди за разработка на вграден софтуер

За разработката на управляващ софтуер за вградени микропроцесорни системи е необходимо да се направи избор на среда за разработка на програмният продукт. В тази чест на дипломната работа ще бъдат представени два

такива продукта за разработване на вграден софтуер за фамилията микропроцесори AVR на Atmel Corp.:

- AVR Studio
- IAR Embedded Workbench

AVR Studio е интегрирана среда за разработването на вграден управляващ софтуер. Производителят на системата е Atmel Corp. и е разработена специално за тази фамилия микропроцесори – AVR. Средата се състои от мениджър на проектите, редактор за сорс файлове и чип симулатор. Предлага богати възможности за следене на програмното изпълнение, както и такова стъпка по стъпка. Поддържаните езици са C, Pascal, BASIC и асемблер. Средата не съдържа компилатор и трябва да бъде добавят външен такъв. WinAVR е подходящ компилатор за тази цел, предназначен за интегрирането в средата AVR Studio. Работи под операционна система Windows. Тази среда за разработване, както и компилаторът се разпространяват свободно.

IAR Embedded Workbench е професионалното решение за разработването на вграден софтуер. То представлява набор от инструменти за разработване, настройка и коригиране на вградени програмни продукти, написани на асемблер, C или C++. Средата се състои от мениджър на проектите, редактор за сорс файлове, дебъгер и вграден компилатор. Предлагат се оптимизации на кода с цел намаляване на големината на изходното приложение. Предлага се и „MISRA C Checker”, тоест модул, който проверява дали кода е съобразен с изискванията за

надеждност на софтуера MISRA (***M****otor* ***I****ndustry* ***S****oftware* ***R****eliability* ***A****ssociation е асоциация, разработила система от правила за създаване на вграден управляващ софтуер за автомобилите)*. Продуктът работи под операционна система Windows и лицензът му е платен. IAR Kickstart Edition е безплатен, но поради наложеното ограничение от четири килобайта размер на кода, използването му е неприложимо за текущата дипломна работа.

1.5. Проучване на езици за разработване на вграден софтуер

Друг важен въпрос при разработването на вграден софтуер е изборът на програмен език. В програмирането на микропроцесори най-разпространени са езиците C, C++ и асемблер. Възможно е и използването на езици като Pascal и Java, но тези практики не са особено разпространени.

Някой възможности на AVR микропроцесорите могат да бъдат използвани само с код, написан на асемблер. В почти всички случай изходният код, написан на асемблер ще бъде по-малък от този, написан на език за програмиране от високо ниво. Затова асемблерът бива използван за приложения, където големината на изходния код е от решаващо значение.

C++ е разпространен език за програмиране. Не голямото му използване се дължи на недостатъчната преносимост, не добрата стандартизация, както и на по-слабата му поддръжка от MISRA.

Езикът C е най-разпространеният подход за програмирането на вградени микропроцесорни системи. Той е по-лек за хардуера от C++, а освен това предлага много добри възможности за програмиране и на ниско ниво, което го прави конкурентноспособен на асемблера.

1.6. Проучване на RTOS

Операционните системи от реален тип(RTOS - **R**eal-**T**ime **O**perating **S**ystem) са много често използвани във вградените микропроцесорни системи за приложения от реален тип. Специфичното при този тип операционни системи е, че е важно не само изпълнението на определена задача, но и времето за което ще бъде извършено. Real-Time означава, че е необходимо извършването на дадена задача единствено и само в определени граници. „Дали закъснелият отговор е толкова лош, колкото и грешният отговор?”. Това е въпросът, на който трябва да се даде отговор, за да се разбере, дали приложението е от реален тип и съответно операционната система необходима за реализацията трябва да е от този тип. В настоящата дипломна работа е от особена важност своевременното показване на работните параметри на

автомобила. Предлагат се, както платени операционни системи от реален тип, така и такива със свободен лиценз:

Платени: Windows CE (Microsoft Corp.), ThreadX (Express Logic Inc.) и др.

Свободни: RTLinux(Victor Yodaiken), AvrX(Larry Barello)

Глава II

Във втората глава на текущата дипломна работа ще се дефинират изискванията към вградения програмен продукт, ще бъдат описани в детайли използваните програмни езици, среди за разработка на вграден софтуер, задълбочен обзор на използваните вградени модули на процесора.

Структура на втора глава:

- Дефиниране на изискванията към предмета на дипломната работа
- Средствата за разработка
- Детайлен преглед на целевия автомобил
- Детайлен преглед на развойната хардуерна платформа OLIMEX AVR TLCD128CAN
- Детайлен преглед на използваните вградени модули

2.1. Дефиниране на изискванията към предмета на дипломната работа

В текущата дипломната работа предстои да бъде създаден електронен блок, реализиращ функцията бордови компютър. Изискванията към него ще бъдат дефинирани в тази част на дипломната работа.

Електронният блок – бордови компютър, реализиран в текущата дипломната работа, трябва да поддържа следните функционалности и да изпълнява следните условия:

- Визуализиране на скоростта на автомобила

Визуализирането на скоростта да се извършва чрез показване на измерената стойност на екрана на хардуерната платформа. Мерната единица да бъде километър в час (km/h).

- Визуализиране на оборотите на двигателя

Визуализирането на оборотите да се извършва чрез показване на измерената стойност на екрана на хардуерната платформа. Мерната единица да бъде обороти в минута (rpm). При работа на двигателя на максимални обороти и активиране на ограничителя на оборотите на двигателя (2500rpm - вж т. 2.3.) да се показва индикация, отразяваща ситуацията.

- Визуализиране на текущото количество гориво в резервоара

Визуализирането на текущото количество гориво в резервоара да се извършва чрез показване на измерената стойност на екрана на хардуерната платформа. Да се извършва в два режима – абсолютен и относителен. Мерната единица да бъде литри(L) в абсолютен режим и проценти(%) за относителен режим. При достигане на минимума на количеството гориво в резервоара (14% или 147 литра при вместимост от 1050 литра - вж т. 2.3.) да се

показва индикация, информираща за ситуацията, че е достигнат прага за индикация на „резервата”.

- Визуализиране на моментен разход

Визуализирането на моментния разход да се извършва чрез показване на измерената стойност на екрана на хардуерната платформа. Стойността да се извежда в следния формат: **XX.Y**, където **XX** е цялата част от стойността, а **Y –** дробната, т.е. точността да измерването да е до първи знак след десетичната запетая. Мерната единица да бъде литри на 100km (L/100km).

- Визуализиране на изминато разстояние – временно

Визуализирането на временното изминатото разстояние да се извършва чрез показване на изчислената стойност на екрана на хардуерната платформа. Мерната единица да бъде километри (km). Показанието на този параметър да може да бъде нулирано.

- Визуализиране на изминатото разстояние – цялостно

Визуализирането на цялостното изминатото разстояние да се извършва чрез показване на изчислената стойност на екрана на хардуерната платформа. Мерната единица да бъде километри (km). Показанието на този параметър да не може да бъде нулирано.

- Визуализиране на оставащ пробег до следващо зареждане

Визуализирането на оставащия пробег до следващото зареждане с гориво да се извършва чрез показване на

изчислената стойност на екрана на хардуерната платформа. Мерната единица да бъде километри (km).

- Визуализиране на външната температура

Визуализирането на външната температура да се извършва чрез показване на измерената стойност на екрана на хардуерната платформа. Мерната единица да бъде градуси Целзий (°C). Диапазона на измерваната температура да бъде от „-40°C” до „+125°C”. Точността на измерването да бъде ±2°C.

- Визуализиране на текущата предавка

Визуализирането на текущата предавка да се извършва чрез показване на изчислената стойност на екрана на хардуерната платформа. Освен цели предавки да бъдат показвани и половинки такива (Осем-степенна предавателна кутия с половинки предавки – бързи и бавни - вж т. 2.3.)

- Реализиране на съветник за превключване на предавките

Реализирането на съветника за предавките да се извършва, чрез показване на екрана на хардуерната платформа на вертикални стрелки, съветващи на водача в каква посока да смени предавка – превключване надолу или превключване нагоре. Съветника да работи в два режима – икономичен и динамичен. При първия режим водачът да се съветва за начин на смяна на предавките с цел оптимизиране на разхода на гориво, а при втория да се съветва за начин на превключване на предавките с цел

по-голямо ускорение. При първата предавка да не се дава съвет за превключване надолу. При последната предавка да не се дава съвет за превключване нагоре.

- Реализиране на часовник и календар

Да се визуализира на екрана на хардуерната платформа следната информация относно календара и часовника:

- Ден от седмицата – една от стойностите: "**Mon**", "**Tue**", "**Wed**", "**Thu**", "**Fri**", "**Sat**", "**Sun**"
- Дата – да отговаря на броя дни за съответния месец;
- Месец – една от стойностите: "**Jan**", "**Feb**", "**Mar**", "**Apr**", "**May**", "**Jun**", "**Jul**", "**Aug**", "**Sep**", "**Oct**", "**Nov**", "**Dec**"
- Година - да се прави проверка за високосна година, с цел определяне броя на дните на месец Февруари;
- Часове и минути – да отговарят на формата **HH**:**MM**, където **HH** е показанието за часовете, а **MM,** показанието за минутите

При реализирането на индикация на параметри на автомобила като скорост, обороти на двигателя трябва да бъде известна техническа информация за целевия автомобил, както и за начина, по който целевият автомобил измерва тези стойности. Тази информация е изложена в точка 2.3. на настоящата дипломна работа.

2.2. Средствата за разработка

В тази част на втората глава на текущата дипломна работа ще бъдат разгледани необходимите средства за създаване на управляващия софтуер на бордовият компютър. Тези средства са :

- Език за програмиране;
- Среда за програмиране;
- Компилатор и библиотеки;
- Операционна система.

В следващите части на тази точка ще бъде направено описание на всяко от използваните средства, както и мотивация за избора на всяко едно от тях.

Езикът за програмиране C е избран в настоящата дипломна работа, за разработване на вградения управляващ софтуер на бордовия компютър. Този език придоби огромна популярност при програмирането на вградени микропроцесорни системи. Преимуществата на този език пред асемблера са няколко: по-малкото време за разработване, лесната поддръжка и преносимост на кода между устройствата и възможността на многократната му употреба в множество различни приложения. Недостатъците на езика C пред асемблера са: по-големия обем на програмата, което често води до по-бавната ѝ изпълнение от микропроцесора. Избраният процесор от фамилията AVR на Atmel Corp. е оптимизиран за работа с

този език за програмиране, което допълнително мотивира избора на този език за разработка.

Средата за програмиране, избрана за разработка в текущата дипломна работа е **AVR Studio**, създадена от Atmel Corp. – производителят и на фамилията микропроцесори, избрана в настоящата дипломна работа. **AVR Studio** е интегрирана среда за разработка, предоставяща широки възможности за създаване, настройка и коригиране на вграден управляващ софтуер за всички процесори от фамилията AVR на Atmel Corp. Средата предоставя текстов редактор за сорс-кода на приложението, големи възможности за проследяване на начина на изпълнението на приложението, инструменти за управление на проектите. Предоставени са още възможности за симулиране на работата на развойния хардуер, съставен от осем битов микропроцесор AVR. Това са основните причини, поради които **AVR Studio** е избрана за среда за разработка в текущата дипломна работа.

Компилаторът и библиотеките, избрани за дипломната работа са **WinAVR**, който представлява пакет, включващ всички необходими инструменти за разработването на вградени приложения за микропроцесорите AVR. Компилаторът е GNU GCC и има поддръжка за C и C++. Освен компилатор пакетът WinAVR включва и програматор и дебъгер, които лесно се свързват с средата за разработка AVR Studio. Включени са и библиотеки за работа с процесорите AVR, които улесняват много процеса на разработка на вграден софтуер.

Леснотата за работа и за свързване с развойната среда AVR Studio, както и факта, че се разпространяват безплатно са основните причини за избора на WinAVR за настоящата дипломна работа. С цел повишаване на преносимостта на кода на дипломната работа, то са предефинирани основните типове данни. При необходимост от пренасянето на кода на друга платформа дефинициите могат да бъдат сменени.

Операционната система, използвана в текущата дипломна работа е AvrX. Създателят ѝ е Larry Barello. (вж. Използвана литература 3.) Това е многозадачна RTOS. Написана е специално за фамилията микропроцесори AVR на Atmel Corp. Основните и характеристики са следните:

- Многозадачност;
- Механизми за синхронизация;
- Управление на таймерите;
- Опашки от съобщения;
- Изпълнение на програмата стъпка по стъпка с цел настройка.

Написана е на асемблер, което прави кода ѝ малък и лек и не се натоварва допълнително хардуера, на който работи. Операционната система се представя като библиотеки от функции и затова паметта, която заема е само тази на използваните библиотеки. Може да бъде използвана с езикът С и такава е употребата ѝ в текущата дипломна работа. Операционната система се разпространява свободно.

AvrX поддържа многозадачност. Превключването на изпълнението на отделните задачи (нишки), съответно на контекста на всяка от задачите се извършва от т.н. „scheduler” или диспечер на задачите. Всяка задача има свой приоритет, който може да бъде на едно от шестнадесетте нива. По-малкото число, означава по-голям приоритет, т.е. задача с приоритет 0 има най-висок приоритет, респективно 16 – най-малък приоритет. Всяка задача има свой Process ID block (**PID**) или блок, който съдържа уникален идентификатор на задачата, както и т.н. Stack pointer, който следи инструкциите за изпълнение.

На лице са следните функции, които се предоставят от операционната система за организиране на многозадачност:

`AVRX_GCC_TASKDEF(demo, 200, 1)` – Дефиниране на задачата с име „demo”, стек с размер 200 и приоритет 1;

`AvrXInitTask(&taskTCB)` – Инициализират се всички регистри, но не се стартира задачата;

`AvrRunTask(&taskTCB)` – Последователно извикване на **`AvrXInitTask(&taskTCB)`** и на **`AvrXResume();`**

`AvrXSelf()` – Връща идентификаторът на извикващата задача;

`AvrXPriority(&PID)` – Връща приоритета на извикващата задача;

AvrXChangePriority(&PID, unsigned) – Променя приоритета на задачата от първия аргумент, с втория аргумент;

AvrXSuspend(&taskPID) – Задачата се маркира за премахване от опашката на изпълняващите се задачи;

AvrXResume(&taskPID) – Задачата се маркира за поставяне в опашката на изпълняващите се задачи;

AvrXTaskExit() – Задачата е приключила работата си и не е нужно повече да се изпълнява;

AvrXTerminate(&taskPID) – Същото като **AvrXTaskExit()**, но с разликата, че тази функция се използва за терминиране на други задачи, различни от извикващата;

AvrXYield() – Задачата се премахва от опашката на изпълняващите се задачи и се зарежда отново на нея;

На лице са следните функции, които реализират механизми за синхронизация между задачите, чрез използването на семафори:

AVRX_MUTEX(A) – деклариране на семафор;

AvrXSetSemaphore(&mutex) – Семафора преминава в състояние **SEM_DONE**(1). Събуждат се чакащи задачи, ако има такива;

AvrXIntSetSemaphore(&mutex) – Същото като **AvrXSetSemaphore(&mutex)**, но с разликата, че тази функция може да бъде извиквана от функция, обработваща прекъсване;

AvrXWaitSemaphore(&mutex) – Задачата започва да чака на този семафор;

AvrXTestSemaphore(&mutex) – Тестване на състоянието на семафора. Извикващата задача не блокира на извикването на тази функция. Връщаната стойност е една от следните:

- **SEM_PEND**(0) – семафорът е изчистен(начално положение);
- **SEM_DONE**(1) – семафорът е бил задействан;
- **SEM_WAIT**(2) – всяка друга стойност е адрес на идентификатор на задача;

AvrXIntTestSemaphore(&mutex) – Същото като AvrXTestSemaphore(&mutex), но с разликата, че тази функция може да бъде извиквана от функция, обработваща прекъсване;

AvrXResetSemaphore(&mutex) – Сменя състоянието на семафора на **SEM_PEND**(0);

Операционната система AvrX предоставя и възможности за работа таймери. За тяхната работа е необходимо да се зададе времева база на операционната система. Това става чрез функцията **AvrXTimerHandler()**. Тази функция трябва да бъде извиквана на определен интервал от време. Това се извършва чрез функцията, обработваща прекъсване на някой от хардуерните таймери на процесора (вж. т. 2.4.). Времето между прекъсванията на хардуерния таймер е всъщност времето между извикванията на функцията **AvrXTimerHandler()**.

На лица са следните функции, предоставящи възможността за работа с таймери:

AVRX_TIMER(A) – деклариране на таймер;

AvrXTimerHandler() – Задава се времева база за работа на операционната система;

AvrXStartTimer(&TCB, unsigned) – Стартира таймерът в първия параметър да брои до втория параметър. Втория параметър представлява брой времеинтервали или иначе казано броя извиквания на функцията **AvrXTimerHandler()**;

AvrXDelay(&TCB, unsigned) – Това е блокираща версия на функцията **AvrXStartTimer** и представлява последователно извикване на функциите **AvrXStartTimer** и **AvrXWaitTimer**;

AvrXCancelTimer(&TCB) – Спира таймера, предаден чрез първия параметър;

AvrXWaitTimer(&TCB) – Извикващата задача блокира, докато не изтече таймера, предаден чрез първия параметър;

AvrXTestTimer(&TCB) – Проверява се състоянието на таймера. Връщаните стойности са същите като при семафорите;

Други функции, предоставени от операционната система:

BeginCritical – Забраняват се глобално всички прекъсвания;

EndCritical – Разрешават се глобално всички прекъсвания;

`Epilog()` – Превключване на контекста от този на операционната система, към този на първата задача в опашката на задачите;

`IntProlog()` – Превключване на контекста от този на задачите на този на операционната система;

Фактът, че операционната система AvrX е написаната на асемблер, което предполага малкото количество на памет, заемано от нейния код, както и това, че е проектирана специално за фамилията микропроцесори AVR на Atmel Corp са основните причини за избора ѝ за текущата дипломна работа. Операционната система предоставя функции които са необходими за целта на дипломната работа, като от тях най-важните са функциите свързани с реализацията на реакция на софтуера в реално време и многозадачието - това е определящата за избора на операционната система причина.

2.3. Детайлен преглед на целевия автомобил

В настоящата дипломната работа за целеви автомобил е избран Mercedes-Benz Actros MPII 1844 LS. Целта на дипломната работа е бордовият компютър да може да работи със специфичната информация за този автомобил, както и да може управляващият му софтуер лесно да бъде преконфигуриран, за да работи със специфичните данни на друг автомобил.

Информацията за целевия автомобил се предоставя на разработчика и той няма задължението да я набавя сам. В настоящата дипломна работа информацията е набавена главно от интернет и от книжна литература за целевия автомобил (вж „Използвана литература”). Поради факта, че някой данни не се оповестяват публично е невъзможно свободното им намиране и използване. Необходимата информация, която трябва да се набави за целевия автомобил може да бъде разделена по следния начин:

- Техническа информация за целевия автомобил;
- Специфична информация за начина, по който целевия автомобил измерва работните параметри.

За конкретния целеви автомобил Mercedes-Benz Actros MPII 1844 LS бяха набавени тези данни с цел прилагането им в управляващия софтуер на бордовия компютър, за да се даде пример за работоспособността на вградения управляващ програмен продукт.

2.3.1.Техническа информация за целевия автомобил

Mercedes-Benz Actros е тежкотоварен камион, представен от Мерцедес-Бенц през 1996 година. Той е предназначен за превозването на товари на дълги разстояния, включително на тежки извънгабаритни товари,

както и за строителни цели. Способен е да превозна товари между 18 и 44 тона. Използват се шест и осен цилиндрови дизелови двигатели с мощност от 313к.с. до 612к.с. , снабдени с турбокомпресори и интеркулери. Actros е снабден с множество електронни системи за сигурност като ABS и ASR са само част от тях. Actros е и екологичен – поддържаните стандарти за опазване на околната среда са EURO4 и EURO5. Първото поколение на Actros се произвежда от 1996 до 2001 година. Второто - от 2002 до 2008, а третото е представено в края на 2008 година. Всяко от трите поколения на Actros е печелило престижната награда “Truck Of The Year” съответно за 1997, 2004 и 2009 година. (вж. Използвана литература 3.).

На фигура 2.3.1.1 е показана снимка на целевия автомобил – Mercedes-Benz Actros

Фигура 2.3.1.1

Целевият автомобил за текущата дипломна работа е Mercedes-Benz Actros MPII 1844 LS – седлови влекач от второ поколение (MP2 = **M**odell**p**rojekt **2**). Техническите данни на въпросната машина са дадени в таблица 2.3.1.2.

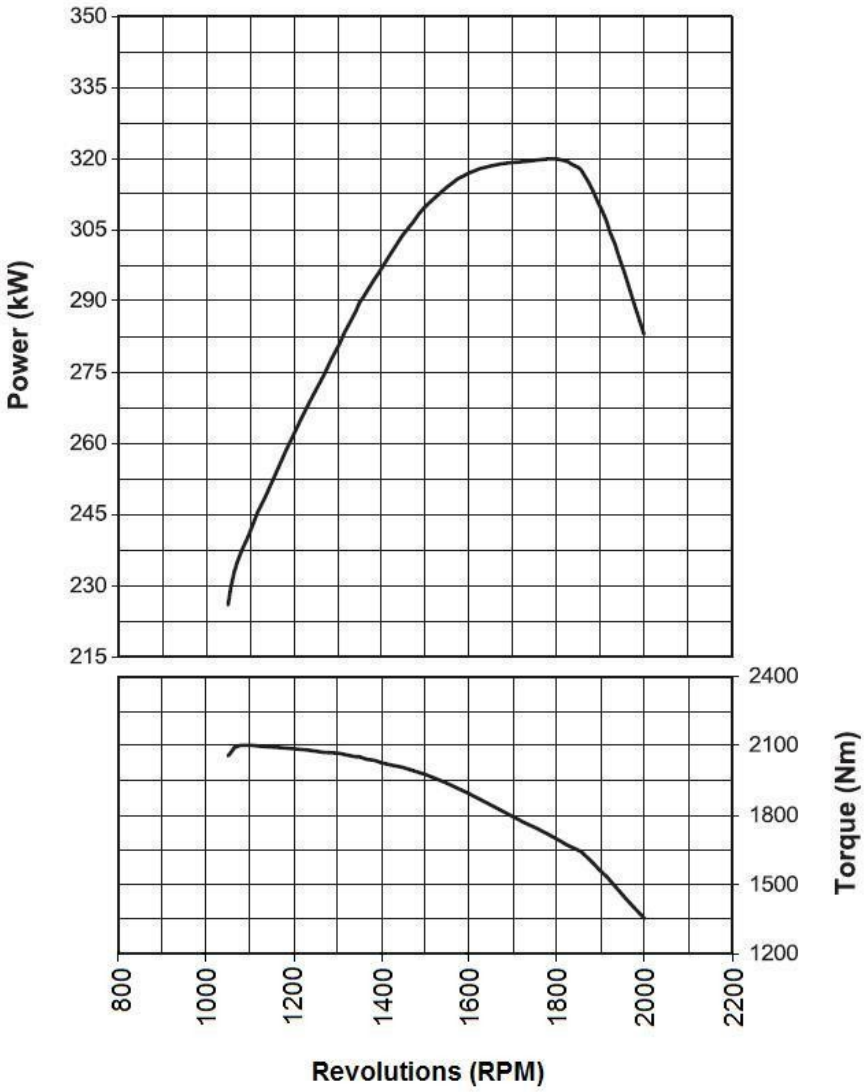
Технически данни за Mercedes-Benz Actros MPII 1844 LS

Величина	**Стойност**
Дължина	6117mm
Ширина	2494mm
Височина	3850mm
Междуосие	3900mm
Маса	6910kg
Допустима маса с ремарке	40 000kg
Мощност	(320KW)435к.с. @ 1800rpm
Максимален въртящ момент	2100Nm @ 1080rpm
Работен обем	11.95 l
Цилиндри	6, V-образно разположени
Клапани на цилиндър	4
Горивна система	Директно впръскване
Гориво	Дизел
Капацитет на резервоара	1050 l
Скоростна кутия	Механична
Брой предавки	16
Брой предавки-задни	2

Фиг. 2.3.1.2.

На фигура 2.3.1.3. е дадена динамичната характеристика на двигателя на целевия автомобил на дипломната работа.

Динамична характеристика на двигателя OM 501 LA 2. Serie (320 kW (435 PS) / 2100 Nm) - Фиг. 2.3.1.3.



Динамичната характеристика на всеки двигател е графика, която показва каква е стойността на параметрите на двигателя с променяне на оборотите. Най-често параметрите, сравнявани с оборотите на двигателя са мощността и въртящият момент. Двигателят на Mercedes-Benz Actros MPII 1844 LS е дизелов. Тези двигатели се характеризират с по-висок въртящ момент и по-малка мощност в сравнение с еквиваленти бензинови двигатели.

Максималният въртящ момент се постига при по-ниски обороти в сравнение с бензиновите двигатели.

На динамичната характеристика на двигателя ясно се виждат пиковете на двата разглеждани параметъра – максимална мощност и максимален въртящ момент. А именно максимална мощност от 435к.с. (320KW) се постига при 1800 оборота в минута, а и максималният въртящ момент от 2100Nm при 1080 оборота в минута.

За да бъде реализиран съветник за превключване на предавките на целевия автомобил е необходима информация за скоростната кутия като цяло, както и за предавателните числа на скоростната кутия, както и за предавателното число на диференциала.

Целевият автомобил Mercedes-Benz Actros MPII 1844 LS е снабден с осемстепенна механична предавателна кутия с електронно управление и пневматично превключване на предавките. Предавките за движение напред са общо шестнадесет. Осем предавки, като всяка от тях има две половинки-предавки – бърза половинка и бавна половинка. Предавките за движение назад са две.

Предназначението на скоростната кутия на автомобила е да предава оборотите на двигателя към задвижващите колела. Предавателните числа са основна характеристика на предавателната кутия. Те представляват съотношението между на оборотите на двигателя и оборотите на карданния вал. По този начин се управлява мощността на двигателя. Предавателните числа

на скоростната кутия и диференциала са дадени на фигура 2.3.1.4.

Предавателни числа на предавките на целевия автомобил

Предавка	Предавателно число
1 – бавна	17.03
1 – бърза	14.19
2 – бавна	11.50
2 – бърза	9.58
3 – бавна	7.80
3 – бърза	6.50
4 – бавна	5.28
4 – бърза	4.40
5 – бавна	3.87
5 – бърза	3.22
6 – бавна	2.61
6 – бърза	2.18
7 – бавна	1.77
7 – бърза	1.48
8 – бавна	1.20
8 – бърза	1.00
Задна – бавна	15.48
Задна – бърза	12.90
Диференциал	2.846

Фигура 2.3.1.4

Поради факта, че целевият автомобил на дипломната работа е тежкотоварен камион, по закон неговата максимална **скорост** е ограничена на **90** км/ч. Това ограничение е реализирано чрез електронен ограничител на скоростта. При достигане на тази скорост от **90** км/ч двигателят спира да увеличава оборотите си независимо от натискането на педала на газта.

Както всеки автомобилен двигател, така и този на Mercedes-Benz Actros MPII 1844 LS има ограничител на **оборотите** с цел предотвратяване на претоварване на двигателя и предпазване от повреждането му. Максималните обороти са **2500** оборота в минута, а минималните такива, т.е. празният ход на двигателя е **500** оборота в минута.

Резервоарите с **гориво** на целевия автомобил са с обем от **1050** литра.

2.3.2. Специфична информация за начина, по който целевия автомобил измерва работните параметри.

Отделните автомобили измерват по различен начин работните параметри на автомобила и затова е особена важност да има специфична информация, която да дава яснота как основните параметри биват измервани.

В текущата дипломна работа е необходима информация за начина на измерване на :

- Скоростта на автомобила;
- Оборотите на двигателя;
- Моментният разход на гориво на автомобила;
- Количеството гориво в резервоара.

Най-често **скоростта на автомобила** се измерва от датчик в скоростната кутия, който генерира импулси с определена честота, въз основа на скоростта на въртене на изходящия вал. В случая с целевия автомобил на текущата

дипломна работа скоростта бива измервана от датчик, монтиран на скоростната кутия и генериращ импулси на базата на въртенето на карданния вал. За едно завъртане на вала се генерират четири правоъгълни импулса. За един километър се генерират **4414** импулса. Като е известна тази информация лесно може да се проектира управляващият вграден софтуер, който да измерва скоростта на автомобила на базата на генерираната от датчика поредица от правоъгълни импулси. Използваният режим на таймера е Брояч на събития (вж. т. 2.5.).

Оборотите на двигателя на целевия автомобил се измерват от датчик, който брои зъбците на маховика на коляновия вал на двигателя. На един оборот на маховика, респективно на коляновия вал, се генерират 120 правоъгълни импулса. Като се знае, тази информация лесно могат да се изчислят оборотите на двигателя на база на тази честота, постъпваща от датчика за оборотите. Използваният режим на таймера е Брояч на събития (вж. т. 2.5.).

Моментния разход на гориво се измерва по различен начин от различните автомобили, поради разликите в горивните системи и различни инженерни решения. В конкретния случай се следи броят и продължителността на впръскванията на гориво в цилиндрите. Това се извършва на цикли от по 10 секунди. При празен ход дюзите впръскват гориво за около 900 микросекунди. Използваният режим на таймера е Брояч на събития (вж. т. 2.5.).

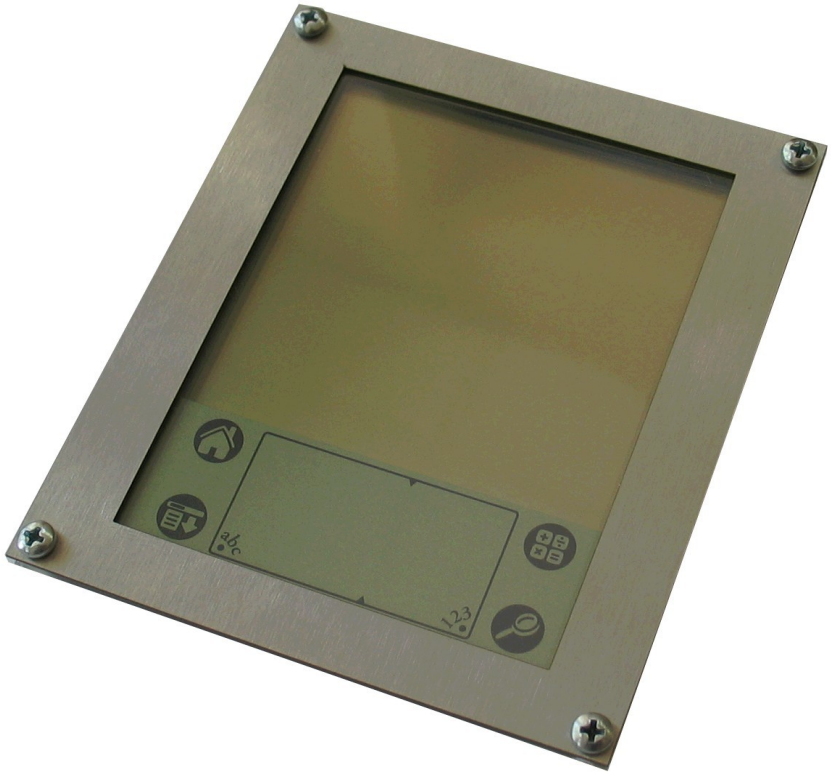
Количеството на гориво в резервоара се измерва от резистивна сонда, която променя съпротивлението си в зависимост от текущото ниво на горивото в резервоарите. На целевия автомобил резервоарите са два – един с вместимост от 450 литра и един с обем 600 литра. Общата вместимост на гориво е 1050 литра. При падане на нивото на горивото под 14% или 147 литра, се включва индикация, която показва, че горивото е малко и трябва да предприеме зареждане. Целевият автомобил може да работи с различни резистивни сонди, т.е. при поставяне на сонда с различно съпротивление от предишната трябва да се извърши параметриране на машината за минималната и максимална стойност на конкретната сонда и автомобилът започва да работи с новите стойности. Изводът е, че не са важни конкретните резистивни стойности на сондата за гориво, понеже машината може да работи с всякакви такива. Използваният метод е измерване на пада на напрежение на резистивния датчик с помощта на вградения в процесора периферен модул Analog to Digital Converter (ADC) (вж. т. 2.5.).

2.4. Детайлен преглед на развойната хардуерна платформа OLIMEX AVR TLCD128CAN

В настоящата дипломната работа за развойна хардуерна платформа е избран OLIMEX AVR TLCD128CAN.

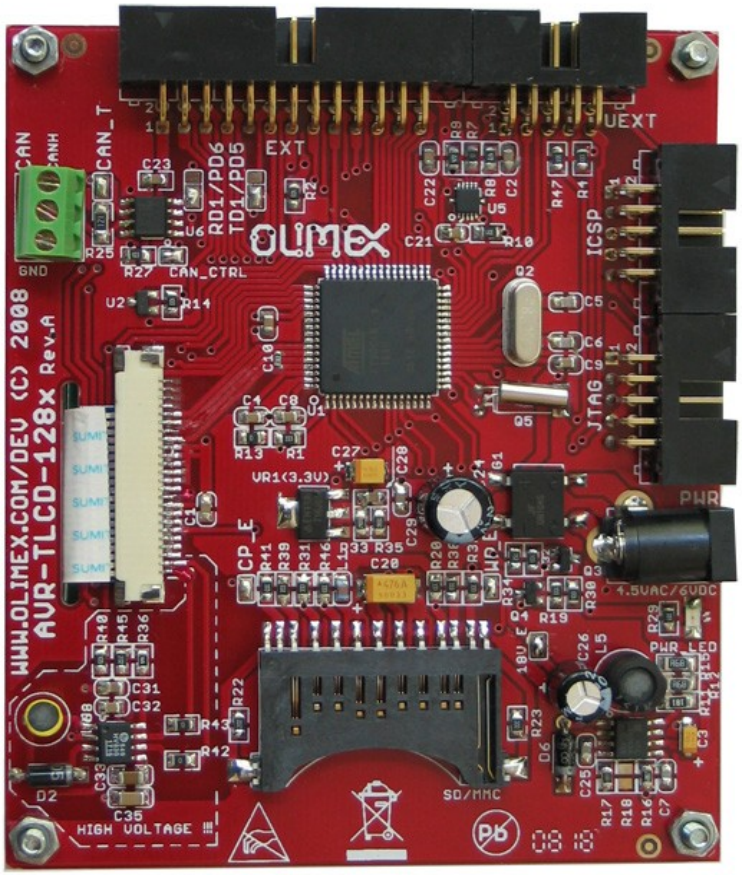
Той представлява готов за програмиране развоен хардуер , произведен от фирма OLIMEX. (вж. Използвана литература 2) В основата му е заложен осем битов процесор на Atmel Corp., а именно AVR AT90CAN128. Развойният хардуер притежава монохромен течнокристален дисплей с разделителна способност 160x160 пиксела и панел, реагиращ на допир. Програмирането може да бъде извършвано чрез JTAG интерфейс. На лице са и слот за SD/MMC карта памет, както и конектор за CAN връзка. На фигура 2.4.1. е показана снимка на развойният хардуер.

фигура 2.4.1. Развойният хардуер OLIMEX AVR



TLCD128CAN - отпред

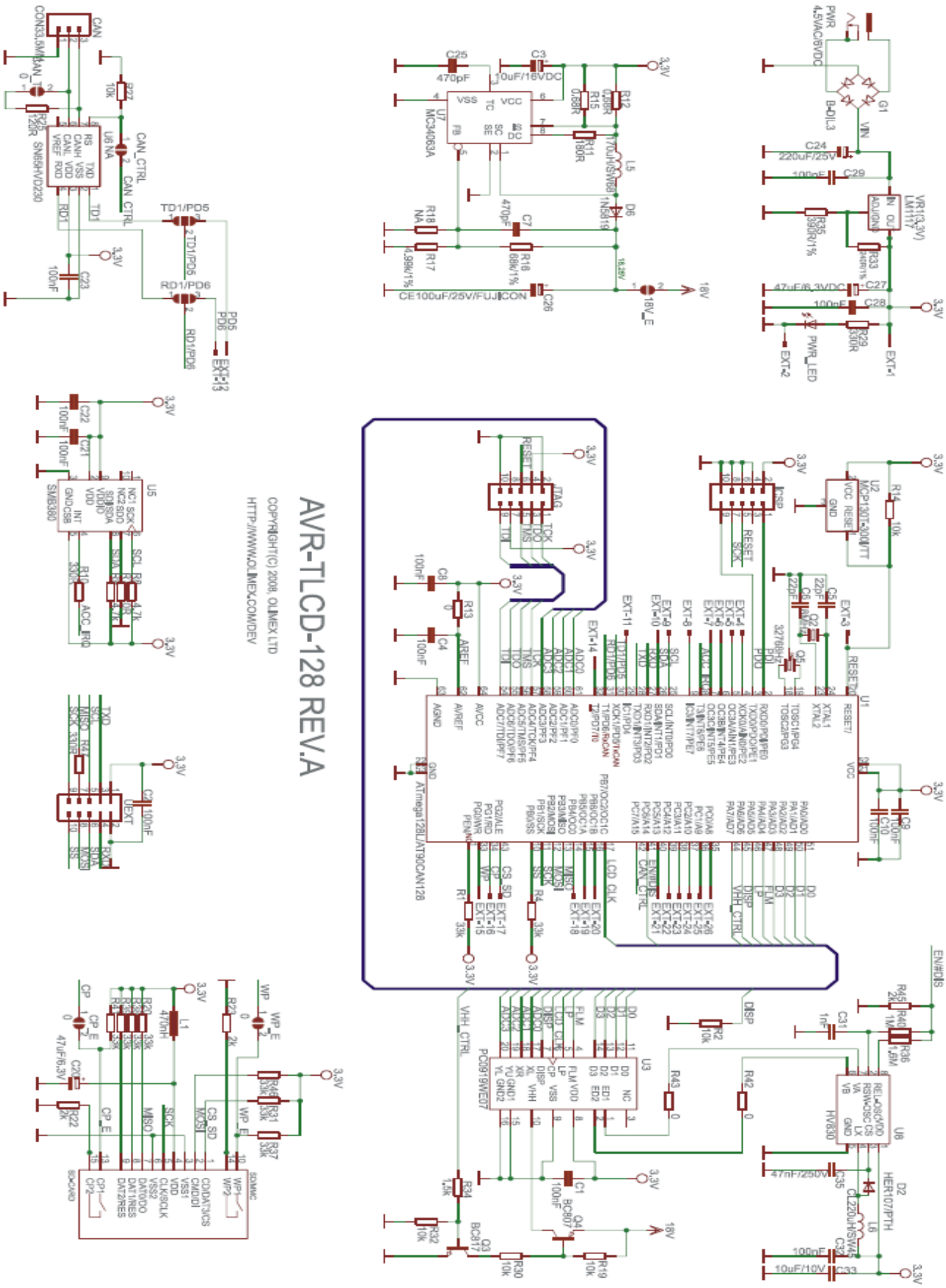
Развойният хардуер OLIMEX AVR TLCD128CAN - отзад



Процесорът на развойния хардуер е AVR AT90CAN128, производство на Atmel Corp. На фигура 2.4.2. са дадени основните параметри на микропроцесора.

Основни параметри на процесора AVR AT90CAN128 - Фиг. 2.4.2.

Параметър	Стойност
Тип	High-performance, 8-bit
Архитектура	Advanced RISC Architecture
Брой инструкции	133
Максимална честота	16MHz
Производителност	16MIPS (при 16 MHz, или 1 MIPS/1 MHz)
Flash памет(програмна)	128KB
EEPROM(енергонезависима за данни)	4KB
RAM	4KB
Други	▪ Два 8-битови таймера ▪ Два 16 битови таймера ▪ 10 битов Аналогово-цифров преобразувател ▪ CAN контролер 2.0 ▪ TWI модул ▪ PWM ▪ Два Input Capture модула ▪ Два USART модула ▪ 53 програмируеми входно-изходни пина ▪ Програмиране чрез JTAG интерфейс


AVR-TLCD-128 REV.A
COPYRIGHT(C) 2008, OLIMEX LTD
HTTP://WWW.OLIMEX.COM/DEV

Фиг. 2.4.3.

При програмирането на микропроцесорни вградени системи е необходимо софтуера да е съобразен с цялостната реализация на хардуерната платформа. На фигура 2.4.3. е показана принципната електрическа схема. На нея ясно се вижда начинът на свързване на процесора с основните периферни устройства на развойната среда като дисплеят и контактният панел например. Тя дава още информация как портовете на процесора могат да бъдат свързани към външни устройства с цел разширяване на функционалността и отчитане на външни влияния като измерване на честотата на поредица и правоъгълни импулси или преобразуване на аналогова величина в цифрова. Акселерометърът и слотът за карта памет са също показани на схемата на развойната среда.

2.5. Детайлен преглед на използваните вградени модули

В тази част на дипломната работа ще бъде описана работата с вградените модули на процесора, използвани за реализирането на отделните части на управляващия софтуер на бордовия компютър.

Ще бъдат разгледани следните вградени модули:

- Таймери
- Input Capture(IC)

- Analog to Digital Converter(ADC)
- Two-Wire Interface(TWI)

Таймерите са вградени модули на процесорите, широко използвани в управляващия софтуер на много и различни приложения. Хардуерната платформа на настоящата дипломна работа има процесор, снабден с два осембитови и с два шестнадесетбитови таймера. Осембитовите таймери могат да отброят до 255, а шестнадесетбитовите – до 65535. Когато таймерът е включен, на всеки цикъл на системната тактова честота стойността му ще бъде увеличавана. Когато таймерът достигне максималната си стойност, той се препълва и се вдига флаг в специален регистър, обозначаващ специфични ситуации относно този таймер (всеки таймери има свой такъв регистър). Ако прекъсването на конкретния таймер е разрешено, както и глобалните прекъсвания, то ще бъде генерирано прекъсване за препълването на таймера. След появата на прекъсване процесорът предава изпълнението на т.н. функция, обслужваща прекъсването, която е написана от програмиста и в нея се намират действията, които трябва да се извършат при препълването на таймера. Ако в прекъсването таймерът бъде зареден със стойност, то след изпълнението на прекъсването броенето започва от нея, в противен случай броенето започва от нула. В настоящата дипломна работа честота на процесора е 16MHz, което означава, че стойността на таймера се увеличава 16 милиона пъти в секундата. Това значи, че дори и с шестнадесетбитов

таймер ще може да бъде отброено много малко време. Затова са въведени т.н. предварителни делители на таймери (Timer Prescalers). Той служи за разделяне на системната тактова честота, за да се забави броенето на таймера и да увеличи обхвата на възможните за отброяване времеинтервали. Делителя се избира от определени стойности, дадени в спецификацията на процесора. В конкретния случай той може да бъде едно от следните числа: **1, 8, 64, 256, 1024.**

При избора на определен делител времето, отброено от един времеинтервал зависи от тактовата честота на таймера и се пресмята по формулата, дадена на фигура 2.5.1

Фигура 2.5.1.: Формула за пресмятане на времето за един времеинтервал на таймера:

TICKTIME = PRESCALER/CPUCLK

Означения:

TICKTIME – време за един времеинтервал на таймера

PRESCALER – делител на таймера

CPUCLK – честота на процесора

Основната функция на таймерите е да отброяват времето между периодично случващи се събития. Първо е необходимо да се намери каква е честотата на това събитие. За да се направи това изчисление е необходимо

да се знае периода на това събитие. Това се изчислява по формула 2.5.2.

Фигура 2.5.2: Формула за пресмятане на честотата на дадено събитие

TICKRATE = 1/PERIOD

Означения:

TICKRATE – честота на събитието

PERIOD – период на събитието

Времеинтервалите на таймера, необходими за отброяването на това времето между събитията се пресмята по формулата, дадена на фигура 2.5.3.

Фигура 2.5.3: Формула за пресмятане на броя на времеинтервалите, необходими за отброяването на дадено време

TICKCOUNT = CPUCLK/TICKRATE/PRESCALER

Означения:

TICKCOUNT – броя времеинтервали

CPUCLK – тактова честота на процесора

TICKRATE – честота на събитието

PRESCALER – делител на таймера

За отброяване на това време е необходимо таймерът да бъде инициализиран с начална стойност и до препълването му да е изминало желаното време. Тази начална стойност се изчислява по формула 2.5.4..

Фигура 2.5.4: Формула за пресмятане на началната стойност на таймера

TICKINIT = TIMERMAX – TICKCOUNT

Означения:

TICKINIT – начална стойност на таймера

TIMERMAX – максимален брой времеинтервали на таймера

TICKCOUNT – броя времеинтервали

TIMERMAX се избира спрямо **TICKCOUNT**. Ако **TICKCOUNT** е по-малко от 255 (0xFF), то **TIMERMAX** се избира 255, т.е. може да бъде използван осембитов таймер, но ако **TICKCOUNT** е по-голямо от 255, то **TIMERMAX** се избира 65535 (0xFFFF) – използване на шестнадесетбитов таймер. Ако **TICKCOUNT** е по-голямо от 65535 (0xFFFF) даденото време не може да бъде отброено с този делител на таймера.

Input Capture е режим на работа на таймера. Той се употребява най-често се употребява за измерването на честотата на поредица правоъгълни импулси.

Процесорът AVR AT90CAN има два Input Capture модула. Те са свързани с неговите шестнадесетбитови таймери. За реализирането на измерването на честота е необходимо да се измери периодът на поредицата правоъгълни импулси. Измерването става като на крачето на процесора, на което е изведен входа на Input Capture модула се свърже изходът на генератора на измерваната

честота. Тези крачета на процесора са изведени на разширителния съединител от щифтов тип на развойната хардуерна платформа. По-точно 8-и и 11-и щифт.

Модулът за Input Capture реагира при промяна на състоянието на входният сигнал. Тази промяна може да бъде от високо в ниско ниво или обратно – от ниско във високо ниво. Модулът за Input Capture може да бъде настроен да реагира на една от тези две ситуации. При такава промяна в специален регистър на модула за Input Capture се вдига флаг, който обозначава събитието. Ако прекъсването за Input Capture е включено, както и, ако флагът за прекъсванията в Статус регистъра е вдигнат, то ще бъде генерирано прекъсване за Input Capture и процесорът ще предаде изпълнение на т.н. функция, обслужваща прекъсването. Идва ред на основната роля на Input Capture – при поява на събитие, състоянието на таймера, към който работи конкретният Input Capture се dакопира в отделен регистър на Input Capture модула. Тоест така се знае точното време, в което е възникнало събитието. При запазване на тази стойност от програмиста и при появата не следващото събитие, състоянието на таймера отново бива запазено в специалният регистър на Input Capture модула и може да бъде определена разликата между двете показания. Тази разлика е времето между възникването на двете събития. То това време не е време в секунди, а представлява брой на времеинтервалите на таймера. При използването на: формулата за пресмятане на времето за един времеинтервал на таймера, дадена на

фигура 2.5.1. лесно може да се намери реалното време между възникването на двете събития. То всъщност представлява периода на поредицата от правоъгълни импулси. Честотата на тези импулси може да бъде намерена по-формулата, дадена на фигура 2.5.5..

Фигура 2.5.5.: Формула за изчисляване на честота:

FREQUENCY = 1/PERIOD

Означения:

FREQUENCY – честота

PERIOD – период

Ако прекъсванията се броят, то след определен период от време, в който е извършвано броенето, се намира броя импусли за този отрязък от време, което е честотата на поредицата правоъгълни импулси. Този режим се нарича Брояч на събития.

Броячът на събития е използван в настоящата дипломна работа за реализиране на измерването на честота, показваща скоростта, оборотите и моментният разход на гориво.

Analog to Digital Converter (ADC) е много често използван модул при разработката на вградени микропроцесорни системи. Той позволява да бъдат преобразувани аналогови величини в цифрови. Такава величина е например напрежение, което позволява на микропроцесора да измерва пада върху резистивен делител на напрежение.

Измерването на напрежение чрез използването на Аналогово-цифровият преобразувател (**АЦП**) се извършва чрез свързване на пина на АЦП към електрическата верига, в която ще бъде извършвано измерването на напрежение. Измерваното напрежение е потенциалът спрямо „нулата”, която е сигналната верига GND на електрическата схема на възелът, в който се измерва напрежението, и трябва да се свърже към масата на процесора. Пиновете на АЦП са осем, но АЦП модулът е само един и затова е използван вграден в процесора мултиплексор, който свързва един от тези осем канала към модула. Преди започване на измерването трябва да се настрои кой канал да бъде свързан към АЦП модула чрез мултиплексора.

На процесора, избран за настоящата дипломна работа АЦП е десет-битов. Това означава, че преобразуваната стойност след работата на АЦП е с размер десет бита, т.е. максималната стойност при преобразуването е 1023. При използването на АЦП е нужен избора на т.н. опорно напрежение (Reference Voltage - **RV**). Това е напрежението, при което преобразуваната стойност от АЦП би била максимална. Тоест, ако RV е избрано захранващото напрежение (в конкретния случай 3.3V), при измерване на напрежение 3.3V от АЦП, преобразуваната стойност ще бъде 1023. Измерената стойност е в правопропорционална зависимост от измерваното напрежение – при промяна на това напрежение се променя и получената стойност в АЦП регистрите. Тези регистри могат да се използват от софтуера. При завършване на преобразуването в

специален регистър на АЦП се вдига флаг, който показва събитието край на преобразуването. Ако е разрешено прекъсването на АЦП, както и глобалните прекъсвания ще бъде генерирано прекъсване за края на преобразуването и процесора ще пренасочи изпълнението към т.н. функция, обслужваща прекъсването.

Така чрез АЦП се реализира връзка между чисто хардуерна величина, каквато е напрежението и вградения управляващ софтуер на микропроцесора.

Този метод е използван в настоящата дипломна работа за реализирането на работата на панелът, реагиращ на допир, както и за измерване на количеството гориво в резервоара.

Two-Wire Interface (TWI) е един от начините за серийна комуникация между процесора и периферни устройства, като датчици, дори и други процесори. Общият брой на устройствата може да бъде 128, поради седем-битовата адресация на TWI. Технологията е изобретена и стандартизирана от Philips. Още се нарича **I2C**. Името идва от това, че комуникацията се извършва по две шини, за които се свързват всички устройства, участващи в комуникацията. Тези две шини са **SDA** и **SCL**. **SDA** е за данните, предавани по шината, а **SCL** – за тактова честота. По стандарт честотата на комуникацията е между 100 и 400 kHz. Тази честота се изчислява по формулата, дадена на фигура 2.5.6.

Фигура 2.5.6.: Формула за пресмятане на честотата на SCL.

$$SCLFREQ = \frac{CPUCLK}{16 + 2(BITRATE) * 4^{PRESCALER}}$$

Означения:

SCLFREQ – честота на SCL

CPUCLK – честотата на процесора

BITRATE – стойността на BITRATE регистъра на TWI модула

PRESCALER - стойността на PRESCALER регистъра на TWI модула

Стойностите на BITRATE и PRESCALER в тази формула трябва да бъдат така подбрани, че стойността на SCL честотата да бъде в стандартизираните граници от 100 до 400 kHz.

Трябва да бъдат въведени следните понятия, отнасящи се до TWI:

- MASTER – УПРАВЛЯВАЩ - устройството, което започва предаването, прекъсва го и задава честотата на SCL
- SLAVE – ПОДЧИНЕН - устройството, адресирано от MASTER-а
- TRANSMITTER – устройството, което изпраща данни по шината

- RECEIVER – устройството, което чете данни от шината.
- START – започване на комуникацията
- STOP – прекратяване на комуникацията
- REPEATED START – започване на нова комуникация без загуба на контрол над шината.
- ACK – потвърждение
- NACK - **НЕ**потвърждение

TWI е сериен протокол за комуникация между свързаните устройства и данните по шината се предават последователно – от TRANSMITTER към RECEIVER.

Когато TWI модулът е включен, той поема контрола върху пиновете SDA и SCL на процесора. Преди включването това са били обикновени входно/изходни пинове на процесора. Модулът се грижи за честотата на SCL, изпълнява командите от софтуера за изпращане и получаване на данни по шината. При всяка промяна на статуса на TWI шината се генерира прекъсване на TWI и процесорът прехвърля изпълнението на функцията обработваща прекъсването. В тази функция трябва да се провери статуса на шината, спрямо който да се вземе решение какво действие да се предприеме. Прекъсване ще се генерира при една от следните ситуации:

- TWI модулът е изпратил START/REPEATED START;
- TWI модулът е изпратил адрес на SLAVE + READ/WRITE бит;
- TWI модулът е получил байт данни;

- TWI модулът е загубил контрол върху шината;
- TWI модулът е адресиран от УПРАВЛЯВАЩ;

За да се извърши комуникация между **управляващ** и **подчинен**, **управляващият** трябва да изпрати т.н. START по шината. След това **управляващият** трябва да изпрати по шината седем-битовия адрес на **подчинения**, с когото иска да комуникира, последвано от READ/WRITE бит, който определя дали **управляващият** ще получава информация от **подчинения** или ще му изпраща такава. Когато **подчинения** бъде адресиран от **управляващия** той трябва да му изпрати потвърждение(ACK), че е свободен (не участва в комуникация с друг **управляващ**) и комуникацията може да бъде извършена. Ако **управляващият** е изпратил READ след адреса на подчинения, то **управляващият** е в режим MASTER-RECEIVER, а ако е изпратил WRITE, то той е в режим MASTER-TRANSMITTER. След обмена на тази служебна информация започва същинското предаване на данните, между **управляващия** и **подчинения**.

Ако режимът е **MASTER-TRANSMITTER**, то след получаване на потвърждение(ACK) от **подчинения**, **управляващият** може да започне да изпраща байтове с данни до **подчинения**. След всеки получен байт данни, **подчиненият** трябва да изпрати потвърждение(ACK) на **управляващия**. Когато **управляващият** иска да прекрати предаването на данни изпраща STOP към **подчинения**. Когато иска да смени режима си на работа, изпраща

REPEATED START, последвано от адреса на **подчинения** и READ бит.

Ако режимът е **MASTER-RECEIVER**, то след получаване на потвърждение от **подчинения**, **управляващият** може да започне да приема байтовете данни, изпращани му от **подчинения**. След всеки получен байт информация **управляващият** трябва да изпрати потвърждение (ACK), което би означавало продължаване на предаването на още байтове данни или **НЕ**потвърждение (NACK), което значи, че **управляващият** иска да прекрати приемането на информация. След това **управляващият** може да предаде по шината STOP за прекратяване на комуникацията или REPEATED START, последван от адреса на **подчинения** и WRITE бит за смяна на режима на работа.

След предаване/приемане на данните и предаване на STOP, **управляващия** губи контрол над TWI шината. При предаване на REPEATED START, **управляващият** не губи контрол на шината, въпреки, че сменя режима си на работа – от RECEIVER в TRANSMITTER или обратно.

Всяко едно от този действия се извършва във функцията, обслужваща прекъсването на TWI модула. В началото на всяко извикване на тази функция трябва да се провери статусът на TWI шината и да се вземе решение как да се продължи комуникацията, в зависимост от режима на управляващия.

Комуникацията чрез TWI е използвана в настоящата дипломна работа за реализирането на комуникацията с температурният датчик, измерващ външната температура.

Глава III

В трета глава на настоящата дипломна работа ще бъде направен обзор на изпълнението на задачите по създаването на бордовия компютър. В същинската част ще бъде разгледана в детайли реализацията на вградения управляващ софтуер.

3.1. Обзор

Вграденият управляващ софтуер на бордовия компютър е разделен на модули. Всеки от тези модули е отговорен за реализирането на отделна функционалност. Тази функционалност може да е директно обвързана с изискванията, т.е. да се отнася до конкретен параметър на пътуването, но може задачите, за които отговаря модулът да имат спомагателна роля при реализирането на останалите функционалности, като например модулът за течнокристалният екран или панелът, реагиращ на допир.

Следва списък на отделните модули:

- Модул на LCD екрана;
- Модул на панела, реагиращ на допир;
- Модул на часовника и календара;
- Модул за температурата
- Модул за количеството гориво в резервоара
- Модул на предавките, скоростта и оборотите

- Главен модул

3.2. Същинска част

В тази част на трета глава на настоящата дипломна работа ще бъдат разгледани конкретните реализации на всеки един от модулите на вградения софтуер на бордовия компютър. За всеки модул ще бъде изяснено какъв хардуер се управлява, както и детайлно описание на начина на реализиране на функционалността, за която е отговорен модулът.

3.2.1. Модул на LCD екрана

LCD екранът е основната индикация, с която е снабдена развойната хардуерна платформа и модулът, отговарящ за него е от особена важност.

Функционалност:

- Реализиране на функции, отпечатващи символи по екрана
- Реализиране на функции, отпечатващи иконки по екрана

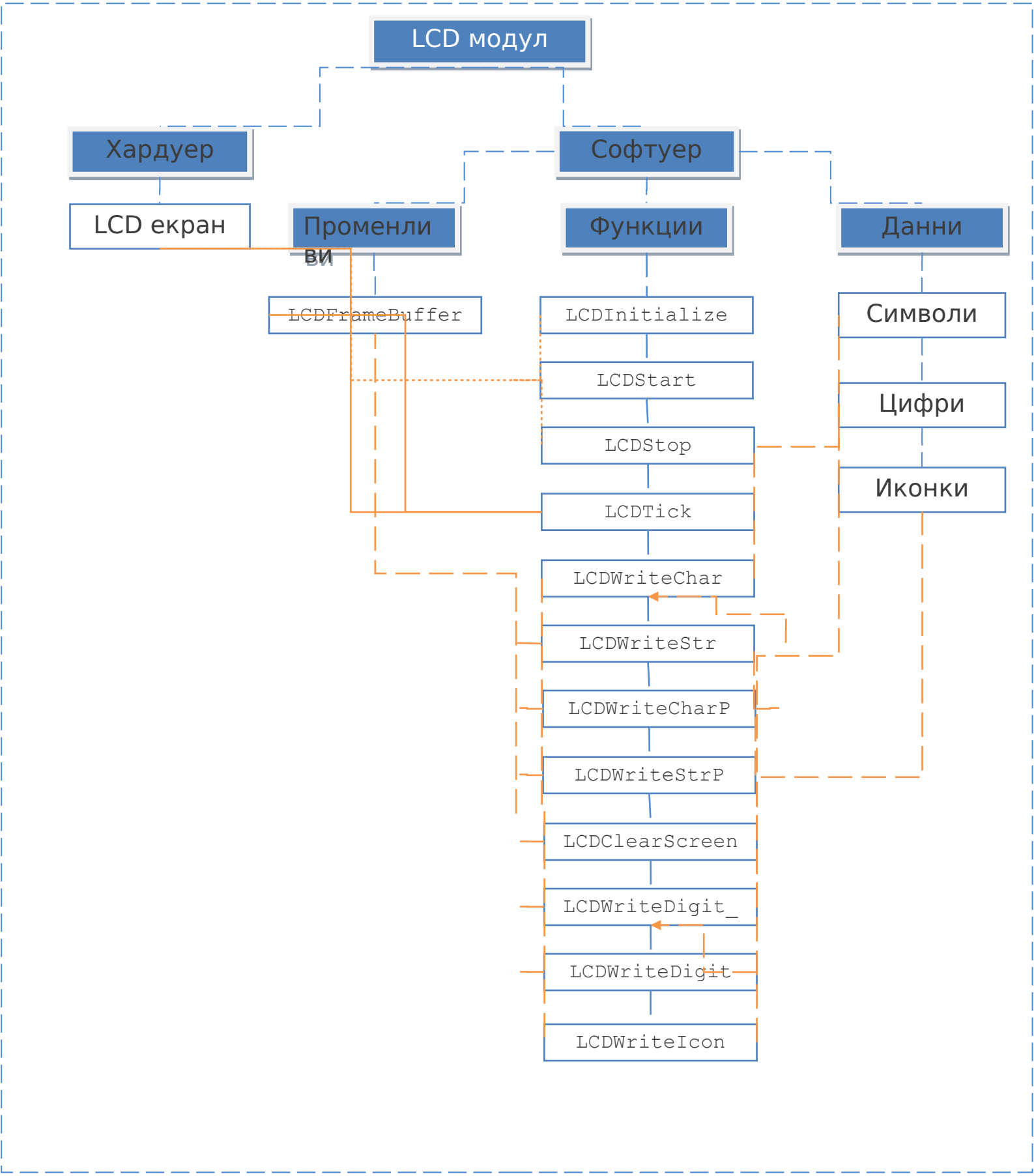
Използван хардуер:

- LCD екран PC0919WE07 от Palm III
- PORTA , PORTB7 и PORTC6 на процесора

Файлове с управляващия софтуер

- lcd_driver.h
- lcd_driver.c
- arrays.h
- flash_data.h
- font_8x8.h

Фигура 3.2.1.1: Модел на LCD модула


LCD модул
Хардуер
Софтуер
LCD екран
Променливи
Функции
Данни
LCDFrameBuffer
LCDInitialize
LCDStart
LCDStop
LCDTick
LCDWriteChar
LCDWriteStr
LCDWriteCharP
LCDWriteStrP
LCDClearScreen
LCDWriteDigit_
LCDWriteDigit
LCDWriteIcon
Символи
Цифри
Иконки

Описание на модела на LCD модула

На фигура 3.2.1.1. е показан моделът на LCD модула. На него ясно се виждат основните клонове на които се разделя той. Поради факта, че става дума за вграден управляващ софтуер, от значение е и това на какъв хардуер се работи, затова освен софтуерната част на модула има и хардиерна такава.

Хардуер

Екранът, използван в настоящата дипломна работа е с означение PC0919WE07. Това е 3.5” течнокристален дисплей, използван в Palm III устройствата (вж. Използвана литература 2). Разделителната му способност е 160x160 пиксела, което означава, че има общ брой от 25 600 пиксела. Екранът е монохромен и има подсветка (backlight). Екранът се управлява от процесора посредством всички осем пинове на PORT A, чрез пин 7 от PORT B и пин 6 от PORT C. Информацията се предава на екрана чрез първите четири пина от PORT A, а чрез останалите пинове се изпраща управляваща информация.

Софтуер

На модела на LCD модула е дадено разделението на софутерната част от реализацията на модула:

- Променливи – това е т.н. `LCDFrameBuffer`, който представлява масива от байтове, представящ екрана софтуерно;
- Данни – това са различните символни данни като цифри, букви и други символи

- Функции – реализират задачите, изпълнявани от модула, като отпечатването на символи и иконки по екрана. Те реализират връзката между отделните софтуерни клонове, както и връзката хардуер-софтуер в модула.

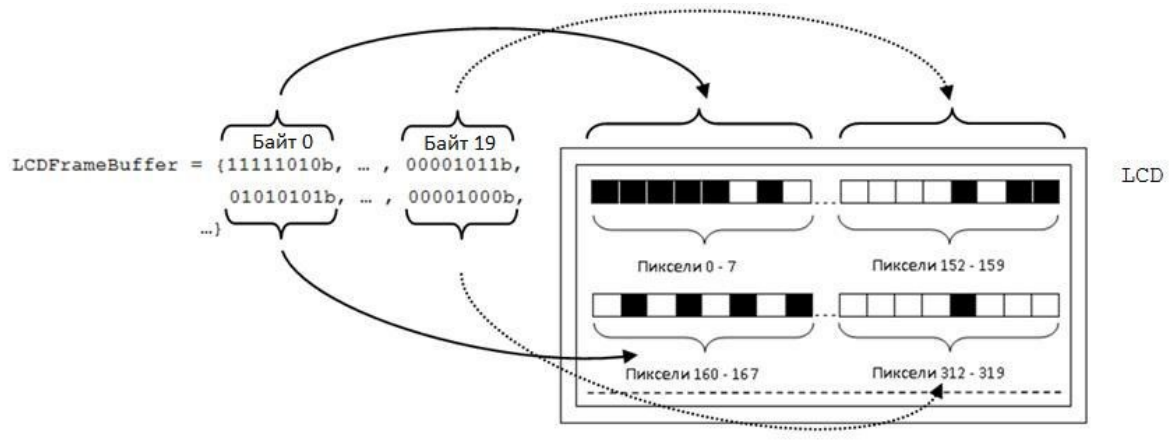
Променливи

```
extern U8 LCDFrameBuffer[LCD_BUFFER_SIZE];
```

Екранът се представя в паметта чрез масив от байтове. Размерът му е равен на броя пиксели на екрана.

```
//*****************************************************************************
// LCD dimensions
//*****************************************************************************
#define LCD_HEIGHT      160
#define LCD_WIDTH       160

#define LCD_WIDTH_BYTES 20    // = LCD_WIDTH / 8
#define LCD_BUFFER_SIZE 3200  // = LCD_WIDTH_BYTES*LCD_HEIGHT
```

На показания фрагмент код е даден начина на образуване на размера на екрана. Ширината на екрана е 160 пиксела. Тоест броя байтове на един ред от него е 160/8 = 20 байта. Редовете на екрана са също 160 пиксела, т.е.на всеки от тези 160 реда има по 20 байта, което определя и размера на масива от 3200 байта. Всеки бит във всеки от байтовете на масива, представлява един пиксел от екрана. Поради факта, че екранът е монохромен съществуват само две състояния на пикселите – засветено



и изгасено. Ако битът, представящ даден пиксел е в състояние единица, то съответният пиксел ще бъде в светнато състояние.

Фигура 3.2.1.2. Връзка между масива с байтове и пикселите на екрана

На фигура 3.2.1.2 се вижда връзката между масива и екрана. Нулевият байт в масива съответства на първите осем пиксела от екрана, т.е. пикселите, намиращи се най-горе в ляво. Последователно пикселите в дясно се запълват спрямо следващите байтовете в масива. За състоянието на един ред пиксели отговарят 20 байта, т.е. следи отпечатването на байтовете от 0 до 19 от масива на екрана, започва отпечатването на следващия ред пиксели от ляво надясно с байтове 20 – 39. Тоест с един байт от масива, се опрделя състоянието на осем пиксела от екрана в хоризонтална посока.

Данни

Данните в реализацията на LCD модула представляват всички символни данни, коит могат да бъдат отпечатани по екрана от функциите за работа с него. Това са всички цифри, букви и други символи, както и иконки, необходими за визуализирането на работните параметри на целевия автомобил, както и на интерфейса за работа с бордовия компютър. Видовете данни, дадени на фигура 3.2.1.2 са следните:

Символи – тези данни се състоят от 96 симовола. Тези символи включват следните:

- Числа: 0-9;
- Букви: A-Z; a-z;
- Други: „ , !“#$%&‘()*+,-./:;<=>? @[]\^_{}~”?

```
// font sizes
// have to be exact multiple of 8.
#define FONT_CHAR_WIDTH         8
// can be any value
#define FONT_CHAR_HEIGHT        8

// LCD dimensions for 8x8 fixed font chars
#define LINE_NUM_CHARS          20
#define NUM_ROWS                20
```

С този фрагмент код се задават размерите на тази символи, а именно осем пиксела височина и също осем пиксела ширина. Ширината трябва да бъде число, кратно на осем, за да може символът да бъде във форма, удобна за работа с `LCDFrameBuffer-a`. Поради тези размери, броя на символите, които могат да се разположат на един ред и на една колона на екрана са 20. (160 пиксела ред/колона делено на 8 пиксела за всеки символ).

```
#define FONT_START_CHAR   0x20
```

Символите са подредени по реда им в ASCII таблицата, като първият от тях е интервал. Той е с номер 32 – затова е и горната дефиниция.

Символите са представени като масиви, съставящи един обща масив, съдържащ всичи символи.

```
// font definition
const U8 Font8x8 [0x60][8] PROGMEM =
{
      {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, // sp
      {0x04,0x04,0x04,0x04,0x00,0x00,0x04,0x00}, // !
      {0x0A,0x0A,0x0A,0x00,0x00,0x00,0x00,0x00}, // "
      {0x0A,0x0A,0x1F,0x0A,0x1F,0x0A,0x0A,0x00}, // #
```

```
...
```

Това е дефиницята и инициализирането на масива `Font8x8 [0x60][8]` от файла **`font_8x8.h`**. При дефинирането му е използван атрибутът `PROGMEM`, който оказва масивът да бъде разположен в паметта на процесора, предназначена за програмния код на вградения управляващ софтуер, което спестява от оперативната памет. Масивът е двимерен, като по първото му измерение с размер 96(броя на символите) са разположени различните символи, а по второто самите символи с размер 8(височината на символите). Масивът е инициализиран с шестнадесетични стойности. Това се прави поради по-лесното отпечатване в сравнение с двоичните – една шестнадесетична стойност съответства на 4 двоични. Стойността **`0xFF = 0b11111111`.** Всяка една от тези шестнадесетични стойности представлява осем пиксела, т.е. един ред от самия символ. Чрез подреждането на тези стойности една под друга в `LCDFrameBuffer-a` се получава символът, изписан на екрана.

Цифри – това са символи, с размер, по-голям то този на цифрите от предишната точка. Представляват два комплекта цифри с два различни размера:

- 24x33 пиксела: 0-9;
- 16x21 пиксела: 0-9;

И двата комплекта са оформени като седемсегменти цифри поради стилистични причини.

Комплектите са разположени в два масива – един за малкия комплект и един за големия комплект, дефинирани със файла `arrays.h`

- Голям комплект - `const U8 BigDigit[10][99] PROGMEM`
- Малък комплект - `const U8 SmallDigit[10][42] PROGMEM`

При дефинирането им е използван атрибутът `PROGMEM`, който оказва масивът да бъде разположен в паметта на процесора, предназначена за програмния код на вградения управляващ софтуер, което спестява от оперативната памет. Масивът е двумерен като по първото направление се намира съответната цифра, а по второто се намират всички байтове от самата цифра.

Размерите на тези цифри са описани в друг масив, съдържащ само общият размер на цифрите в байтове и ширината на цифрата.

```
const U16 SizeWidthValues[18][2] PROGMEM = {
            {99, 3},  //BIG_DIGIT
            {42, 2},  //SMALL_DIGIT
```

Първия елемент е размери на цифрата в байтоте, а втория ширината на цифрата отново в байтове. Тези стойности се достигат чрез следните дефиниции за успеснение.

```
#define SIZE    0
#define WIDTH   1
```

При дефинирането на масива с размерите също е използван атрибутът за разполагането му в паметта за код, а не в оперативната – `PROGMEM`.

При използването на цифри от големия/малкия комплект, на определената позиция в `LCDFrameBuffer-a` се

записват байтовете от масива за определената цифра. След записването на три/два байта (ширината на цифрата от големия/малкия комплект) се минава на нов ред и така след отпечатването на всички байтове от цифрата, тя се вижда на екрана.

Иконки – това са всички символни дани като бутони, по-специални символи, които не влизат в някоя от горите две категории, но са необходими за визуализирането на работните параметри и за интерфейса на бордовия компютър.

Иконките, използвани в настоящата дипломна работа са:

- Индикация за икономичен режим на съветника за предавките
- Индикация за динамичен режим за съветника за предавките
- Индикация за бърза половника на предавка
- Индикация за бавна половинка на предавка
- Индикация за съвет „Превключи нагоре”
- Индикация за съвет „Превключи надолу”
- Индикация за неутрална предавка
- Индикация за задна скорост
- Горивна колонка
- Линия
- Бутон OK
- Бутони – стрелки
- Минус

- Градус Целзий

Иконките са разположени в масиви, дефинирани във файла arrays.h. Всяка иконка се намира с свой отделен масив от байтове, поради факта, че иконките имат различни размери.

```
// Fuel tank
const U8 array10[42] PROGMEM = {

      0x00,0x00,
      0x7F,0xC0,
      0x7F,0xC0,
      0x60,0xF0,
      0x60,0xF8,

      ...
```

Това е дефиницията на масива за иконката за горивната колонка. Използване атрибутът `PROGMEM`, за спестяване на оперативна памет. Рамерър на масива на определанта иконка, както и ширината на иконката в байтове се дава в масива `SizeWidthValues[18][2] PROGMEM`.

Първият елемент е размери на цифрата в байтоте, а вторият - ширината на цифрата. Тези стойности се достигат чрез следните дефиниции за улеснение.

```
#define SIZE    0
#define WIDTH   1
```

Чрез използването на определена иконка в `LCDFrameBuffer-a`, се поставят на определена позиция байтовете на иконката. Чрез намиране на ширината на иконката се определя колко байта да бъдат поставени на текущия ред преди да е необходимо преминаване на следващия ред.

Функции

Функциите изграждат интерфейса за работа с LCD модула. Чрез него се извършват следните операции:

- Реализиране на функционалността на LCD модула
- Реализиране на връзката между хардуера и софтуера на модула

Функциите, са декларирани във файла lcd_driver.h, а са имплементирани във файла lcd_driver.c.

```
//*****************************************************************************
// Function prototypes
// for description of action and parameters see corresponding function in
// lcd_driver.c
//*****************************************************************************

void LCDTick            (void);
void LCDInitialize      (void);
void LCDStart           (void);
void LCDStop            (void);
void LCDWriteChar       (U8 x, U8 y, U8 ch,        U8 inv);
void LCDWriteStr        (U8 x, U8 y, U8 * str,     U8 inv);
void LCDWriteCharP      (U8 x, U8 y, U8 * ch_ptrP, U8 inv);
void LCDWriteStrP       (U8 x, U8 y, U8 * strP   , U8 inv);
void LCDClearScreen     (void);
void LCDCopyScreen      (void);
void LCDWriteDigit_     (U8 x, U8 y, U8 digit_type, U8 digit);
void LCDWriteDigit      (U8 x, U8 y, U8 digit_type, U16 digit, U8
                        pos_count);
void LCDWriteIcon (U8 x, U8 y, U8 icon);
```

Показания фрагмент код показва списък на декларациите на функциите. Те реализират отпечатването на символи, цифри и иконки по екрана, както и връзката с хардуера на екрана.

Инициализиране на екрана - `void LCDInitialize (void);`

Инициализира екрана за работа. Това се случва, чрез настойката на пиновете, свързващи процесора с екрана.

Пиновете се настройват като изходи и се инициализират с логическа нула. Само сигналът **CP** остава във състояние на логическа единица.

Функцията не приема параметри и не връща резултат

```
void LCDInitialize (void)
{
	// say LCD is not initialized during initialization sequence
	LCDInitialized = 0;

	// Initialize the peripherals:
	// CP signal inactive
	LCD_CP_HIGH();
	// CP pin: make it output -- sbi (DDRB,PORTB7); or DDRB = DDRB |
	1<<DDB7;
	sbi (mCat2(DDR,LCD_PORT_CP),mCat3(PORT,LCD_PORT_CP,LCD_PIN_CP));

	// Data port: make all 8 pins outputs
	mCat2(DDR,LCD_PORT) = 0xFF;
	// Data port + some control signals (macros not used for single
	instruction)
	mCat2(PORT,LCD_PORT) = 0;

	// EN signal
	LCD_EN_LOW();
	// EN pin: make it output
	sbi (mCat2(DDR,LCD_PORT_EN),mCat3(PORT,LCD_PORT_EN,LCD_PIN_EN));
}
```

Всички сигнали на процесора, отговарящи за работа с екрана са дефинирани по следния начин

```
	// LCD control: CP pin
	#define LCD_PORT_CP		B
	#define LCD_PIN_CP		7
```

Това позволява лесното използване на този код на друг процесор. Само трябва да бъдат подменени името на порта и номера на крачето, отговарящи за съответния сигнал и кода ще работи отново.

За всеки от сигналите са дефинирани макроси за смяна на състоянието на пина: логичска нула или логическа единица. Пример за това е следния фрагмент код:

```
#define LCD_CP_HIGH()		\
	sbi(mCat2(PORT,LCD_PORT_CP),mCat3(PORT,LCD_PORT_CP,LCD_PIN_CP))
```

Макросът променя състиоянието на пина в логическа единица, а следващият – в логическа нула

```
#define LCD_CP_LOW()        \
     cbi(mCat2(PORT,LCD_PORT_CP),mCat3(PORT,LCD_PORT_CP,LCD_PIN_CP))
```

Поради използваният прийом за дефиниране на пиновете на сигналите, са използвани следните макроси, които конкатенират(сливат) тези дефиниции. Създадени са макроси за сливане на два или три аргумента:

```
// macros for macro concatenation of 2 arguments
#define mCat2_(x,y)   x##y
#define mCat2(x,y) mCat2_(x,y)

// macros for macro concatenation of 3 arguments
#define mCat3_(x,y,z) x##y##z
#define mCat3(x,y,z) mCat3_(x,y,z)
```

В единият макрос се извиква другия поради факте, че при извикване само на единия ще бъдат конкатенирани не стойностите на аргументите, а техните имена.

```
// say LCD is not initialized during initialization sequence
LCDInitialized = 0;
```

Този фрагмент, се използва, за да се предотврати активирането на функцията за опресняване на екрана преди самия екран да бъде инициализиран.

Стартиране на екрана - `void LCDStart (void);`

С тази функция екрана се стартира. Извършват се важни операции като нулиране на масива, представящ екрана в паметта, както и разрешаване на функцията за опресняване. Функцията не приема параметри и не връща резултат.

Нулирането на масива:

```
// Clear the frame buffer
      memset (LCDFrameBuffer, 0, sizeof(LCDFrameBuffer));
```

Разрешаване на фунцията за опресняване на екрана:

```
// Initialize refresh function
      RowCounter = 0;

// Clear the frame buffer
      memset (LCDFrameBuffer, 0, sizeof(LCDFrameBuffer));
```

Поради факта, че опресняващата функция отпечатва по един ред на екрана на всяко извикване, в стартиращата функция се задава редът за отпечатване да бъде нулевият.

```
// Set VHH pin to high
LCD_VHH_HIGH();

// wait the delay:
AvrXDelay(&LCDInitTimer,mConvertMsToTicks(LCD_INIT_DELAY));
// Set DISP pin to high
LCD_DISP_HIGH();

// wait the delay:
AvrXDelay(&LCDInitTimer,mConvertMsToTicks(LCD_INIT_DELAY);
```

Функцията работи с хардуера и в нея сигналите `VHH` и `DISP` преминават в състояние на логичеса единица, което активира екрана. Между тези промени трябва да бъде направена пауза, за да може хардуера на екрана да реализира командата на софтуера. Това се извършва чрез функцията, предоставена от операционната система `AvrXDelay`() (вж. т.2.2.)

Спиране на екрана - `void LCDStop (void);`

С тази функция екрана се спира. В него се извършват противоположни действия на тези, извършвани в стартиращата функция, а именно – сигналите `VHH` и `DISP` преминават в състояние на логическа нула и се спира функцията за опресняване на екрана – екрана не трябва да бъде опресняван, докъто е спрян.

Функцията не приема параметри и не връща резултат.

```
void LCDStop (void)
{
	// say LCD is not initialized (stops the refresh)
	LCDInitialized = 0;

	// wait the delay:
	AvrXDelay(&LCDInitTimer,mConvertMsToTicks(LCD_INIT_DELAY));

	// Set DISP pin to low
	LCD_DISP_LOW();

	// wait the delay:
	AvrXDelay(&LCDInitTimer,mConvertMsToTicks(LCD_INIT_DELAY));

	// Set VHH pin to low
	LCD_VHH_LOW();

	// wait the delay:
	AvrXDelay(&LCDInitTimer,mConvertMsToTicks(LCD_INIT_DELAY));
	}
```

Отново е налице паузата след отделните команди към хардуера. Изчакването е в размер на един системен цикъл. Това се вижда от дефиницията:

```
#define LCD_INIT_DELAY     1
```

Опресняване на екрана - `void LCDTick (void);`

Функцията, отговорна за опресняване на екрана. С нейното използване се реализира връзката между софтуера – масива, представящ екрана и хардуера на екрана. При едно нейно извикване се опреснява един ред от екрана, т.е. за 160 извиквания целият екран е опреснен. Функцията се извиква на всеки 100 микросекунди в прекъсването на осем-битовия таймер нула на процесора. Тоест на всеки 16 милисекунди се опреснява целия екран. Целия екран се опреснява 62 пъти в секунда. Работата с екрана е най-ресурсоемкта част от цялата разработка – тя

заема 33% от процесорното време и 78% от оперативната памет.

Функцията не приема параматри и не връща резултат. На фрагмента код е показано тялото на функцията за опресняване на екрана.

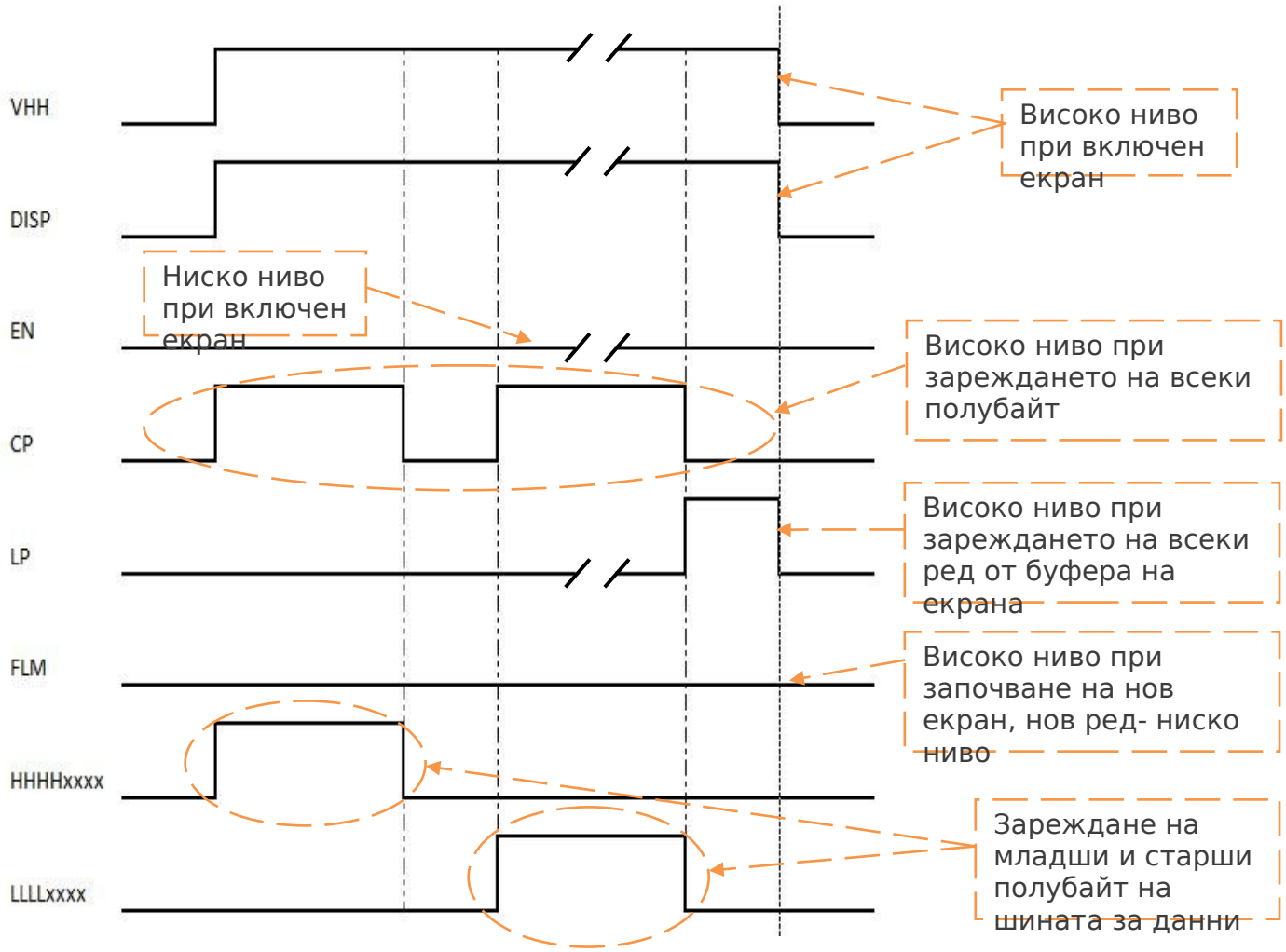
```
void LCDTick (void)
{

  U8  j;
  U8  b;

    if (LCDInitialized != 0)
    {

      for(j=0; j<LCD_WIDTH_BYTES; j++)
      {
        // get current byte
        b = LCDFrameBuffer[RowCounter+j];

        // set CP strobe to high
        LCD_CP_HIGH();

        // send high-order nibble HHHHxxxx on the data bus
        mCat2(PORT,LCD_PORT) = (mCat2(PORT,LCD_PORT) & 0xF0) | (b >> 4);

        // set CP strobe to low (latch data into LCD row buffer)
        LCD_CP_LOW();

        // set CP strobe to high
        LCD_CP_HIGH();

        // send low-order nibble xxxxLLLL on the data bus
        mCat2(PORT,LCD_PORT) = (mCat2(PORT,LCD_PORT) & 0xF0) | (b & 0x0F);

        // set CP strobe to low (latch data into LCD row buffer)
        LCD_CP_LOW();
      }

      //Latch the sent row into LCD
      LCD_LP_HIGH();

      // when running on 16 MHz, pulse is too short without some delay
      asm volatile ("nop\n");

      LCD_LP_LOW();

      // remove the new frame signal if it was set
      LCD_FLM_LOW();
```

```
      RowCounter += LCD_WIDTH_BYTES;
      if (RowCounter >= LCD_BUFFER_SIZE)
      {
        RowCounter = 0;
        // set FLM (new frame) signal to active
        LCD_FLM_HIGH();
      }
    }
  }
```

Функцията започва с проверка дали екранът е инициализиран, с цел да се избегне нежелантана ситуация екранът да се опреснява без да е инициализиран и стартиран. Поради факта, че изпращането на данни към екрана се извършва по четири шини, то се налага един байт да бъде разделен на две половини, за да бъде изпратен. Преди изпращането на всяка половина от байта сигналът `CP` преминава във високо ниво. След поставянето на половината от байта на шините за комуникация, този сигнал се поставя в ниско ниво, за да може информацията да бъде изпратена към буфера за редове на екрана. След попълването на байтовете от целия ред, с вдигането на сигнала `LP` във високо ниво, се изпраща реда от буфера на самия екран. За синхронизация, т.е. изчакване целия ред да бъде пренесен от буфера на екрана, е използвана асемлберна инструкция `NOP`. След това сигналът `LP` отново е свален в логическа нула. Ако е възникнал сигнал за нов фрайм(започване на опресняването от първи ред на екрана) той се нулира. Броячът на редовете, т.е. показателят за опреснения ред се увеличава с брой байтове на един ред, за да бъде готов за следващото извикване на функцията и се проверява дали вече не е опреснен и последният ред от екрана. При тази ситуация

се нулира показателят за редовете и сигналът FLM се вдига във високо ниво – създава се нов фрейм - следващото опресняване ще започне от първи ред на екрана.

На фигура 3.2.1.3. е показана времедиаграмата за работа с екрана.



На фигурата ясно се вижда написаното в горните редове. Времедиаграмата представлява комбинацията от нивата на сигналите във времето при опресняването на един ред от екрана.

Отпечатване на символ на екрана - `void LCDWriteChar (U8 x, U8 y, U8 ch, U8 inv)`

С тази функция се отпечатва символ на екрана. Това става чрез попълването на байтовете на символа в масива, предствящ екрана.

Функцията не връща стойност, но приема четири аргумента:

- X – позиция по оста X, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- Y – позиция по оста Y, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- ch – символът, който да бъде отпечатан. Недопустими стойности за аргумента са букви на кирилица.
- inv – дали аргументът да е инвертиран или не. Аргумента може да приема една от следните стойности:

```
#define DISP_NORMAL        0
#define DISP_INVERTED 1
```

В началото на функцията прави проверка за валидността на аргументите. При невалидност на някой от аргументите, не се отпечатва нищо.

```
// adjust vertical position from char lines to pixels
y = y * FONT_CHAR_HEIGHT;
```

С този фрагмент код, аргументът Y се преобразува от позиция за отпечатване на символ от 0-20 във вертикалната позиция в редове на екрана.

```
// adjust character for displaying
ch = ch - FONT_START_CHAR;
```

Първите символи, които се използват за отпечатване са подредени по ASCII таблицата, както и заради това, че първите 32 символа от нея не са налични в таблицата, се налага предаденият символ като аргумент да бъде преобразуван по този начин, за да може да бъде намерен в масива със символи.

```
// display the selected character
for (c = 0; c < FONT_CHAR_HEIGHT; c++)
{
      // get next line from the character image
      ch_line  = pgm_read_byte(&Font8x8[ch][c]);

      // calculate the position in frame buffer
      fb_index = ((y+c)*LCD_WIDTH_BYTES) + x;

      // display it
      if (inv == DISP_INVERTED)
      {
            // inverted
            LCDFrameBuffer[fb_index] = ~ch_line;
      }
      else
      {
            // normal
            LCDFrameBuffer[fb_index] = ch_line;
                  }
```

Обхождането на масива се реализира с цикъл за всеки от редовете на символа, който символът се намира в масива със символи, изчислява се позицията за отпечатването му и на базата на аргументът за инвертиране се копира в масива, представящ екрана в нормален или инвертиран вид (Инвертиран е видът в който всички нули в байта са единици и обратно.).

Отпечатване на символен низ - `void LCDWriteStr (U8 x, U8 y, U8 * str, U8 inv)`

Функцията се свежда до множествено извикване на функцията за отпечатване на символ.

Функцията не връща стойност и приема следните четири аргумента:

- X – позиция по оста X, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- Y – позиция по оста Y, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- str – символният низ, който да бъде отпечатан. Недопустими стойности за аргумента са букви на кирилица.
- inv – дали аргументът да е инвертиран или не. Аргументът може да приема една от следните стойности:

```
#define DISP_NORMAL        0
#define DISP_INVERTED      1
```

Във функцията не се прави проверка за валидността на аргументите, но понеже функцията се свежда до множествено извикване на функцията за отпечатване на символ, предадените ѝ аргументи се проверяват и при невалидност не се извежда нищо.

Функцията последователно обхожда символите от низа, предаден като аргумент и ги предава на функцията за отпечатване на символ.

```
// display the current character
LCDWriteChar(x++, y, *str++, inv);
```

Освен смяната на символа за извеждане се сменят и координатите за извеждане на съответния символ – координатата X се увеличава.

Прави се проверка, дали всеки символ не е някой от следните, с цел извършване на специални операции:

- **`Carriage return`** – връща в началото на реда
- **`Line feed`** – преминава на следващия ред
- **`Clear screen`** – изчиства се съдържанието на целия масив, представящ екрана

Отпечатване на символ, разположен в паметта за код - `void LCDWriteCharP (U8 x, U8 y,  FLASH U8 * ch_ptrP, U8 inv)`

Функцията отпечатва единичен символ на екрана, като изпълнението се свежда до извикването на функцията за отпечатване на символ **`LCDWriteChar`**.

Функцията не връща стойност, но приема четири аргумента:

- X – позиция по оста X, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- Y – позиция по оста Y, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- ch_ptrP – указател към символа, разположен в паметта, предназначена за програмент код, който да бъде отпечатан. Недопустими стойности за аргумента са букви на кирилица.

- inv – дали аргументът да е инвертиран или не. Аргумента може да приема една от следните стойности:

```
#define DISP_NORMAL        0
#define DISP_INVERTED      1
```

```
LCDWriteChar(x,y,pgm_read_byte(ch_ptrP),inv);
```

Функцията `pgm_read_byte(ch_ptrP)` приема като аргумент указател към символ в паметта за код и връща самия символ.

Отпечатване на символен низ, разположен в паметта за код - `void LCDWriteStrP (U8 x, U8 y, FLASH U8 * strP, U8 inv)`

Функцията извежда символен низ, разположен в паметта, предназначена за програмния код на управляващия софутер. Свежда се до множествено извикване на функцията за отпечатване на символ.

Функцията не връща стойност, но приема четири аргумента:

- X – позиция по оста X, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- Y – позиция по оста Y, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- strP – указател към символният низ, който да бъде отпечатан. Недопустими стойности за аргумента са букви на кирилица.

- inv – дали аргументът да е инвертиран или не. Аргумента може да приема една от следните стойности:

```
#define DISP_NORMAL        0
#define DISP_INVERTED      1
```

Във функцията не се прави проверка за валидността на аргументите, но понеже функцията се свежда до множествено извикване на функцията за отпечатване на символ, предадените ѝ аргументи се проверяват и при невалидност не се извежда нищо.

Символът се прочита чрез функцията `pgm_read_byte()`.

```
// read next char
     ch = pgm_read_byte(strP);
```

Започва се цикъл за обхождането на символите от масива. При завършване на стринга се излиза от този цикъл.

```
// exit the loop at the end of the string
          if (ch == 0)
          {
               break;
               }
```

Изчистване на екрана - `void LCDClearScreen(void)`

Функцията изчиства целия екран – инициализира го с нули. Това става чрез извикване на memset().

```
// clear screen
memset (LCDFrameBuffer, 0, sizeof(LCDFrameBuffer));
```

Отпечатване на цифра на екрана – `void LCDWriteDigit_ (U8 x, U8 y, U8 digit_type, U8 digit)`

Функцията извежда цифра на екрана, която може да бъде от големия или от малкия комплект и отпечатваната цифра може да бъде едноцифрана.

Функцията не връща стойност, но приема следните аргументи:

- X – позиция по оста X, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- Y – позиция по оста Y, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- `digit_type` – тип на извежданата цифра. Може да бъде една от стойностите:

```
//digit attribute
#define BIG_DIGIT          0
#define SMALL_DIGIT        1
```

- `digit` – цифрата, която да бъде отпечатана

Функцията не проверява коректността на аргументите, поради факта, че не се извиква директно, а се извиква от друга функция, която подава коректни данни.

```
void LCDWriteDigit_     (U8 x, U8 y, U8 digit_type, U8 digit)
{

      U16 i, pos;

      //determine start position
      pos = y*LINE_NUM_CHARS + x;

      //loop for all digit bytes
      for(i=0;i<pgm_read_byte(&SizeWidthValues[digit_type][SIZE]);i++){

      //print digit
      if(digit_type == BIG_DIGIT)
            LCDFrameBuffer[pos] = pgm_read_byte(&BigDigit[digit][i]);
      else
            LCDFrameBuffer[pos] = pgm_read_byte(&SmallDigit[digit][i]);

      //new row
      if(((i+1)%pgm_read_byte(&SizeWidthValues[digit_type][WIDTH]))==0)
            pos=pos+LINE_NUM_CHARS -
            pgm_read_byte(&SizeWidthValues[digit_type][WIDTH])+1;
      else
            pos++;
```

```
            }
}
```

На първо място във функцията се пресмята позицията, на която ще се отпечатва. Цикъл обхожда всички байтове на цифрата, като размерът му е определен спрямо типа на цифрата. Спрямо типът на символа се избира и кой масив да се ползва, за да се прочетат байтовете на съответната цифра. Като първо направление в масива за голям/малък комплект цифри се предава самото число, а като втори – номера на байта от символа, който се иска да се прочете. Отново спрямо типът на цифрата се определя колко байта е ширината ѝ – за да се определи кога да се премина на следващия ред за отпечатване на цифрата.

Отпечатване на многоцифрени числа - `void LCDWriteDigit (U8 x, U8 y, U8 digit_type, U16 digit, U8 pos_count)`

Функцията се свежда до последователно извикване на функцията за отпечатване на цифри. Не се връща резултат, но се приемат следните параметри:

- X – позиция по оста X, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- Y – позиция по оста Y, където да бъде отпечатан символът. Допустимите стойности на аргумента са от 0 до 20
- `digit_type` – тип на извежданата цифра. Може да бъде една от стойностите:

```
//digit attribute
#define BIG_DIGIT          0
#define SMALL_DIGIT        1
```

- `digit` – число, което да бъде отпечатано. От интервала 0-65535.
- `pos_count` – броя на позициите, в които да се изведе числото – реализиране на дясно подравняване

```
void LCDWriteDigit (U8 x, U8 y, U8 digit_type, U16 digit, U8 pos_count)
{

	//get init positin
	U8 init_pos_count = pos_count;

	do{
		if((digit == 0) && (pos_count!=init_pos_count))
		{
			 //clear unused positions
			LCDWriteIcon(x+(pos_count -1) *
			pgm_read_byte(&SizeWidthValues[digit_type][WIDTH]), y,
			digit_type);
		}
		else
		{
			//write the digit
			LCDWriteDigit_( x +(pos_count-1) *
			pgm_read_byte(&SizeWidthValues[digit_type][WIDTH]), y,
			digit_type, digit%10);
			digit = digit/10;
		}

		pos_count--;

	}
	while(pos_count!=0);

}
```

Във функцията се отпечатва числото отдясно наляво. Това става чрез деление на десет и намереният остатък е най-дясното число. След това числото се разделя на десет и цялата част се превръща в числото, които на следващата итерация ще бъде делено на десет и търсен остатъкът. След отпечатване на цялото число се изчистват позициите, незаети от числа, за да се избегнат нежелани ситуации.

Отпечатване на иконки - `void LCDWriteIcon(U8 x, U8 y, U8 icon)`

Функцията отпечатва на екрана иконки, записани в паметта, предназначена за кода на управляващия софтуер. Функцията не връща резултат, но приема следните параметри:

- X – позиция по оста X, където да бъде отпечатана иконката. Допустимите стойности на аргумента са от 0 до 20
- Y – позиция по оста Y, където да бъде отпечатана иконката. Допустимите стойности на аргумента са от 0 до 20
- `icon` – идентификатор на иконката. Може да бъде една от стойностите:
  - Индикация за икономичен режим на съветника за предавките

```
#define ECONOMIC_MODE2
```

  - Индикация за динамичен режим за съветника за предавките

```
#define SPORT_MODE        3
```

  - Индикация за бърза половника на предавка

```
#define FAST_GEAR         4
```

  - Индикация за бавна половинка на предавка

```
#define SLOW_GEAR         5
```

  - Индикация за съвет „Превключи нагоре”

```
#define GEAR_UP      6
```

  - Индикация за съвет „Превключи надолу”

```
#define GEAR_DOWN         7
```

  - Индикация за неутрална предавка

```
#define N_GEAR            8
```

- Индикация за задна скорост

```
#define R_GEAR              9
```

- Горивна колонка

```
#define FUEL_TANK           10
```

- Линия

```
#define LINE                11
```

- Бутон OK

```
#define BUTTON_OK           12
```

- Бутони – стрелки

```
#define ARROWS_PAD          13
```

- Минус

```
#define MINUS               14
```

- Градус Целзий

```
#define CELSIUS       15
```

- Празни символи за изчистване на област

```
#define CLR_TEMP            16
#define CLR_MINUS           17
#define CLR_GEAR_ADV        18
#define CLR_HALF_GEAR       19
```

Цялата функция представлява един оператор **switch**, който според подадения аргумент за идентификатор на иконка избира от кой масив да бъдат прочетени данните.

```
void LCDWriteIcon (U8 x, U8 y, U8 icon){

      U16 i, pos;

      pos = y*LINE_NUM_CHARS + x;

            for(i=0;i<pgm_read_word(&SizeWidthValues[icon][SIZE]);i++){

            //Determining which array in FLASH to be used
            switch(icon)
            {
                  case CLR_BIG_DIGIT:     LCDFrameBuffer[pos] =
                        pgm_read_byte(&array14[i]);
                        break;
                  case CLR_SMALL_DIGIT:LCDFrameBuffer[pos] =
                        pgm_read_byte(&array15[i]);
                        break;
```

```
            case ECONOMIC_MODE: LCDFrameBuffer[pos] =
                  pgm_read_byte(&array2[i]);
                 break;
            case SPORT_MODE:  LCDFrameBuffer[pos] =
                 pgm_read_byte(&array3[i]);
                 break;
            case FAST_GEAR:         LCDFrameBuffer[pos] =
                 pgm_read_byte(&array4[i]);
                 break;
            case SLOW_GEAR:         LCDFrameBuffer[pos] =
                 pgm_read_byte(&array5[i]);
                 break;
            case GEAR_UP:           LCDFrameBuffer[pos] =
                 pgm_read_byte(&array6[i]);
                 break;
            case GEAR_DOWN:         LCDFrameBuffer[pos] =
                 pgm_read_byte(&array7[i]);
                 break;
            case N_GEAR:            LCDFrameBuffer[pos] =
                 pgm_read_byte(&array8[i]);
                 break;
            case R_GEAR:            LCDFrameBuffer[pos] =
                 pgm_read_byte(&array9[i]);
                 break;
            case FUEL_TANK:         LCDFrameBuffer[pos] =
                 pgm_read_byte(&array10[i]);
                 break;
            case LINE:              LCDFrameBuffer[pos] =
                 pgm_read_byte(&array11[i]);
                                          break;
            case BUTTON_OK:         LCDFrameBuffer[pos] =
                 pgm_read_byte(&array12[i]);
                 break;
            case ARROWS_PAD:  LCDFrameBuffer[pos] =
                 pgm_read_byte(&array13[i]);
                 break;
            case MINUS:             LCDFrameBuffer[pos] =
                 pgm_read_byte(&array16[i]);
                 break;
            case CELSIUS:           LCDFrameBuffer[pos] =
                 pgm_read_byte(&array17[i]);
                 break;
            case CLR_TEMP:          LCDFrameBuffer[pos] =
                 pgm_read_byte(&array18[i]);
                 break;
            case CLR_MINUS:         LCDFrameBuffer[pos] =
                 pgm_read_byte(&array19[i]);
                 break;
            case CLR_GEAR_ADV:            LCDFrameBuffer[pos] =
                 pgm_read_byte(&array20[i]);
                 break;
            case CLR_HALF_GEAR:     LCDFrameBuffer[pos] =
                 pgm_read_byte(&array21[i]);
                 break;
      }

                 if(((i+1)%pgm_read_byte(&SizeWidthValues[icon]
[WIDTH]))==0)
            pos=pos+LINE_NUM_CHARS-pgm_read_byte(&SizeWidthValues[ico
            n][WIDTH])+1;
      else
```

```
                pos++;
        }
}
```

Цикълът се изпълнява, докато се обходят всички байтове от иконката, която ще се изведе. За всеки един от възможните входове чрез аргумента за идентификатор на иконката има отделен клон в оператора **switch**. В него се определя от кой масив, разположен в паметта, предназначена в код на програмата, да се прочетат байтовете и да се копират в масива, представящ екрана.

3.2.2. Модул на панела, реагиращ на допир

Панелът, реагиращ на допир е единственият начин за реализиране на обратна връзка между потребителя и хардуерната платформа.

Функционалност:

- Реализиране на функции, определящи координатите на точката, в която е упражнен натиск

Използван хардуер:

- Панел, реагиращ на допир на LCD екрана PC0919WE07 от Palm III
- PORTF0 и PORTF2 на процесора

Файлове с управляващия софтуер

- touchscreen_driver.h
- touchscreen_driver.c

Описание на модела на модула(фиг. 3.2.2.1.)

На фигура 3.2.2.1. е показан модела на модула на панела, реагиращ на допир. На него ясно се вижда структурата на модула – разделението на хардуер и софтуер, като основни клонове от реализацията на модула.

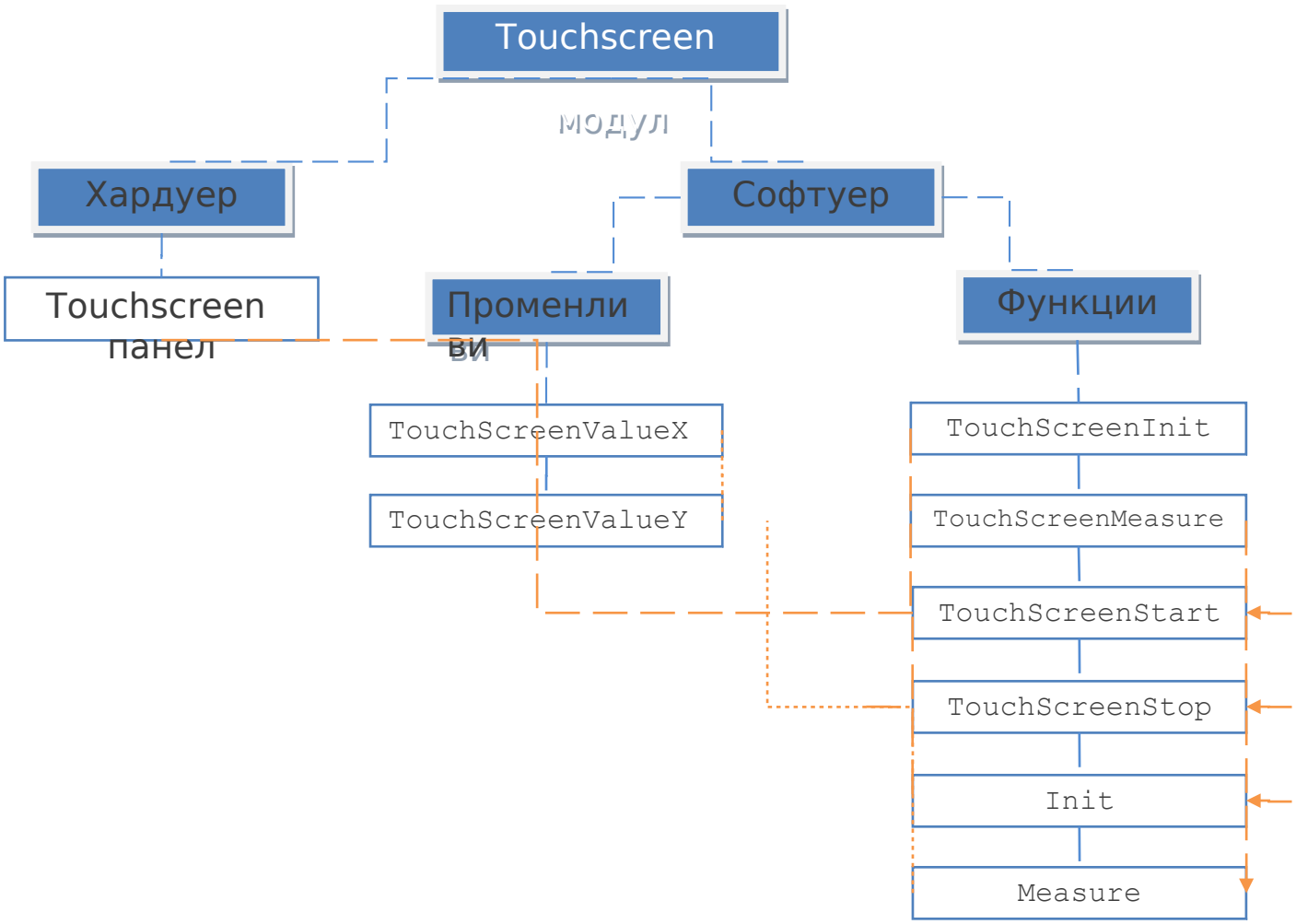
Хардуер

Хардуерът, използва за реализирането на модула е **панел, реагиращ на допир**. Той е разположен върху LCD екрана, но е по-голям от него и има област, в която touchscreen панелът не е върху екрана. Това може да се

види ясно на фигура 2.4.1.. Панелът е с дължина на диагонала 4”.

Всъщност панелът може да се разглежда като мост от резистори, които променят съпротивлението си при всяко натискане на панела.

Фигура 3.2.2.1.: Модел на модула на панела, реагиращ на допир



Според това къде е натиснат панела, съпротивлението се променя по различен начин, съответно и пада на напрежение е различен. Измерва се промяната на напрежението по две направления – това са двете

координатни оси – абсциса и ординатa, за да се установят и координатите на точката, в която е приложен натискът. Поради факта, че става дума за пад на напрежение е необходимо използването на вградения модул ADC или Аналогово-цифров преобразовател - описана е в т.2.5. на настоящата дипломна работа. Използват се два канала на блока на ADC на процесора. Това са каналите 0 и 2, свързани съответно към пиновете 0 и 2 на PORTF на процесора.

Софутер

На модела на Touchscreen модула е дадено разделението на софутерната част от реализацията на модула:

- Променливи – това променливите `TouchScreenValueX` и `TouchScreenValueY`, които представят координатите на точката, в която е натиснат панела;
- Функции – функциите реализират функционалностите на модула, като намиране на координатите на точката, в която е натиснат панела. Те реализират връзката между отделните софтуерни клонове, както и връзката хардуер-софтуер в модула.

Променливи

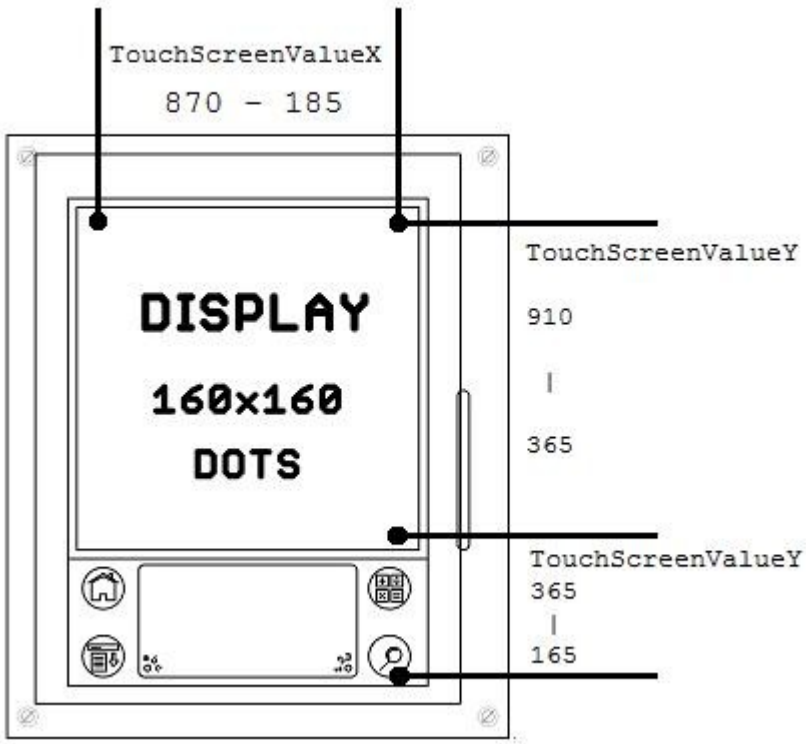
```
extern U16 TouchScreenValueY;
extern U16 TouchScreenValueX;
```

Променливите съдържат координатите на точката, в която е натиснат. Те съдържат резултата от

преобразуването на аналоговата величина в цифрова. Единствено функцията `Measure()` има достъп до тези применливи. Тя има задачата да опреснява стойностите им на базата на реалното измерване, извършено от ADC модула на процесора. Използва се филтиране на резултата от преобразуването с цел избягване на смущения в измерването. Филтрирането се извършва чраз усредняване на стойността от няколо измервания. Броят на измерваният, от които се усреднява стойността формират прецизността на измерването и на стойностите на променливите.

На фигура 3.2.2.2. са показани стойностите на `TouchScreenValueX` и `TouchScreenValueY` в различни места на панела, реагиращ на допир.

Фигура 3.2.2.2.: Стойности на `TouchScreenValueX` и `TouchScreenValueY` в различни места на панела, реагиращ на допир



Стойността на `TouchScreenValueX` се изменя в интервала 870 – 185.

Стойността на `TouchScreenValueY` се изменя в интервала 910 – 165. Стойностите в интервала 910 – 365 отговарят на позиция на натиск върху екрана, докато стойностите в интервала 365 – 165, съответстват на точка разположена извън екрана – в областта, където има само панел, реагиращ на допир без екран.

Функции

Функциите на модула за Touchscreen реализират функционалността за работа с панела, реагиращ на допир. Те отговарят за опресняването на стойностите на променливите `TouchScreenValueX` и `TouchScreenValueY`.

Следва списък на функциите, декларирани във файла touchscreen_driver.h и имплементирани във файла touchscreen_driver.c.

```
void TouchScreenInit(void);
void Init(U8 axis);
void TouchScreenStart(void);
void TouchScreenStop(void);
void TouchScreenMeasure(void);
void Measure(U8 axis);
```

Инициализиране на панел, реагиащ на допир – `void TouchScreenInit(void);`

Функцията има ролята да извърши инициализацията на ADC модула на процесора за измерването на стойностите на Touchscreen-а. В тази функци се извършват следните операции:

- Избиране на опорно напрежение;
- Избиране на делител на честота (50kHz – 200kHz са нужни за работата на модула);
- Изчистването на битовете, които избират канала за измерване.
- Разрешаване на опресняването на стойностите за координатите
- **НЕ** се включва ADC модулът

```
void TouchScreenInit(void)
{

	//Enabling AREF as reference voltage
	cbi(ADMUX,REFS1);
	cbi(ADMUX,REFS0);

	//Setting Right adjustment for the result of ADC
	conversion
	cbi(ADMUX,ADLAR);//cleaer left adjust


	//Clearing the channel selection bits
	ADMUX  &= ~(0x1F<<MUX0);

	ADCSRA &= ~(0x07<<ADPS0);		//clear prescaler
	ADCSRA |= (ADC_PRESC<<ADPS0);	//select prescaler

	cbi(ADCSRA,ADATE);

	TouchScreenInitialized=1;
}
```

За работа с ADC модулът трябва да се прави справка с спецификацията на процесора, избран за настощата дипломна работа, как се реализира работата на този модул.

Инициализиране на модула за измерване на координатите по определан ос – `void Init(U8 axis);`

Функцията има за цел да инициализира модула за ADC за измерване на показанието на конкретна ос – X или Y. Функцията приема аргумент, който определя, коя от осите да бъде измерена. Не се връща резултат. В тази функция се извършват дейностите:

- Настройват се пиновете за измерването, както и това дали да са входове или изходи;
- Определяне на канала за измерване – за оста X това е канал 2, а за оста Y – канал 0;
- Разрешаване на функцията за опресняване на стойнсотите. Това се извършва със следния фрагмент код:

```
//Enabling the Measure() function
TouchScreenInitialized=1;
```

Стартиране на модула ADC на процесора - `void TouchScreenStart(void)`

Функцията не приема аргументи и не връща стойност. В нея се включва модула ADC на процесора със следния ред код:

```
//enable ADC measurement
 sbi(ADCSRA, ADEN);
```

Във функциите за инициализация се разрешава измерването на координатите на панела, но включването на модула се извършва в тази функция, т.е. без извикването ѝ не се извършва измерване.

Спиране на модула ADC на процесора - `void TouchScreenStop(void)`

Функцията не приема аргументи и не връща стойност. В нея се спира модула ADC на процесора със следния ред код:

```
//Disabling the Measure() function
TouchScreenInitialized=0;
 cbi(ADCSRA, ADEN);
```

Освен това се забранява и измерването на координатите на екрана като се записва нула в променливата `TouchScreenInitialized`.

Обща функция за измерване на координатите на панела - `void TouchScreenMeasure(void)`

Функцията не приема аргументи и не връща стойност. Операциите, които се извършват в нея целят да се измерят координатите на точката, в която е натиснат панелът. Тази функция се извиква във всяка задача, използваща модула на панела, реагиращ на допир. Изпълнението ѝ се свежда до извикване на други функции:

```
void TouchScreenMeasure(void)
{
	if(TouchScreenInitialized)
	{
		TouchScreenStop();
		Init('X');
		TouchScreenStart();
		Measure('X');

		TouchScreenStop();
		Init('Y');
		TouchScreenStart();
		Measure('Y');
	}
}
```

Както се вижда всички операции във функцията се извършат само ако е разрешено опресняването на

координатите на панела. Във функцията последоватлно се извършват следните операции:

- Спиране на модула на панела;
- Инициализирането му за измерване на координантите по определена ос;
- Стартиране на панела;
- Измерване на координатите по измерваната ос.

Измерване на координатите по определена ос -

```
void Measure(U8 axis);
```

Реалното измерване на показанието на ADC модула се извършва от тази функция. Тя опреснява координатите по определената ос в зависимост от подадения параметър, който указва коя ос да бъде измерена.

```
void Measure(U8 axis){

	U8 i;
	U16 sum=0;

	sum=0;

	//Real measurement
	for(i=0;i<PRECISION;i++){
		sbi(ADCSRA, ADSC);
		cbi(ADCSRA, ADATE);

		while(!(ADCSRA & (1<<ADIF)));
		sbi(ADCSRA, ADIF);
		sum+=ADC;
	}

	if(axis=='X'){//measuring TouchScreenValueX

		TouchScreenValueX=sum/PRECISION;

	}else{
		//measuring TouchScreenValueY
		TouchScreenValueY=sum/PRECISION;
	}
}
```

При измерването се изпълнява цикъл като на всяка една от итерациите измерва показанията на модула. Броят на итерациите зависи от следната дефиниция:

```
#define PRECISION 5
```

От нея се определя колко пъти да се извърши измерване и осредняване на стойността от изерването.

```
//Real measurement
      for(i=0;i<PRECISION;i++){
            sbi(ADCSRA, ADSC);
            cbi(ADCSRA, ADATE);

            while(!(ADCSRA & (1<<ADIF)));
            sbi(ADCSRA, ADIF);
            sum+=ADC;
      }
```

При реалното измерване се извършват следните операции:

- Стартиране на аналогово-цифрово преобразуване;
- Да бъде извършено само едно преобразуване;
- Изчакване преобразуването да завърши;
- Прибавяне на преобразуваната стойност към предишните стойности от началото на текущото извикване на функцията.

След това се извършва присвояването на измерената стойността за конкретната ос на съответната променлива

```
if(axis=='X')
{
      //measuring TouchScreenValueX
      TouchScreenValueX=sum/PRECISION;
}
else
{
      //measuring TouchScreenValueY
```

```
        TouchScreenValueY=sum/PRECISION;
}
```

3.2.3. Модул на часовника и календара

Модула на часовника и календара добавя допълнителна функционалност към бордовия компютър, различна от тази да се показват работните параметри на машината.

Функционалност:

- Реализиране на часовник
- Реализиране на календар

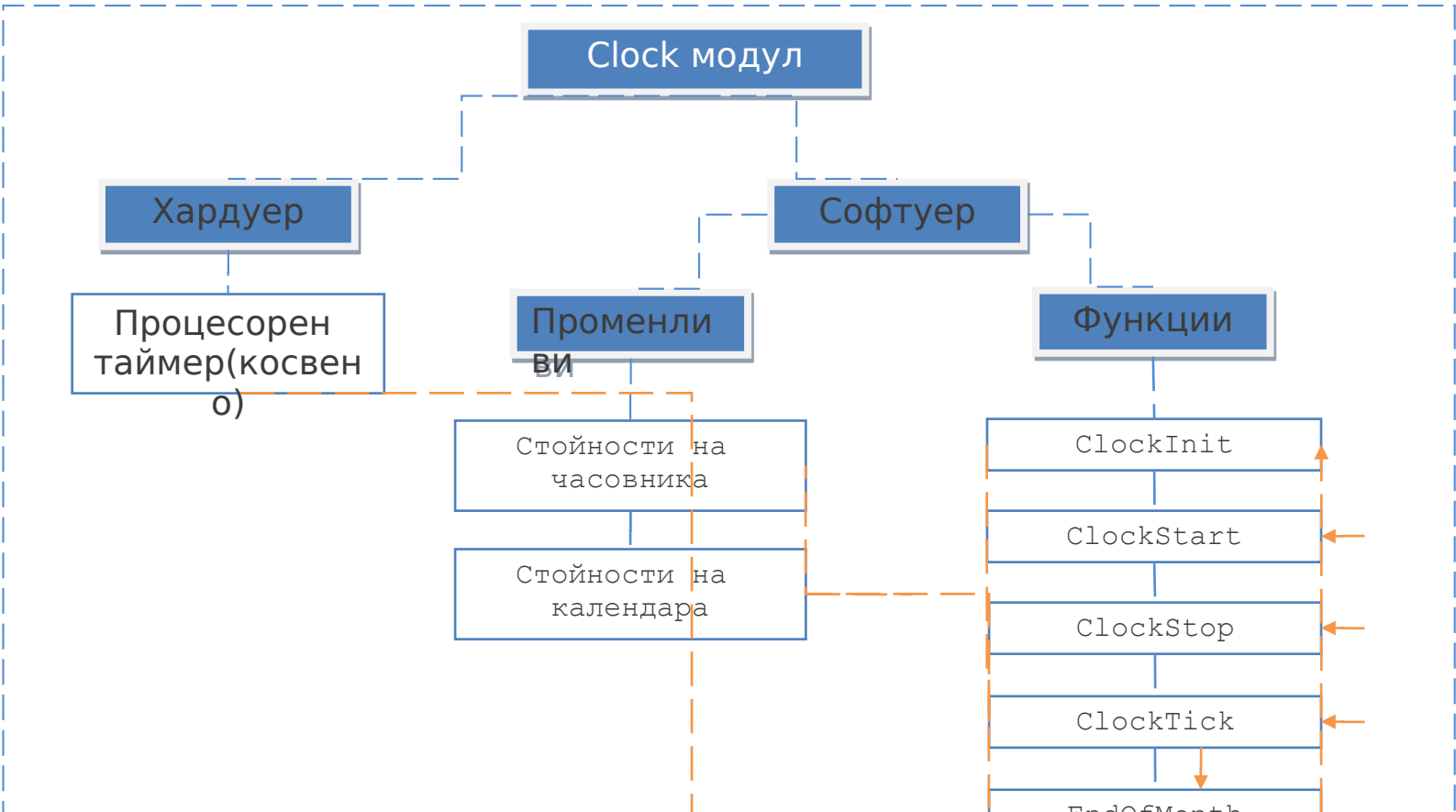
Използван хардуер:

- Таймер на процесора

Файлове с управляващия софтуер

- clock.h
- clock.c

Фигура 3.2.3.1. Модел на модула за календара и



часовника

Описание на модула за часовника и календара

Хардуер

За реализацията на този модул косвено е използван таймер на процесора. Косвено ще рече, че не е използван способ за директно конфигуриране на таймера, с цел да се отмери времето за часовника и календара (вж. т.2.5.). Начина за реализиране на измерването на период от време е реализиран чрез функцията, предоставена от

операционната система `AvrXDelay` (вж. т.2.2.). Тази функция от своя страна работи на базата на `AvrXTimerHandler()`, която се извиква в прекъсването на таймер на процесора.

Софтуер

На модела на Clock модула е дадено разделението на софутерната част от реализацията на модула:

- Променливи – това променливите съдържащи данните за часовника и календара
- Функции – реализират предназначението на модула, като показване на текущия час, и данните на календара.

Променливи

```
//*********************************************************
// Time and Calendar variables
//*********************************************************
extern U8 hours;
extern U8 minutes;
extern U8 seconds;
extern U8 date;
extern U8 month;
extern U8 year;
extern U8 day;
```

Това са променливите съдържащи информацията за часовника и календара.

- Часове – показанието на часовника за часове във формат с 24 часа - **HH**

- Минути - показанието на часовника за минутите - **MM**
- Секунди - показанието на часовника за секундите
- Дата – показанието на календара за текущата дата - **DD**
- Месец - показанието на календара за текущия месец - **MMM**
- Година – показание на календара за текущата година - **YY**
- Ден от седмицата - показанието на календара за текущия ден от седмицата - **DDD**

Извеждането на екрана става в следния формат:

DDD MMM DD YY HH:MM

Функции

Функциите в модула за часовника и календара отговарят за опресняването на стойностите на променливите, споменати в горната точка, както и за управлението на модула като цяло. Функциите са декларирани в clock.h и са имплементирани в clock.c. Ето списък на всички функции, дефинирани в модула:

```
//*************************************************************
// Real-Time Clock functions declarations
//*************************************************************
void ClockInit        (void);
void ClockStart       (void);
void ClockStop        (void);
void ClockTick        (void);
U8   EndOfMonth       (U8 monthp, U8 yearp);
```

Инициализиране на модула - `void ClockInit (void);`

Функцията е отговорна за привеждането на модула в работоспособно състояние, което включва и инициализирането на променливите, съдържащи стойностите на часовника и календара.

Стартиране на модула - `void ClockStart (void);`

В тази функция се разрешава опресняването на стойностите на променливите на календара и часовника. Това става с този фрагмент код:

```
//Start Clock
ClockStarted = 1;
```

Спиране на модула - `void ClockStop (void);`

Функцията спира функцията, която опреснява стойностите на променливите на календара и часовника. Това става с този фрагмент код:

```
//Stop Clock
ClockStarted = 0;
```

Опресняване на стойностите на календара и часовника - `void ClockTick (void);`

Функцията извършва същинската работа на часовника и календара. Тази функция бива извиквана на интервал от една секунда и така се реализира отмерването на изминалото време. При започване изпълнението на функцията се проверява дали е включен модулът. Ако не е не се предприема действие – модулът не може да работи без да е включен. Ако е включен започва работата на функцията. Основата на функцията е увеличаване на секундите, след което следват множество проверки дали е

необходимо увеличаване на някой от следващите променливи: минути, часове, дни, месеци, години.

Във функцията се правят проверки за специфични ситуации. Тези ситуации ограничават работата на модула, за да се показват коректни данни. Пример за такива специфични ситуации са:

- Секундите и минутите не могат да имат стойност по-голяма от 59. При достигане на тази стойност параметрите се нулират и се увеличава следващият параметър – съответно минути или часове;
- Часовете не могат да имат стойност, по-голяма от 23. При достигане на тази стойност параметърът се нулират и се увеличава следващият параметър – дата и ден от седмицата;
- Датата не може да е със стойност, по-голяма от броя на дните в текущия месец. За спазване на тази ситуация се използва функцията `EndOfMonth`, която определя колко е броя на дните в месеца – 30/31 или 28/29 за месец февруари. При достигане на тази стойност параметърът се установява в единица и се увеличава следващият параметър – месец.
- Денят от седмицата не може да е по-голям от седми ден – Неделя. – При достигане на тази стойност параметърът се установява в единица.

- Месецът не може да е по-голям от дванадесети месец - Декември. При достигане на тази стойност се увеличава следващият – годината.

Определяне на броя на дните в месеца - `U8 EndOfMonth (U8 monthp, U8 yearp);`

Тази функция се използва за определяне на броя на дните във всеки месец.

Функцията приема два параметъра:

- `monthp` - месец, за който се иска да се върне броя на дните му
- `yearp` – текуща година, за да се направи проверка, дали е високосна, което е необходимо за определянето на броя на дните на месец февруари. Проверката се извършва със следния фрагмент код:

```
if ((yearp & 0x03) == 0)
          return 29;
else
          return 28;
```

Ако двата най-младши бита от байта на годината са нули, това означава, че годината, върху която се извършва проверката е високосна. Тогава върнатата стойност за броя на дните на месец февруари е 29. В противен случай, т.е. някой от двата най-младши бита е единица, то годината не е високосна и върнатата стойност е 28.

Функцията връща броя на дните в месеца от годината, предадени като параметри.

3.2.4. Модул за температурата

Модула за температурата дава информация за външната температура. Това се извършва чрез цифров датчик, поддържащ TWI (вж. т. 2.5.).

Функционалност:

- Реализиране на измерване на температурата

Използван хардуер:

- Датчик **TCN75;**(вж. Използвана литература 4.)
- PORTD0 и PORTD1

Файлове с управляващия софтуер

- TWI.h
- TWI.c

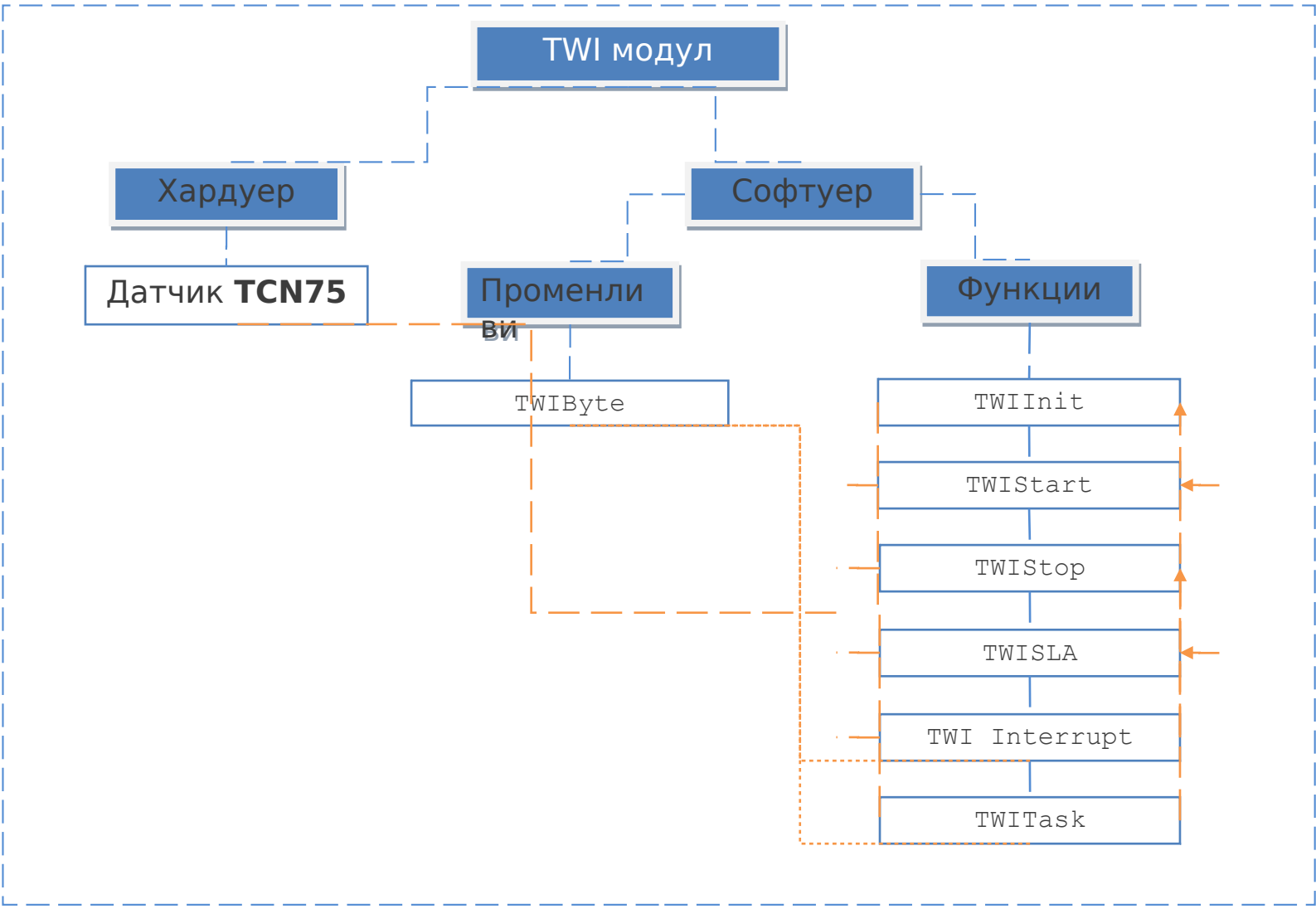
Описание на модула на температурата

Хардуер

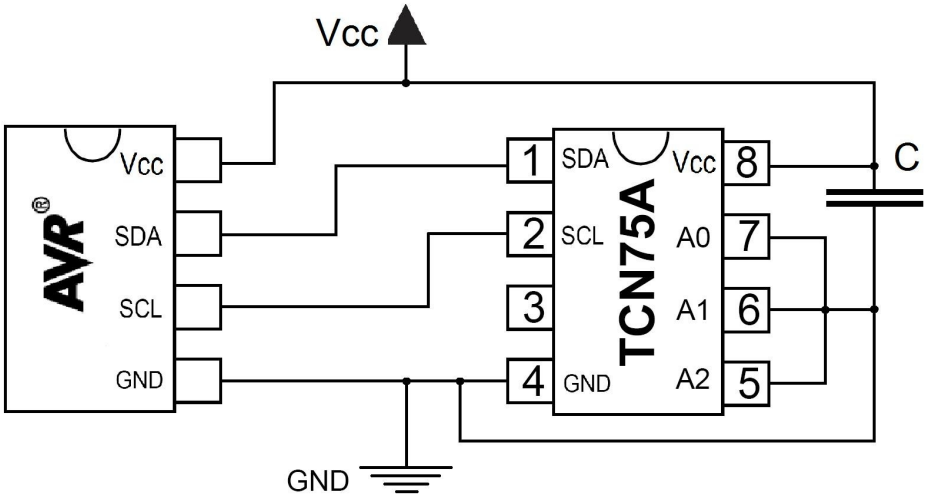
Използваният хардуер за реализиране работата на този модул е датчикът **TCN75**. Това е цифров датчик, който предава измерената температура на процесора чрез TWI (вж. т.2.5.). Използваната схема на свързване е показана на фигура 3.2.4.2. На нея се вижда, че двата сигнала, необходими за комуникацията чрез TWI са свързани с процесора, както и двата захранващи проводника – датчикът бива захранван с напрежение от захранващия блок на хардуерната развойна среда. Пинове A0, A1 и A2

имат функцията за определят адреса на устройството за организиране на комуникацията с процесора.

Фигура 3.2.4.1. Модел на модула за календара и часовника



Фигура 3.2.4.2.: Схема на свързване на датчика **TCN75** с



процесора **AVR**

В спецификацията на датчика е записан начина на определяне на седембитовия адрес. Старшите четири бита са заводски определени да бъдат **1001**. И с останалите три пина за адрес се запълват низшите три позиции на адреса. Чрез свързването на пина към захранване (Vcc), съответния бит в адреса се установява в единица. При свързване към земя (GND), бита в адреса се установява да е нула. В конкретната реализация и трите пина са свързани към земя, което означава, че низшите три бита от адреса са **000**, а целият адрес е **1001000**.

Кондензаторът **C** е със стойност **100nF** и има за цел да филтрира смущенията от захранването. Според информацията от спецификацията на датчика е добре, той да бъде монтиран възможно най-близо до захранващите изводи на датчика.

Софтуер

На модела на TWI модула е дадено разделението на софутерната част от реализацията на модула:

- Променливи – това са променливите съдържащи данните, получени от датчика, относно измерената температура
- Функции –реализират предназначението на модула, като показване на измерената температура от датчика. Те реализират връзката между отделните софтуерни клонове, както и връзката хардуер-софтуер в модула.

Променливи

```
extern U8 TWIByte;
```

Това е променливата, съдържаща измерената температура, изпратена от датчика чрез TWI. Показанието ще бъде изпратено в допълнителен код, ако измерената температура е отрицателна. Границите, които се поддържат от датчика като температурен обхват са -40 до +125 градуса Целзий. Точността е 2 градуса Целзий. Полученото показание е в цяло число.

Функции

Функциите в модула за температурата отговарят за актуализирането на стойността на показваната температура, спомената в горната точка, както и за управлението на модула като цяло. Функциите са

декларирани в TWI.h и са имплементирани в TWI.c. Ето списък на всички функции, дефинирани в модула:

```
void TWIInit(void);
void TWIStart(void);
void TWIStop(void);
void TWISLA(U8 sla_byte);
```

Инициализиране на модула - `void TWIInit(void);`

Функцията отговаря за инициализирането на TWI модула.

```
void TWIInit(void){

      //Prescaler is 1 - by default

      //Setting Bit Rate - 18
      TWBR = 18;

      //Enabling TWI
      sbi(TWCR, TWEN);

      //Enbaling TWI interrupt
      sbi(TWCR, TWIE);


}
```

Показания фрагмент код представя тялото на разглежданата функция. В нея се извършват следните операции:

- Задаване на Bit Rate на TWI шината
- Prescaler е зададен по подразбиране
- Включване на TWI модула на процесора – поема се контрол върху сигналите SDA и SCL
- Разрешаване на прекъсването на TWI модула

Поради факта, че в тази функция се включва модула, без извикването ѝ, няма да е възможно извършването на комуникацията с датчика и показването на температурата.

Стартиране на модула - `void TWIStart(void);`
В тази функция се извършва стартирането на модула. Това става със следния фрагмент код:

```
TWIStopped = 0;

//Send START condition
sbi(TWCR, TWSTA);
sbi(TWCR, TWINT);
```

Показва се, че модула е стартиран и се изпраща т.н. условие START. Това е първата стъпка във всяка комуникация, използваща TWI. След изпращането на START, се генерира перкъсване, за изпратеното START условие.

Спиране на модула - `void TWIStop(void);`
Функцията е отговорна за спирането на модула. В нея се задава, че модулът е спрян и се изпраща т.н. STOP условие. Това се извършва със следния фрагмент код:

```
TWIStopped = 1;

//Send STOP condition
sbi(TWCR, TWSTO);
cbi(TWCR, TWSTA);
sbi(TWCR, TWINT);
```

При изпращането на условие STOP се пекратява комуникацията по шината. За да се продължи отново трябва да се изпрати отново условие START.

Адресиране с модула - `void TWISLA(U8 sla_byte);`

Функцията отговаря за адресирането на други устройства чрез TWI шината. Това става чрез приемания параметър от функцията. Този параметър се поставя в регистъра за данни на TWI модула и бива изпратен по шината. В конкретния случай адресът може да бъде една от следните дефиниции:

```
//************************************************************
// Definitions for TWI transmiting
//************************************************************
#define SLAW 0x90
#define SLAR 0x91
```

С тези дефиниции са означени адресът на термодатчика, комбинирани с READ/WRITE бит, които заедно образуват байта, който да се изпрати. От изпращането на този адрес зависи по-нататъшното поведение на модула (вж. т.2.5.).

Прекъсване на TWI модула - `TWI Interrupt`

В прекъсването на TWI модула се проверява статуса на шината и се извършват действията, необходими за продължаване на комуникацията. Фрагмента код показва операциите, извършване в прекъсването на модула:

```
switch(TW_STATUS){
        case TW_START:
                TWISLA(SLAR);
                break;

        case TW_MR_SLA_ACK:
                cbi(TWCR, TWSTA);
                sbi(TWCR, TWINT);
                break;

        case TW_MR_DATA_NACK:
                TWIByte = TWDR;
                TWIStop();
```

```
                    break;
};
```

В `switch-case` оператор се разглеждат отделните статуси на TWI модула при всяко изплънение на функцията, обслужваща прекъсването.

След изпращането на START условие, се генерира прекъсване и изпълнението се насочва към тази част от кода:

```
case TW_START:
     TWISLA(SLAR);
     break;
```

В нея се изпраща адреса на датчика, с който ще се извършва комуникацията.

След изпращането на адреса, при нормално стечение на нещата, датчикът връща потвърждение по шината и изпълнението идва тук:

```
case TW_MR_SLA_ACK:
     cbi(TWCR, TWSTA);
     sbi(TWCR, TWINT);
     break;
```

Според описанието на датчика и според желаното поведение на модула, след изпълнението на този код ще доведе до получаването на байт информация, след който ще бъде върнато **НЕ**потвърждение, което ще доведе до прекратяване на получаването на информация.

След получаването на байта информация, се изпълнява този код:

```
case TW_MR_DATA_NACK:
     TWIByte = TWDR;
     TWIStop();
     break;
```

Чрез него се полученият байт се копира в променливата `TWIByte` и се изпраща условие СТОП по шината.

След повторното стартиране на модула(изпращане на START) тези стъпки от комуникацията се повтарят.

3.2.5. Модул за количеството гориво в резервоара

Количеството на горивото в резервоара не много важен параметър в бордовия компютър

Функционалност:

- Показване на текущото количество гориво в резервоара

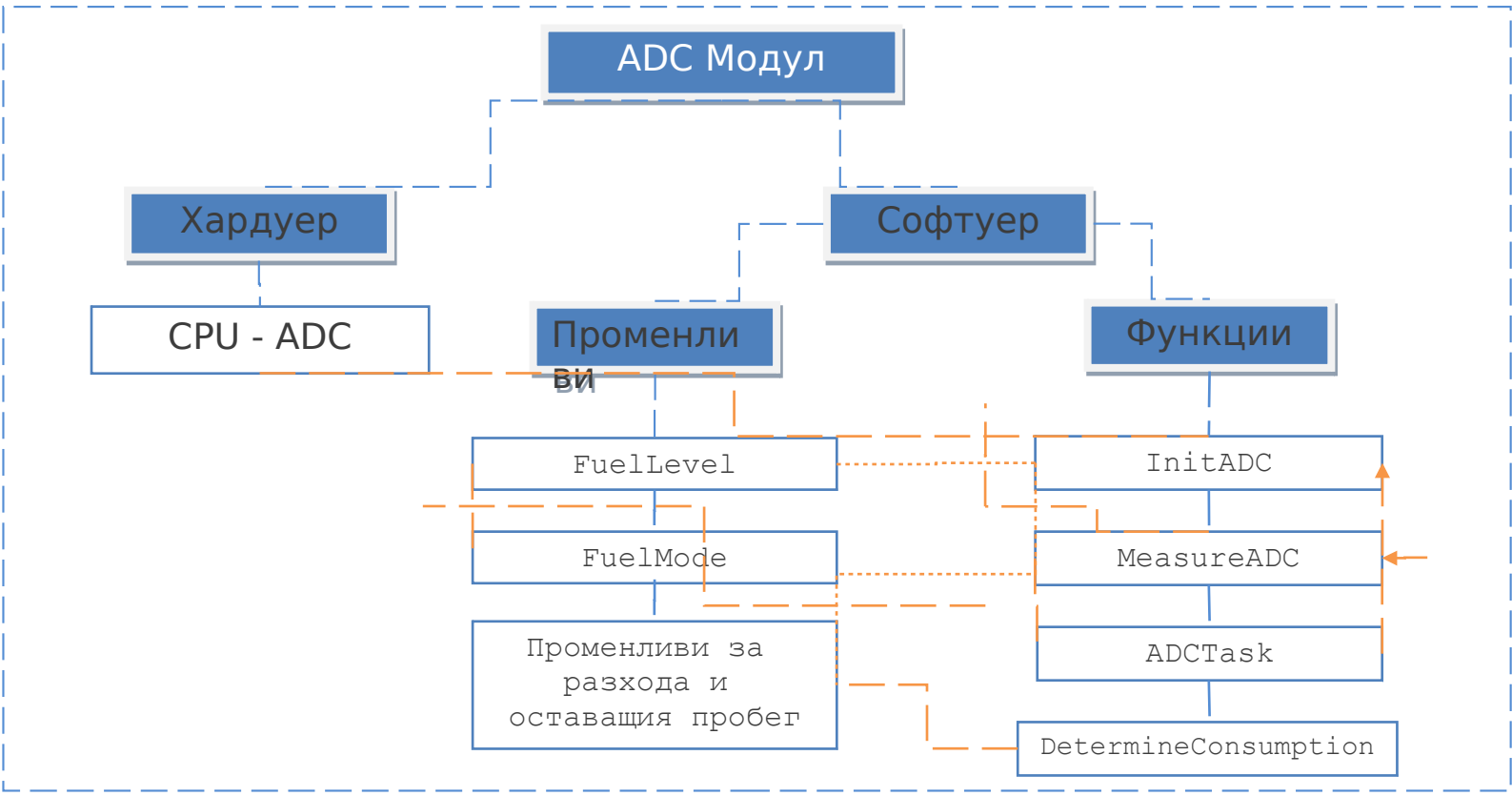
Използван хардуер:

- PORTF3 на процесора

Файлове с управляващия софтуер

- ADC.h
- ADC.c

Фигура 3.2.5.1: Модел на ADC модула



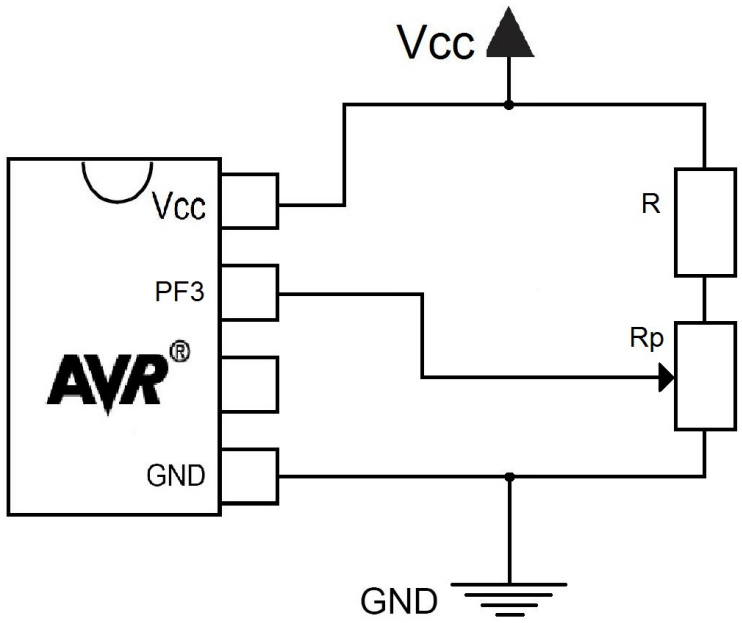
Описание на модула на резервоара

Хардуер

Хардуера, използван в този модул е PORTF3 на процесора, който е свързан към трети канал на ADC модула на процесора. Този пин е свързан към променливия извод на потенциометър, който за лабораторната постановка замества резистивната сонда, измерваща горивото (Сондата, представлява резистивен датчик, който изменя съпротивлението си според наличното количеството гориво в резервоара). Принципът на работа е един и същ. На фигура 3.2.5.2. е показана схемата на свързване. Използваният потенциометър (Rp), заместващ сондата е с големина 4,8KΩ, а на резисторът (R), добавен за защита на

входа на процесора от претоварване – 1,5 KΩ. Тази верига е захранена с напрежение от развойната платформа, т.е. 3.3V. По изчисления по Закона на Ом се получава, че токът протичащ по веригата е 524 μA.

Фигура 3.2.5.2. Схема на свързване



При максимална стойност на потенциометъра, падът върху него е 2,5V, което е 76% от съотнасящото напрежение (равно на захранващото), респективно 76% от максималната стойност на ADC регистъра (1023) след преобразуването, т.е. 783.

Софтуер

На модела на ADC модула е дадено разделението на софутерната част в реализацията на модула:

- Променливи – това са променливите съдържащи данните, получени от преобразуването на ADC модула

- Функции –реализират предназначението на модула, като показване на количеството гориво в резервоара в два режима – в литри или в проценти. Те реализират връзката между отделните софтуерни клонове, както и връзката хардуер-софтуер в модула.

Променливи

```
extern U16 FuelLevel;
extern U8 FuelMode;
extern U8 Consumption;
extern U8 Consumption_Dec;
extern U16 Range;
```

Това са променливите, които участват в изграждането на ADC модула за измерване количеството гориво в резервоара.

`FuelLevel` представлява количеството гориво в резервоара. Това число се получава след измерване на пада на напрежение върху потенциометъра. При максимална стойност на ADC регистъра с резултата (783), тази променлива показва пълен резервоар, т.е. 1050 литра или 100%, респективно при 0 показание на регистъра – 0 литра и 0%. Просто изчисление сочи, че на една стойност в регистъра отговарят 1,34 литра(783/1050). Така лесно може да се получи стойността на горивото, като се знае показанието на ADC регистъра.

`FuelMode` представя какъв е режима на показване на горивото – дали стойността да се показва в литри или в проценти. Алгоритъмът за изчисление е един и същ, но ако е избран режим с проценти се умножава стойността на

FuelLevel със 100% и се получава процентното изражение на количеството гориво в резервоара. Може да бъде една от следните стойности:

```
#define LITER_MODE      0
#define PERC_MODE       1
```

Променливите за разхода и оставащия пробег. Те съдържат информация за разхода на гориво в две части – целочислена и десетична. Оставащия пробег с се изчислява на базата на моментния разход и на оставащото количество гориво в резервоара.

Функции

Функциите в модула за резервоара отговарят за актуализирането на стойността на показваното количество гориво в резервоара, както и за управлението на модула като цяло. Функциите са декларирани в ADC.h и са имплементирани в ADC.c. Ето списък на всички функции, дефинирани в модула:

```
void InitADC();
void MeasureADC();
```

Инициализиране на модула - `void InitADC();`

Тази функция отговаря за подготвянето на модула за работа. В нея се извършват следните дейности:

- Избор на съотнасящо напрежение – AREF

```
//Enabling AREF as reference voltage
      cbi(ADMUX,REFS1);
      cbi(ADMUX,REFS0);
```

- Изчистване на досегашния избор на канал и избор на друг за измерване

```
//Clearing the channel selection bits
      ADMUX  &= ~(0x1F<<MUX0);
```

```
//Selecting Channel
      ADMUX |= 0x03;
```

- Изчистване на селекцията за делител и избор на делител

```
//Clear Prescaler
      ADCSRA &= ~(0x07);

//Select prescaler
      ADCSRA |= 0x07;
```

- Нулиране на стойността на `FuelLevel`

Измерване на количество гориво - `void MeasureADC();`

Тази функция извършва същинската работа на модула. В нея се включва ADC модула на процесора, стартира се аналогово-цифровото преобразуване и се извършва измерване. Стартирането на модула и на преобразуването се извършват със следния код:

```
//starting
      sbi(ADCSRA, ADEN);
      sbi(ADCSRA, ADSC);
```

Измерването се извършва в цикъл с брой итерации, равен на зададената прецизност със следната дефиниция:

```
#define ADC_PRECISION 2
```

Самото измерване се извършва от следния код:

```
      //Real measurement
      for(i=0;i<ADC_PRECISION;i++)
      {
            sbi(ADCSRA, ADSC);
            cbi(ADCSRA, ADATE);

            while(!(ADCSRA & (1<<ADIF)));
            sbi(ADCSRA, ADIF);
            sum+=ADC;
      }
```

Конвертирането се стартира на всяка итерация и се задава да се извърши само едно, а не след края на всяко да започва ново. С вложения цикъл се изчаква да завърши преобразуването и получената стоойност се прибавя към сумата, която след това се усреднява, на база на прецизността. Освен това с извършва и преобразуването на усреднената стойност в литри:

```
FuelLevel = ((sum/ADC_PRECISION/10)*135)/10;
```

Ако режимът е проценти то се извършва преобразуване на литрите в проценти:

```
#define TANK_CAP  1050
FuelLevel = (FuelLevel*10/(TANK_CAP/10));
```

Определяне на моментния разход и на оставащия пробег - `void DetermineConsumption(U16 tick_count);`

Според предадения параметър за броя на импулсите, показващи моментния разход, както и според изминатото разстояние се изчислява разходът на целевия автомобил на настоящата дипломна работа в литри на сто километра. Спрямо получения разход и текущото количество гориво в резервоара се изчислява пробегът, оставащ до следващото зареждане.

3.2.6. Модул на предавките, скоростта и оборотите

Скоростта на движение, оборотите на двигателя и текущата предавка са основни параметри, показвани на екрана на бордовия компютър.

Функционалност:

- Показване на скоростта на движение
- Показване на оборотите на двигателя
- Показване на текущата предавка
- Показване на изминатия пробег
- Показване на изминат пробег - общ
- Показване на изминатия пробег – временен
- Реализиране на съветник за превключване на предавките в два режима – икономичен и динамичен

Използван хардуер:

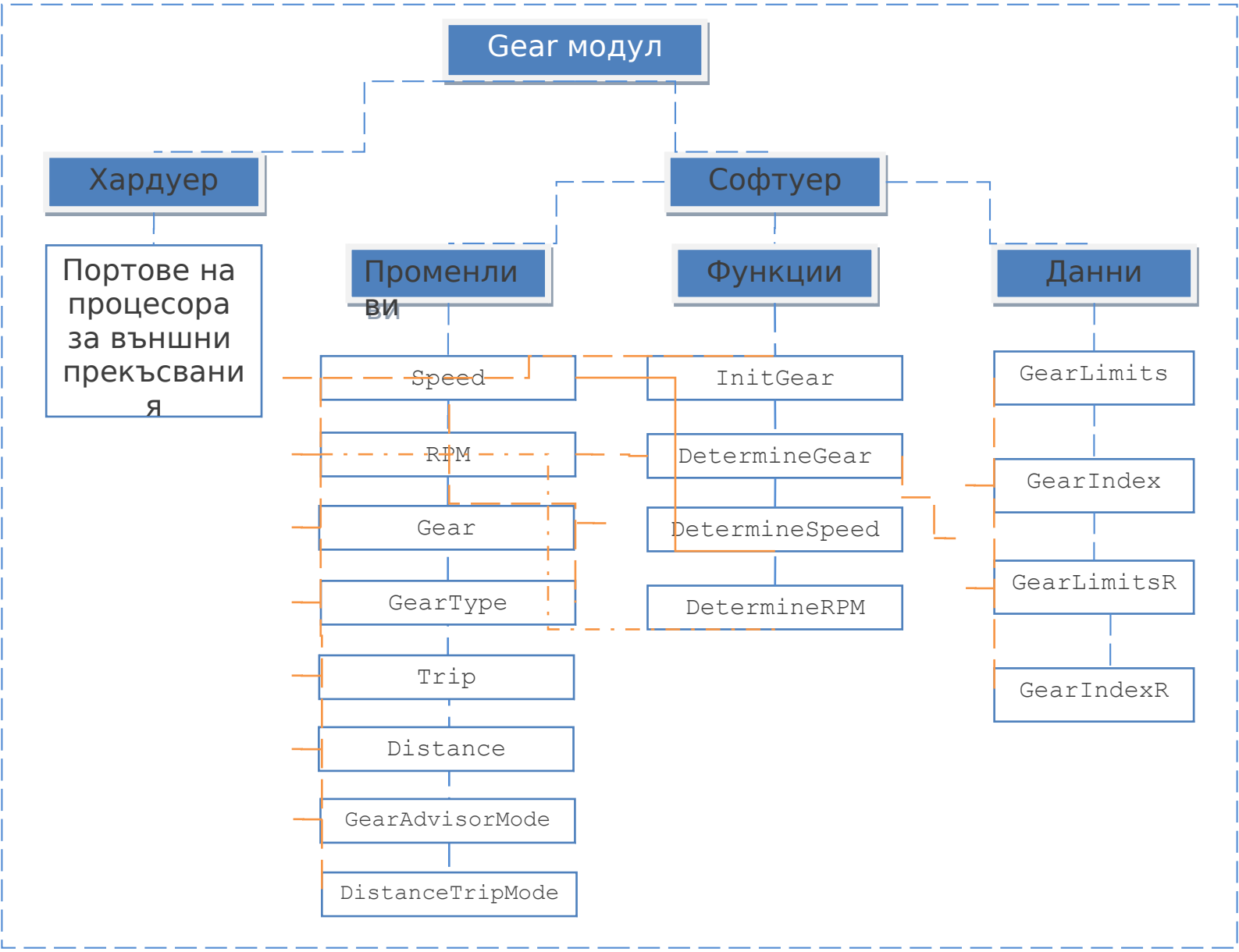
- Портове за външни прекъсвания на процесора

Файлове с управляващия софтуер

- gear.h
- gear.c
- gear_data.h

На фигура 3.2.6.1. е показан модела на модула, който реализира функционалностите за показване на текущите предавка, скорост и обороти.

Фигура 3.2.6.1. Модел на модула на предавките, скоростта



и оборотите

Описание на модула на резервоара

Хардуер

Хардуерът, използван в този модул са портове на процесора, реагиращи на промяна на логическото ниво на сигнала свързан към тях. Това е използвано в текущата дипломна работа за измерване на честота. Когато тези портове реагират на една и съща промяна в логическото

ниво (само от ниско във високо ниво или обратно), то броят на прекъсванията на съответния порт е равен на броя на импулсите на постъпващата поредица правоъгълни импулси. Чрез преизчисление според периода, за който са преброени тези импулсите, може да се изчисли каква е честотата на поредицата.

Софтуер

На модела на модула на предавките, скоростта и оборотите е дадено разделението на софутерната част от реализацията на модула:

- Променливи – това са променливите съдържащи данните за скорстта, оборотите, текущата предавка, данните за пробега, както и контролни променливи, които контролират работата на модула
- Функции – реализират предназначението на модула, като изчисляване и показване на стойностите на променливите. Те реализират връзката между отделните софтуерни клонове, както и връзката хардуер-софтуер в модула.

Променливи

```
//*************************************************************
// Gear, Speed & RPM variables
//*************************************************************
extern U8  Speed;
extern U16 RPM;
extern U8  Gear;
extern U8  GearAdvisorMode;
extern U8  GearType;
```

```
extern U16 Distance;
extern U16 Trip;
extern U8  DistanceTripMode;
```

На показания фрагмент код са дефинициите на променливите от разглеждания модул.

`Speed` е променливата, съдържаща скоростта на движение на автомобила. Поради спецификата на целевия автомобил тази скорост може да бъде само в интервала **0..90** (вж. т. 2.3.), поради електронният ограничител на скоростта. Теоретично максималната скорост на движение е **136 км/ч**.

`RPM` е променливата, съдържаща оборотите на двигателя в минута. Поради спецификата на целевия автомобил тези обороти могат да бъдат само в интервала **500..2500** (вж. т. 2.3.), поради електронният ограничител на оборотите.

`Gear` е променливата, съдържаща текущата предавка. Стойността на тази променлива може да бъде в интервала (**1..16**) за предавки за движение напред или **1** или **2** за предавки за движение назад. Предавките са 8, но всяка една от тях има бърза и бавна половинка и затова предавките са 16 за движение напред и 2 – на назад.

`GearAdvisorMode` е променливата, съдържаща, режима на работа на съветника за превключване на предавките. Може да приема една от стойностите:

```
#define ECONOMIC_MODE     2
#define SPORT_MODE        3
```

`GearType` съдържа вида на предавката – за движение напред или такава за движене назад. Използва се за

управление на функцията за определяне на текущата предавка. Може да приема една от стойностите:

```
#define FORWARD_GEAR  0
#define REVERSE_GEAR  1
```

`Distance` съдържа стойността на целия изминат пробег.

`Trip` съдържа стойността на временния изминат пробег.

`DistanceTripMode` съдържа стойност, която показва дали е избрана Distance или Trip да бъде показвана на екрана, поради факта, че е използвана една и съща позиция на екрана за двата параметъра и показването има се сменя чрез панела, реагиращ на допир.

Данни

Данните в този модул предсавяват таблици с информация, спомагаща за определянето на текущата предавка. Разпределени са в следните масиви:

- `GearLimits;`
- `GearLimitsR;`
- `GearIndex;`
- `GearIndexR.`

`GearLimits` представлява таблица с комбинации от скорост на движене, предавка и граници на оборотите, в които при движене с определената скорост, текущата предавка е тази от същия ред.

```
// Contains combination of Speed, Gear and RPM Limits for Forward
Gears
const U16 GearLimits[524][4] PROGMEM= {
          { 2, 1, 639, 586 },
          { 2, 2, 585, 532 },
          { 3, 1, 958, 879 },
          { 3, 2, 878, 723 },
```

```
{ 3, 3, 722, 593 },
{ 3, 4, 592, 539 },
```

...

Това е дефиницията и инициализацията на този масив. Първият ред от него дава следната информация: при движение с 2 км/ч и ако оборотите са в границите 586 – 639, то текущата предавка е първа бавна. Скоростите на движене започват от 2 км/ч поради факта, че това е минималната скорост на движение.

Изчислението на данните от тази таблица е извършено по формулата, дадена на фигура 3.2.6.2.

Фигура 3.2.6.2.: Формула за изчисляване на данните от таблицата `GearLimits`:

RPM = (GEAR*SPEED*0,017*4414)/4

Означения:

RPM – обороти на двигателя

SPEED – скорост в км/ч

GEAR – предавателно число на предавката(вж. т. 2.3.)

0,017 – определяне на скоростта в км/минута

4414 – брой импулси на датчика за 1000 метра.

4 – брой импулси на датчика за един оборот на кардана

При използването на тази формула по дадена скорост на движение и предавателните числа на предавките се намират съответните обороти, при които се извършва движението с дадената скорост и съответната предавка. Първо се намира скоростта на движение, но в км/минута,

след което се намира колко са оборотите на кардана. Така разликата между оборотите на кардана и тези на двигателя е предавателното число на съответната предавка на скоростната кутия. Това изчисление е направено с всяка стойност на скоростта и от таблицата са извадени стойностите на оборотите, намиращи се извън допустимите граници за този параметър. Валидните записи са **524**, което означава, че записите в тази таблица са **2096**.

След изчисляването на конкретните обороти на базата да на съседните записи се формират лява и дясна граница за конкретната предавка. Това става чрез разделяне на границата от обороти на две и едната част от нея се присвоява към оборотите за едната предавка и другата половина - за другата предавка.

Тези данни са изчислени предварително и са заложени в паметта за код на процесора. Това се налага с цел пестене на процесорно време, което процесорът би изразходвал за изчисляването на тези данни всеки път, когато е необходимо да се определи коя е текущата предавка.

`GearIndex` представлява таблица с индексите на всяка една от възможните скорости да движение(2..90км/ч) в масива с границите на оборотите `GearLimits`. Това се използва, за да се намали процесорното време за определяне на текущата предавка, като се намалява времето за търсене `GearLimits`.

Масивите `GearLimitsR`, `GearIndexR` са използвани със същото предназначение като `GearLimitc`, `GearIndex`, но за движение назад, докато другите два са за движение напред.

Функции

Функциите в този модул отговарят за определяне на текущата предавка, за скоростта, оборотите, както и за двата вида пробег.

Инициализиране на модула - `void InitGear (void);`

В тази функция се инициализират всички променливи на модула – както тези, които се визуализират, така и сервизните такива.

Определяне на текущата предавка - `void DetermineGear(void);`

Тази функция служи за определяне на това коя е текущата предавка на базата на скоростта на движение и оборотите на двигателя. Ако скоростта на движение е под 2 км/ч, то функцията определя текущата предавка на нулевата – неутрална. Ако не се намери съвпадение на скорост и обороти, то отново предавката е неутрална. Представеният фрагмент код показва алгоритъмът на търсене в таблицата `GearLimits` или `GearLimitsR`.

```
for(i=0; i<(pgm_read_word(&GearIndex[Speed-1]) -
      pgm_read_word(&GearIndex[Speed-2])); i++)
{
      if((RPM >= pgm_read_word(&GearLimits
            [pgm_read_word(&GearIndex[Speed-2])+i][LEFTLIMIT]))
            && (RPM <=pgm_read_word(&GearLimits
            [pgm_read_word(&GearIndex[Speed-2])+i][RIGHTLIMIT])))
```

```
	{
		//Return determined gear
		Gear = (U8)(pgm_read_word(&GearLimits
			[pgm_read_word(&GearIndex[Speed-2])+i][GEAR]));
		break;
		//If no gear was found - return neutral gear
	}
	else
		Gear = 0;
}
```

Цикълът се изпълнява толкова пъти, колкото са и записите за съответната скорост за движение в масива с границите на оборотите. За всеки запис се проверява дали текущите обороти на двигателя с в границите от текущия запис. Ако това е изпълнено за текуща предавка се присвоява намерената от функцията. В противен случай текущата предавката е неутрална.

Определяне на скоростта на движение - `void DetermineSpeed(U8 tick_count)`

Тази функция отговаря за изчисляването на скоростта на базата на броя импулси, отброени чрез портовете, реагиращи на външни събития – честотата, генерирана от датчика. Броя на импулсите са записани в константата:

```
#define KM_PULSES		4414
```

Броят импулси, получени чрез аргумента на функцията и тази константа определят какво е изминатото разстояние за времето, за което са преброени импулсите. Това е скоростта на движение на автомобила.

Определяне на оборотите на двигателя - `void DetermineRPM(U16 tick_count);`

Тази функция отговаря за изчисляването на оборотите на двигателя на базата на броя импулси, отброени чрез

портовете, реагиращи на външни събития – честотата, генерирана от датчика. Броя на импулсите са записани в константата:

```
#define RPM_PULSES          120
```

Времето, за което са преброени тези импулси се съотнася със една минута. Броят импулси, получени чрез аргумента на функцията се умножават, за да се получи броя импулси за минута и чрез тази константа се определят оборотите на двигателя за минута.

3.2.7. Главен модул

Главният модул има много важна функция в реализирането на бордовия компютър. Той обединява работата на останалите модули - подготвя ги за работа и контролира изпълнението им.

Функционалност:

- Връзка между отделните модули

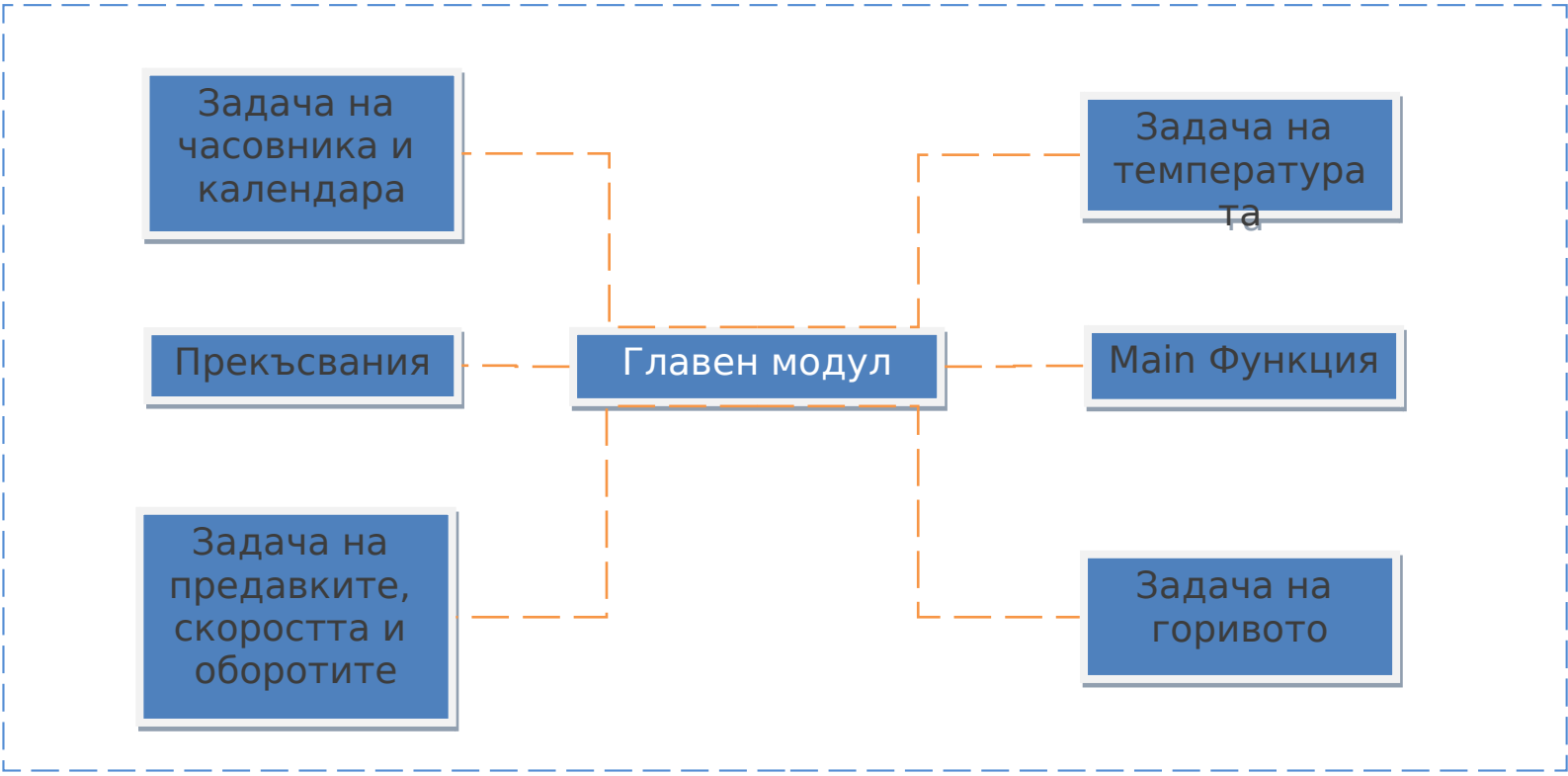
Използван хардуер:

- Няма такъв

Файлове с управляващия софтуер

- c_startup.c

Фигура 3.2.7.1. Описание на Главния модул



Задача на часовника и календара

Тази задача има за цел да управлява изпълнението на модула на часовника и календара, да го подготви за работа.

```
AVRX_TIMER(ClockTaskTimer);
AVRX_GCC_TASKDEF(ClockTask, 200, 5)
```

Това е дефиницията на задачата. Приоритетът е пети и са отделени 200 байта за работата ѝ.

Задачата се разделя на два големи клона. Това разделение зависи от това дали часовникът е в режим на работа или в режим на сверяване. Ако е в режим на работа, то `tobeAdjusted == 0`. В тази част се извеждат стойностите на часовника и календара, извиква се функцията за реализиране на функционалността на задачата `ClockTick()`,

реализира се времето между извикванията на основната функция на модула, както и се следи за това дали е натиснат панела, реагиращ на допир за влизане в режим на сверяване. Отброяването на времето между извикванията на функцията се извършва от следния фрагмент код:

```
// All functions in the cycle without AvrXDelay() take 38 ms
AvrXDelay(&ClockTaskTimer,mConvertMsToTicks(962));
```

Следенето на панела, реагиращ на допир се извършва чрез следния фрагмент код:

```
if((TouchScreenValueX>=190)&&(TouchScreenValueX<=870)&&
(TouchScreenValueY>=878)&&(TouchScreenValueY<=905))
{
            tobeAdjusted = 1;
}
```

Режимът на модула се сменя и при следващото минаване през функцията ще се изпълни другият клон на задачата – този за режим на сверяване. При първото изпълнение на този клон, т.е. след натискането на областта от екрана, където са изведени данните на часовника, се изчиства екрана, спира се часовника и се извеждат панелите с бутони, необходими за извършването на сверяването на часовника. Конкретните действия за сверяването са описани подробно на страницата на четвърта глава на настоящата дипломна работа. Правят се четири различни прочитания на стойностите за toouchsreen-а с цел да се засече натискането на една от четирите стрелки, които сменят показалеца, посочващ параметъра, който да се променя или променят посочения параметър. Съблюдава се параметрите на календара и часовника да не надминават допустимите си

стойности(напр. дата 33). При опит да се надвишат тези допустими стойности не се извършва действието, заявено от потребителя.При натискане на бутона OK часовникът и календара се стартират отново, като освен това режимът на работа се променя на нормален от такъв за сверяване, за да се изпълни другият клон на задачата при следващото ѝ изпълнение. Екранът се изчиства и продължава нормалната работа на бордовия компютър.

Задача на температурата

Тази задача има за цел да управлява изпълнението на модула на температурата, да го подготви за работа.

```
AVRX_TIMER(MainTaskTimer);
AVRX_GCC_TASKDEF(MainTask, 200, 6)
```

Това е дефиницията на задачата. Приоритетът е шести и са отделени 200 байта за работата ѝ.

Тази задача се стартира първа от функцията Main. Тя има за цел да инициализира модулите на бордовия компютър и да стартира някой от тях. Освен това тази задача стартира останалите задачи. Това се извършва чрез следния фрагмент код:

```
// do some initialization here
LCDInitialize();

LCDStart();

TouchScreenInit();

TouchScreenStart();

TWIInit();

TWIStart();

// start another task
```

```
AvrXRunTask(TCB(ClockTask));
AvrXRunTask(TCB(ADCTask));
AvrXRunTask(TCB(GearTask));
```

Изпълняват се действия по извеждането на температурата, получена като информация от термодатчика. В тази задача се стартира модулът за температурата и веднъж на три секунди се извежда стойността на температурата и се стартира отново. При едно стартиране се извършва едно измерване и едно извеждане. При отрицателна стойност на температурата, то се показва знак минус на екрана и се преобразува получената от датчика информация. Ако количеството на горивото в резервоара е ниско и е наложително показване на иконата „колонка”, сигнализиращата това, то не се извежда температурното показание.

Прекъсвания

В настоящата дипломна работа те се използват за задаване на времева база на операционната система, за отброяването на интервали от време, за броене на събития.

Времевата база на операционната система се задава чрез прекъсването на таймер нула. То се генерира веднъж на всеки 100 микросекунди.

Прекъсвания се използват още за броене на събития. Това е нужно за измерането на честотата на поредица правоъгълни импулси, необходимо за показванията на параметрите като скорост, обороти и разход.

Прекъсвания се използват и за отброяване на времеви интервали с таймери. Това се изпозползва за отброяването

на времето между записванията на данните за двата вида пробег в енергонезависимата памет на процесора.

Main Функция

Тази функция се изпълнява първа след подаване на захранване на развойната среда. Тя има за задача да инициализира хардуера за работа. Това включва настойка на таймерите, настройка на входно-изходните пинове на портовете на процесора, както и стартиране на Main задачата. Това се извършва със следния фрагмент код:

```
AvrXRunTask(TCB(MainTask));      // start the program main task
```

След това се превключва контекста към този на първата задача, тоест на стартираната по-горе. Това е началото на работата ѝ.

Задача на предавките, скоростта и оборотите

Тази задача има за цел да управлява изпълнението на модула на предавките, скороста и оборотите, да го подготви за работа.

```
AVRX_TIMER(GearTaskTimer);
AVRX_GCC_TASKDEF(GearTask, 50, 3)
```

Това е дефиницията на задачата. Приоритетът е трети (най-високият спрямо другите задачи)и са отделени 50 байта стек за работата ѝ.

Както останалите задачи, така и тази се изпълнява, ако модулът на часовника и календара не е в режим на сверяване, а е в нормален работен режим.

След инициализиране на модула се пристъпва към отпечатване на стойностите на параметрите, показвани от модула:

- Скорост – през 5 единици;
- Обороти – през 50 единици
- Текуща предавка
- Пробег-общ/Пробег-временен

Поради факта, че двата вида пробег се показват на едно и също място е необходимо да се провери кой от двата е избран да бъде показан. С натискане на touchscreen-а в тази област се сменя показанието, както и съответния етикет. Тези два пробега се записват в енергонезависимата памет на процесора веднъж на определен интервал от време. Наред със стойностите се отпечатват и етикетите и мерните единици.

На база на съответните режими се отпечатват и индикациите за вида скорост, както и за режима насъветника за превключване на предавките:

```
//Indications for Gear Advisor Mode
if(GearAdvisorMode == ECONOMIC_MODE)
{
     LCDWriteIcon(18, 12, ECONOMIC_MODE);
}
else
{
     LCDWriteIcon(18, 29, SPORT_MODE);
```

```
}

//Indication for Reverse Gear
if(GearType == REVERSE_GEAR)
{
      LCDWriteIcon(11, 12, R_GEAR);
      LCDWriteIcon(13, 17, CLR_GEAR_ADV);
      LCDWriteIcon(13, 30,  CLR_GEAR_ADV);
}
else
      LCDWriteIcon(11, 12, CLR_SMALL_DIGIT);
```

Текущата предавка се определя от фунцкията:

```
//Function, that determines which is the current gear
DetermineGear();
```

След извикването ѝ вече е намерена предавката, съответстваща на текущата скорост и обороти. Тя вече може да бъде показана. Ако стойността е нула, то се показва неутрална предавка, ако не е се показва предавката, като за половинките са използвани малки стрелки.

Използвани са външни хардуерни сигнали, представени на лабораторната постановка чрез бутони, които определят дали е включена предавка за движение напред или назад.

Съветникът за превключване на предавките е друга функционалност на задачата. При избран икономичен режим съветникът подканва водачът да шофира в диапзона от обороти от 1000 до 1500 оборота. При избран динамичен режим – оборотите от 1500 до 2000. При текущи обороти под желаните се дава съвет за превключване на предавка надолу. В противен случай – текущи обороти над желаните – съвет нагоре. Логиката на работа на двата

режима е идентична, като единствената разлика е границите от обороти.

Панелът, реагиращ на допир се използва за смяна на режима на работа на съветника за превключване на предавките, както и за смяна не пробега за показване.

Задача на горивото

Тази задача има за цел да управлява изпълнението на модула на горивото, да го подготви за работа.

```
AVRX_TIMER(ADCTaskTimer);
AVRX_GCC_TASKDEF(ADCTask, 50, 4)
```

Това е дефиницията на задачата. Приоритетът е четвърти и са отделени 50 байта за работата ѝ.

Тази задача извежда количеството гориво в резервоара, моментния разход и оставащия пробег до слеващото зареждане.

Количеството гориво в резервоара може да бъде показвано по два начина – в литри или в проценти. При падане на нивото на резервоара под 147 литра или 14% се извежда индикация под формата на колонка за гориво. Режимът на показанието на количеството гориво в резервоара се сменя с използване на панела реагиращ на допир.

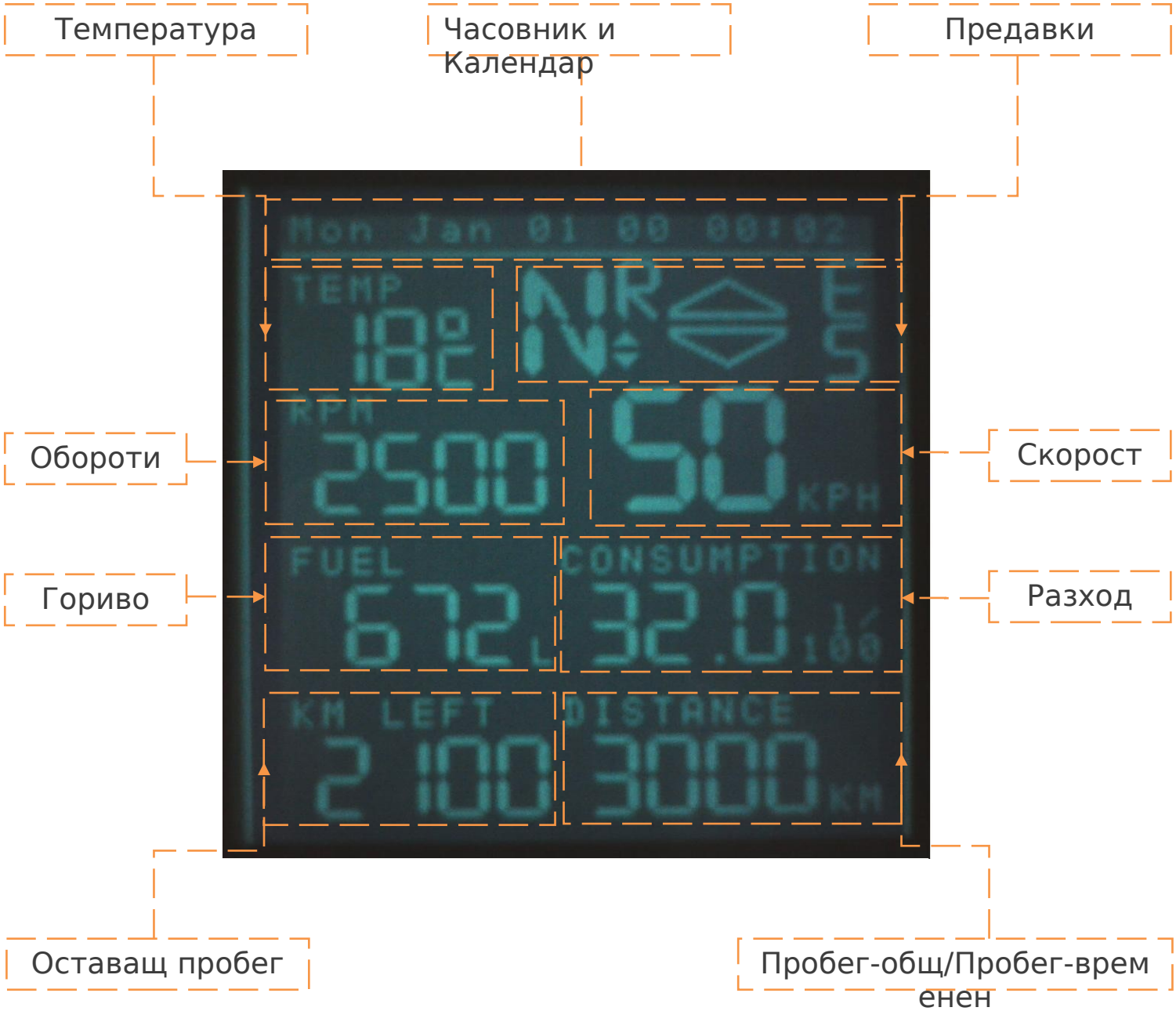
Моментният разход на гориво се извежда в две части – целочислена и десетична с точност до един знак след десетичната точка.

Оставащия пробег до следващото зареждане с гориво се изчислява на базата на другите параметри. Чрез разделяне на количестово гориво в резервоара и разходът

на автомобила за 100 км, и след това отново се умножи със 100, то се получават километрите, които могат да бъдат изминати с текущото количество гориво в резервоара и текущият моментен разход.

Глава IV

Четвърта глава на настоящата дипломна работа представлява „Ръководство на потребителя”. Ще бъдат описани всички показания на параметри от бордовия компютър, всички видове индикации, както и всички места, където е възможна намеса от страна на потребителя.



Фигура 4.1.: Разделение на екрана на бордовия компютър на секции

На фигура 4.1. ясно се вижда разделението на екрана на бордовия компютър на отделни секции:

- Секция „Часовник и календар”
- Секция на „Температура”
- Секция на „Предавки”
- Секция на „Скорост”
- Секция на „Обороти”
- Секция на „Гориво”
- Секция на „Разход”
- Секция „Оставащ пробег”
- Секция „Пробег-общ/Пробег-временен”

4.1. Секция „Часовник и календар”

Секцията на часовника е разположена в най-горната част на екрана на бордовия компютър.

Фигура 4.1.1: Разположение на данните на часовника и календара

На фигура 4.1 е дадено разположението на данните на часовника и календара:

- **Ден от седмицата** – може да бъде една от следните стойности:
  - "**Mon**" - понеделник
  - "**Tue**" – вторник
  - "**Wed**" - сряда
  - "**Thu**" - четвъртък
  - "**Fri**" - петък
  - "**Sat**" - събота
  - "**Sun**" - неделя
- **Месец** – може да бъде една от следните стойности:
  - "**Jan**" - януари
  - "**Feb**" - февруари
  - "**Mar**" - март
  - "**Apr**" - април
  - "**May**" - май
  - "**Jun**" - юни
  - "**Jul**" - юли
  - "**Aug**" - август
  - "**Sep**" - септември
  - "**Oct**" - октомври
  - "**Nov**" - ноември
  - "**Dec**" декември
- **Дата** – може да бъде число в интервала **(1; 30)**, **(1; 31)**, **(1; 28)**, **(1; 29)** в зависимост от това кой е

месецът от годината и дали годината е високосна.

- **Година** – поради факта, че са отделени две позиции, може да бъде показвана година от 0 до 99. Настоящата година 2009 ще бъде показана като 09.
- **Часове** – режимът на работа е 24 часов, което означава, че часовете, могат да са от 00 до 23.
- **Минути** – стойността може да бъде от 00 до 59 минути

Часовникът и календара трябва да могат да се сверяват. Тук е използван панелът, реагиращ на допир на развойната хардуерна платформа. При упражняване на натиск в областта от екрана, показана на фигура 4.1.1.в продължение на около секунда с пръст, стилус, химикалка или какъвто и да е предмет, се появява екрана за

сверяване на часовника и календара. Този екрана е показан на фигура 4.1.2.

Фигура 4.1.2.: Екран за сверяване на часовника и календара

На фигура 4.1.2. ясно се вижда начина на сверяване на показанието на часовника и календара.

С натискане на бутоните за „**Настройка на параметър**”, избраният параметър се променя на базата на натиснатия бутон (тази област от Touchscreen-а). При натискане на бутона „**Нагоре**”, параметърът се увеличава, при натискане на бутона „**Надолу**”, параметърът намалява.

С натискане на бутоните за „**Избор на параметър**”, показалецът се движи и така се избира параметърът, който да се настройва с бутоните „**Настройка на параметър**”. Показалецът представляват малки стрелки под параметърът, който е избран за настройка. При натискане на бутона „**Наляво**”, показалецът се придвижва на ляво, аналогично с бутона „**Надясно**” се придвижва надясно.

С бутонът за потвърждение **“OK”** се запазват промените по параметрите на календара и часовника и на екрана се показва основният екран на бордовият компютър.

При увеличаването и намаляването на стойностите на параметрите на часовника и календара, се съблюдава да не с надхвърлят допустимите стойности на всеки един от

тези параметри. Ситуациите, които се съблюдават са следните:

- Денят от седмицата и месецът да не надвишават изброените си стойности, посочени по-горе в тази точка
- Датата да не надвишава броя на дните в съответния месец. Включително и месец февруари да не надвишава дните си, както за високосна така и за невисокосна година
- Часовете да не надвишават стойността 23
- Минутите да не надвишават стойността 59
- Всички стойности са числа, равни или по-големи от нула

4.2.Секция „Температура”

На фигура 4.2.1 е показана секцията „Температура”.

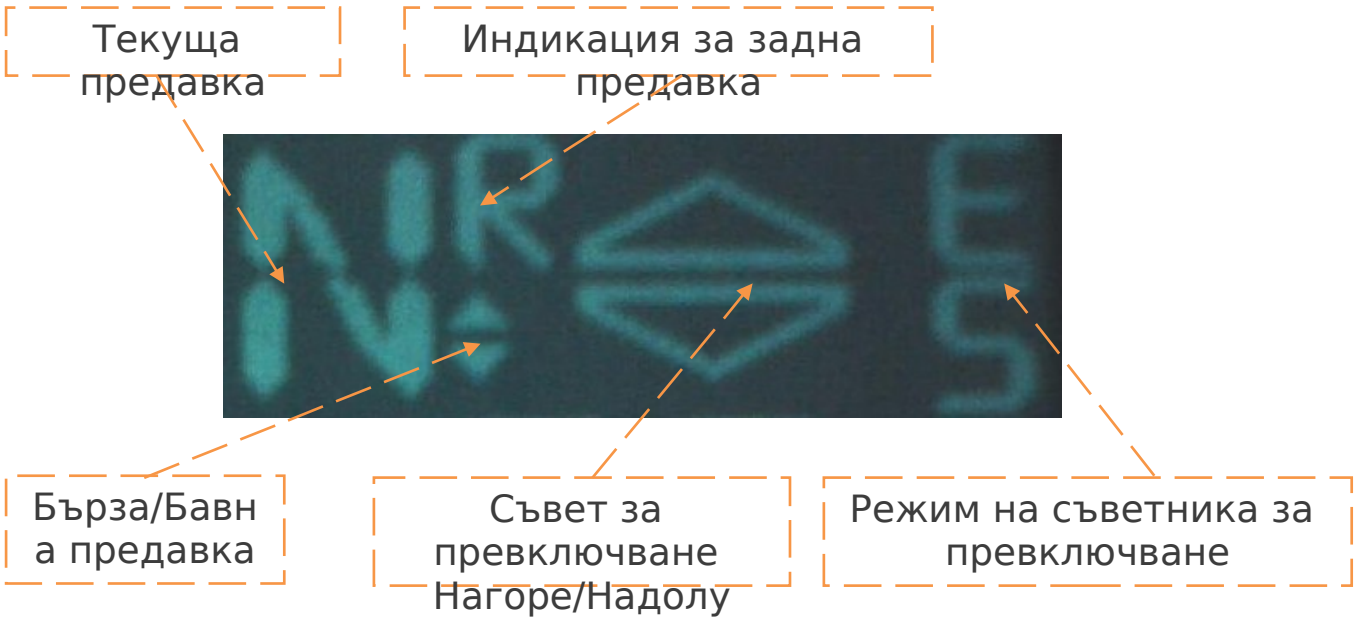
Фигура 4.2.1.: Секция „Температура”

Температурата, показвана от тази секция е в границите (-40; +125). Мерната единица е градус Целзий. При отрицателни температури се визуализира минус през цифрата, обозначаваща температурата.

4.3.Секция „Предавки”

На фигура 4.3.1 е показана секцията „Предавки”. „Текущата предавка” показва коя е включената в момента предавка. Може да заема стойности в интервала (1; 8). Освен това, когато не е включена нито една предавка се показва голяма буква N, а при включена задна предавка, голяма буква R. Малките стрелки от „Бърза/Бавна предавка” показват дали настоящата предавка е бързата или бавната част на предавката, показана в „Текуща предавка”. Може да се показва само една стрела от „Бърза/Бавна предавка” едновременно, както и е възможно само показването или на N, или на R, но не и едновременното им показване.

Фигура 4.3.1.: Секция „Предавки”



Големите стрелки от „Съвет за превключване Нагоре/Надолу” показват в коя посока трябва да се превключи предавката, за да се спазва режима на каране. Едновременно може да бъде показвана само една от тези две стрелки.

Двете големи букви E и S показват какъв е режим на съветника за превключване на предавките

- E – икономичен;
- S – динамичен.

Режимите се сменят чрез докосване на панела, реагиращ на допир в областта показана на фигура 4.3.1. Натискът може да се упражнен с пръст, химикалка, стилус или друг предмет. Не могат да се показват едновременно и двете букви, обозначаващи режима на съветника за превключване на предавките.

4.4. Секция „Обороти”

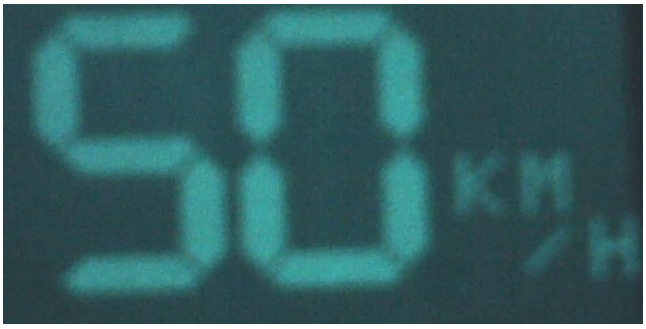
На фигура 4.4.1. е показана секцията „Обороти”.

Фигура 4.4.1.: Секция „Обороти”

В тази секция се показват оборотите на двигателя. Стойностите, които могат да бъдат показвани са в интервала (500; 2500). Поради факта, че 500 е празният ход на двигателя, а 2500 са максималните обороти. При достигането им осветлението на екрана се включва, за да се отбележи екстремната ситуация.

4.5. Секция „Скорост”

На фигура 4.5.1. е показана секцията „Скорост”



Фигура 4.5.1.: Секция „Скорост”

В тази секция се показва скоростта, с която се движи автомобила. Възможните стойности са в интервала (0, 90), поради факта, че електронноограничената скорост на камиона е 90 км/ч.

Мерната единица, която е показана е “КМ/Н” или „км/ч”.

4.6.Секция „Гориво”

Секцията гориво дава информация за количеството гориво в резервоара. Тази секция е дадена на фигура 4.6.1.

Фигура 4.6.1.: Секция „Гориво”

Съществуват два режима на работа на тази секция – абсолютен и относителен. Показаното на фигура 4.6.1. е абсолютният режим на секцията, това означава, че показанието е в литри и има граници интервала (0; 1050), като 1050 е максималната вместимост на резервоара.

Режимът се сменя чрез докосване на областта, която е показана на фигура 4.6.1.. Другият режим е относителният, което означава, че показанието се изразява в проценти и съответно интервала на стойностите е (0, 100) %. При намаляване на количеството гориво до стойността от 14% или 147 литра или 14% (в зависимост от режима на работа) на мястото на секцията на температурата се показва горивна колонка, показваща, че вече е наложително зареждането на резервоарите с гориво. На фигура 4.6.2. е показана тази иконка – горивна колонка.

Фигура 4.6.2.: Иконка „Горивна колонка”

4.7. Секция „Разход”

Секцията разход е показана на фигура 4.7.1.

Фигура 4.7.1.:Секция „Разход”

Показва се моментният разход на гориво, като точността на показанието е до една цифра след десетичната запетая. Ограничения в показанието на този параметър няма. Мерната единица е l/100km.

4.8.Секция „Оставащ пробег”

Секцията оставащ пробег показва пробега, който може да бъде изминат с текущото количество гориво в резервоара и с текущият моментен разход. На фигура 4.8.1. е показана 

тази секция.

Фигура 4.8.1.: Секция „Оставащ пробег”

Тази секция е пряко свързана с показанието на секциите „Разход” и „Гориво”.

4.9. Секция „Пробег-общ/Пробег-временен”

На фигура 4.9.1. е показана тази секция

Фигура 4.9.1.
Секция „Пробег-общ/Пробег-временен”

В тази секция се показва цялото пропътуваното разстояние до момента (Пробег-общ) или пропътуваното разстояние от последното му изчистване (Пробег-временен). Смяната на режимите се извършва чрез панела, реагиращ на допир с докосване в тази област. При смяната на режима се сменя показанието, както и етикета на тази секция. Изчистването на показанието на временния пробег се извършва чрез бутон, монтиран към хардуерната платформа на бордовия компютър.

Заключение

В настоящата дипломна работа бяха разгледани множество страни от програмирането за вградени микропроцесорни системи.

Беше създаден вграден софтуер, който да управлява електронния блок на бордовия компютър. Нуждата от такова устройство нараства все повече, поради факта, че то предоставя множество информация на водача на превозното средство в реално време. Това водачът да е информиран е от особена важност, за да е възможно пътуването да е безопасно и ефективно.

Бъдещото развитие на настоящата дипломна работа би било разширяването на функционалността на електронния блок. Това би могло да се изразява в:

- Реализиране на функция тахограф – водене на непрекъсната справка за скоростта на автомобила и записването на информацията на текстов файл на карта памет
- Използване на акселерометричния датчик и даването на непрекъсната справка, която да се записва отново на карта памет
- Добавяне на термометър, който да измерва и вътрешната температура в салона на целевия автомобил.

Надявам се дипломната работа да е постигнала целите си и това да бъде оценено.

Използвана литература

1.Информация за целевия автомобил:

- **http://en.wikipedia.org/wiki/Mercedes-Benz_Actros**
- **http://de.wikipedia.org/wiki/Mercedes-Benz_Actros**
- **http://www.mbbi.co.uk/MBBISec/Default2.aspx**
- **Инструкция за експлоатация**

2.Информация за хардуерната платформа

- **http://www.olimex.com/dev/avr-tlcd-128can.html**
- **http://www.olimex.com/dev/pdf/AVR/AVR-TLCD-128CAN.pdf**
- **http://www.olimex.com/dev/pdf/AVR/AVR-TLCD-128CAN-SCH-REV-A.pdf**
- **http://en.wikipedia.org/wiki/Palm_III**
- **Спецификация на процесора Atmel AVR AT90CAN128**

3.Информация за използваните средства за разработка:

- **http://www.barello.net/avrx/AvrX-2.6/index.htm**
- **http://winavr.sourceforge.net/index.html**

- **http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725**

4.Информация за използван допълнителен хардуер

- **Спецификация на термодатчика TCN75 - http://ww1.microchip.com/downloads/en/DeviceDoc/21490C.pdf**

5.Информация за използваните вградени модули

- **http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=50106**
- **http://en.wikipedia.org/wiki/I%C2%B2C**
- **http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=56429&highlight=tut+twi**

Съдържание

Увод.....4
Глава I.....6
1.1.Проучване на подобни решения.....6
1.2.Проучване на целевия автомобил.....10
1.3. Проучване на хардуерни платформи.....11
1.4. Проучване на среди за разработка на вграден софтуер.....13
1.5. Проучване на езици за разработване на вграден софтуер.....15
1.6. Проучване на RTOS.....15
Глава II.....17
2.1. Дефиниране на изискванията към предмета на дипломната работа.....17

2.2. Средствата за разработка.....21
2.3. Детайлен преглед на целевия автомобил.....27
2.3.1.Техническа информация за целевия автомобил.....28
2.3.2. Специфична информация за начина, по който целевия автомобил измерва работните параметри.....34
2.4. Детайлен преглед на развойната хардуерна платформа OLIMEX AVR TLCD128CAN.....36
2.5. Детайлен преглед на използваните вградени модули.....40
Глава III.....52
3.1. Обзор.....52
3.2. Същинска част.....53

3.2.1. Модул на LCD екрана.....53
3.2.2. Модул на панела, реагиращ на допир.....81
3.2.3. Модул на часовника и календара.....90
3.2.4. Модул за температурата.....96
3.2.5. Модул за количеството гориво в резервоара.....103
3.2.6. Модул на предавките, скоростта и оборотите.....109
3.2.7. Главен модул.....118
Глава IV.....127
4.1. Секция „Часовник и календар”.....128
4.2.Секция „Температура”.....132
4.3.Секция „Предавки”.....132

4.4. Секция „Обороти”.....134
4.5. Секция „Скорост”.....134
4.6.Секция „Гориво”.....135
4.7. Секция „Разход”.....136
4.8.Секция „Оставащ пробег”.....137
4.9. Секция „Пробег-общ/Пробег-временен”.....137
Заключение.....139
Използвана литература.....140
Използвани означения.....143

Използвани означения

Означение	Описание
TWI	Two-Wire Interface

ADC	**Analog-to-Digital Converter**
RTOS	**Real-Time Operating System**
IC	**Input Capture**