СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Развитие прогресса во второй половине XX века сопутствовал вместе с противостоянием сверхдержав в холодной войне. В XXI веке эти достижение продолжили развиваться, но в мирное русло, применяя достижение прошлого и нового веков в мирных целях. Многие вещи, которые используются в повседневном обиходе у людей, произошли от достижений в военно промышленного комплекса: пластик, липучки, обезболивающие средства, лейкопластырь, микроволновая печь и т.д.

Сферу высоких технологий электронную отрасль эти процессы также не обошли стороной. Достижение прошлого века, придуманные в первую очередь для эффективного ведения боевых действий, в нашем веке стали все больше применяются в мирных сферах жизни людей. Все более и стремительнее. Современные достижения в области высоких технологий основаны именно на этом прогрессе прошлого. Конечно, они морально устарели, зато создали отличную основу для развития новых технологий.

А в тоже время, в современном обществе произошло развитие и улучшение социального положения человека. Все больше и больше людей способны приобрести то, что ранее считалось невозможным среднему классу и было доступно только обеспеченному слою населения. Одним из предметов является автомобиль. Все больше и больше на сегодняшний день появляется автомобилистов, в крупных городах все чаще и чаще в часы пик стали появляется автомобильные пробки, а также дорожно транспортных происшествий.

Современные автомобили оснащаются всеми современными устройствами индикации, которые сигнализируют об окружающей обстановке. Но до сих пор у конструкторов автомобилей не сформировался ответ на главный вопрос: так какая же информация необходима и важна для глаз водителя? Приборный интерфейс постоянно модернизируется и совершенствуется. Необходимость информировать водителя о любых изменениях показателей движения автомобиля и окружающей обстановки с целью улучшения безопасности движения и комфортной поездки. И в тоже время не перегружать водителя лишней информацией и временем за просмотром всех этих показателей. Ведь пользоваться водитель ею должен во время движения транспортного средства.

Еще одной проблемой, для многих пользователей транспортным средством, является контроль скорости. В современных автомобилях продолжают использоваться стрелочные индикаторы скорости, что не совсем удобно при быстрой езде. Так как внимание уделяется на то чтобы посмотреть текущую скорость, при условии, что не на всей шкале нанесены цифры. Водитель должен с помощью одного взгляда узнать свою текущую скорость. Также, все производители автомобилей намеренно изменяют показание приборов по контролю скорости. Спидометры показывают завышенные показатели скорости. Все это связано с «психологическим эффектом» -

уберечь водителя от превышения скорости. Но если смотреть объективно, то больше 60% нарушений водителей в Республике Беларусь является превышение скорости. Искоренить это невозможно. Научить культуре вождения - тоже. Но можно их предостеречь.

Целью данного дипломного проекта является разработка комплекса по измерению скорости объекта. Данный комплекс будет работать на основе системы измерения геолокации. Также в нем будет присутствовать минималистичный оригинальный интерфейс, с возможностью автоматической корректировки яркости, чтобы в темноте данный интерфейс не ослеплял водителя. Также, есть и дополнительные функции: измерение напряжения на аккумуляторной батарее автомобиля, часы реального времени с корректировкой, а также количество видимых устройством спутников.

Данное устройство позволит получать необходимую в движении автомобиля информацию о нем. Предназначено для любого наземного транспортного средства.

Исходя из поставленной цели, были сформулированы следующие задачи:

1) Проанализировать литературу.
2) Составить структурную и функциональную схему прибора.
3) Анализ и выбор необходимых компонентов.
4) Разработка структуры программы управления контроллером.
5) Составить электрическую принципиальную схему.
6) Написание программного кода управления контроллером.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Общие сведения

С развитием в конце XIX – начале XX вв. механической техники дало толчок к появлению и развитию приборов, которые позволяли мерить необходимые показатели. Одним из таких приборов является спидометр. Он входит в число обязательных приборов, которые необходимы для автомобилиста. И не только: контроль скорости используется практически во всех самодвижущихся средствах (на земле, на воде, в воздухе). Спидометр – измерительный прибор, способный определять модуль мгновенного движения тела относительно земли.

В первой половине XX века спидометр считался очень необычной вещью и экстраординарным прибором. Однако с развитием автомобилестроения, транспортные средства того времени стали способны все большие и большие скорости перемещения. Под влиянием этих факторов, а также увеличением дорожно-транспортных происшествий с участием автомобилей, было принято решение устанавливать спидометр как обязательный измерительный прибор автомобиля. Первым изобретателем спидометра считается сербский ученый Никола Тесла.

1.2 Способы измерения скорости

На сегодняшний день, с развитием науки и технологий, методы и способы измерения скорости постоянно меняются и совершенствуются. Стандартный автомобильный спидометр основан на счете количества оборота вторичного вала за определенное время.

Классификация устройств по измерению скорости [1]:

1) Аэрометрический. Основан на измерении динамического напора, функционально связанного со скоростью тела, движущегося в воздушной среде. Средства измерения, построенные на аэрометрическом методе, позволяют измерять скорость с погрешностью, не превышающей 2 – 3%

2) Акустический. Основан этот способ измерения скорости на приеме электроакустическим преобразователем звука от движущихся пар. При этом способе, выделяют так же тональную составляющую из спектра шумоизлучения автомобиля. Этот способ может применяться в условиях плохой видимости, но не является в достаточной степени надежным и не позволяет произвести идентификацию транспортного средства, поэтому такой способ не получил распространения.

3) Корреляционный. Предназначен в основном для контроля скорости движущихся транспортных средств. За перемещаемым объектом следят два фотоприемника, размещенные на расстоянии друг от друга. Они воспринимают отраженный от движущегося объекта свет. Сигналы фотоприемников усиливаются, фильтруются и преобразуются в цифровой вид

после чего поступают в процессор, который выполняет функции коррелятора. При появлении нерегулярностей соответствующие пики выходного сигнала второго фотоприемника оказываются сдвинутыми, что сигнализирует об изменении и на сколько именно.

4) Использование спутниковой системы навигации. Он основан на измерении задержки распространения сигнала от спутника до приемника. Из полученного сигнала приемник получает данные о местонахождении спутника. Подробнее насмотрен в подразделе 1.4.

5) Оптический. Основан на использовании лазера (или лидара). Лазер излучает на уровне инфракрасного диапазона несколько коротких импульсов в сторону измеряемого объекта. Далее это излучение дойдя до объекта отражается от него и попадает на лидар. Устройство управление анализирует дальность до объекта в разные фиксированные моменты времени. Используя константное время излучения инфракрасных импульсов, и динамически изменяющееся расстояние из двух точек в двух моментах времени, устройство управление с легкостью вычисляет скорость объекта.

6) Радиочастотный способ, основанный на использовании эффекта Доплера, имеющий в своем составе радар, включающий в себя источник ультразвуковых колебаний с частотой f_0, и приемник ультразвуковых колебаний, отразившихся от движущегося объекта. Если объект приближается к радару, то частота колебаний f, отразившихся от него, будет больше первоначальной. Если наоборот, то частота будет меньше. Таким образом, по разности частот ($f – f_0$) можно узнать о скорости объекта и о направлении его движения. Есть следующие диапазоны частот: X (10,5 ГГц), K (24,150 ГГц).

1.3 Обоснование выбора метода измерения

Причина, по которой была выбрана методика с использованием навигационных спутников, связана именно с проблематикой обычных автомобильных спидометров – их точность.

Производители транспортных средств намерено завышают показания спидометров в среднем на 5%. Правило ЕЭК ООН № 39 [2], регламентирующее производство измерительных приборов для автомобилей, устанавливает норму, что исправный спидометр автомобиля всегда должен или завышать измеряемую скорость, (на величину не более 10%), или показывать ее абсолютно точно.

Измерение же посредством приема спутникового сигнала всегда показывает скорость движения гораздо более близкую к реальной, чем любой штатный автомобильный спидометр. Некоторое влияние на точность показаний может оказывать положение спутников, но для средних широт эта проблема не так актуальна. Плохая точность такого способа измерения неэффективна в полярных широтах, ввиду удаленности орбит спутников от этих областей. Некоторое влияние на точность измерений скорости могут также оказать рельеф местности, промышленные низкочастотные помехи,

архитектурные сооружения – из-за этих факторов сужается видимый горизонт, но это кратковременные явления.

1.4 Навигационной системы

В современный век развития, спутниковая навигация развивается быстрыми шагами. В настоящее время в мире существуют четыре спутниковые системы навигации: GPS, Galileo, Бэйдоу и ГЛОНАСС.

Вообще, началом спутниковой навигации считается первый полет искусственного спутника Земли (ИСЗ) в 1957 году [3]. Американские ученые во главе с Ричардом Кешнером наблюдали эффект Доплера на испускаемом сигнале от спутника. Возникла идея от обратного: траектория спутника заранее известна, но неизвестен месторасположение приемника – точно зная положение спутника, можно определить собственную скорость и координаты.

Навигационная Система Глобального Позиционирования *Global Positioning System* (GPS) является частью комплекса навигационной системы определения времени и дальности (*Navigation Satellites providing Time And Range* – NAVSTAR), в США [4]. Разработка комплекса навигационной системы была начата 1973 году. Гражданский сегмент спутниковой сети NAVSTAR принято называть аббревиатурой GPS, коммерческая эксплуатация которой началась в 1995 году. Основой системы GPS являются 24 навигационных спутника, движущиеся вокруг Земли по 6 круговым орбитальным траекториям, на высоте 20 тысяч киллометров. Наземная часть системы GPS состоит из десяти станций слежения.

ГЛОбальная Навигационная Спутниковая Система (ГЛОНАСС) была разработана в СССР и запущена в 1982 году. Изначально был также, как и в США, реализован под военные нужды, в первую очередь в навигации на море и позиционирование в пространстве в воздухе. Позже был предоставлен и в гражданский сегмент. Основой системы являются также, как и в GPS 24 спутника, движущихся над поверхностью Земли, но в трех орбитальных плоскостях и на высоте около 19 тысяч километров. Основное отличие от системы GPS в том, что спутники ГЛОНАСС не имеют резонанса (синхронности) с вращением Земли – это обеспечивает большую стабильность, которая в свою очередь не требует постоянных корректировок орбит спутников во время эксплуатации.

1.5 Принцип работы системы GPS

Основу системы составляют сеть ИСЗ, расположенных на околоземной орбите. Орбита каждого спутника точно рассчитана, для того чтобы в любой момент времени можно было знать месторасположение спутника. И спутник, чтобы быстро воспринимал где он находится. Эти спутники непрерывно излучают сигнал в направлении Земли.

Координаты местоположения GPS-приемника вычисляются по

расстоянию до ИСЗ. Для вычисления расстояния используется свойство распространения радиосигнала – электромагнитные волны распространяются со скоростью света. Как упоминалось ранее в разделе 1.4, спутники излучают радиосигнал. Приемник принимает их с учетом известной времени отправки и времени приема. Это называется *беззапросная* радионавигация. Так узнается расстояние от приемника до спутника. Однако для позиционирования этого недостаточно. Необходима также информация от другого спутника. Зная орбиту и получая координаты обоих спутников, полученные путем измерения сдвига радиочастот (эффект Доплера – что дает еще и возможность получить скорость объекта), в точке пересечение окружностей с радиусом равных расстояниям до обоих спутников мы имеем место нахождения приемника. Третий спутник необходим для определения на какой высоте находится приемник относительно уровня моря.

Однако, при данном методе радионавигации точное определение времени распространения сигнала возможно лишь при наличии синхронизированном времени приемника и ИСЗ. Размещение точных часов (атомных) в приемнике заведомо дорогой вариант. Поэтому в состав аппаратуры ИСЗ входят эталонные часы, причем точность спутникового эталона времени исключительно высокая. [6]

На измерениях всегда присутствует ошибка, обусловленная несовпадением времени у спутника и приемника. По этой причине в приемнике вычисляется искаженное значение дальности до спутника (псевдодальность). Поэтому, необходимо вычислять еще и необходимую поправку к часам приемника. Для их определения необходимо выполнить измерения еще на четвертом спутнике. В результате обработки этих измерений в приемнике вычисляются расстояния до трех спутников, что соответствует позиционированию по трем плоскостям, и точное время.

Спутник излучает синусоидальные сигналы на двух несущих частотах: L1 = 1575,42 МГц и L2 = 1227,60 МГц. Прежде, чем передать их на Землю, сигнал кодируется псевдослучайной цифровой последовательностью – происходит фазовая модуляция. Это необходимо чтобы ограничить доступ к GPS, и одновременно повысить надежность сообщений от шумов, так как псевдослучайные коды самые стойкие к таким помехам.

Сигнал L1 имеет два дальномерных кода с псевдослучайным шумом (PRN), P-код и C/A код (Рисунок 1.1). P-код зашифрован для военных целей. C/A код не зашифрован. Сигнал L2 модулируется только с P-кодом. Большинство гражданских пользователей используют C/A код при работе с GPS системами. Обе несущие частоты дополнительно кодируются навигационным сообщением, в котором содержатся данные об орбитах спутника, информация о параметрах атмосферы, поправки системного времени [6]. Приемник проверяет входящий сигнал со спутника и определяет, когда он генерировал такой же код. Далее происходит долгий, для аппаратной части анализ.

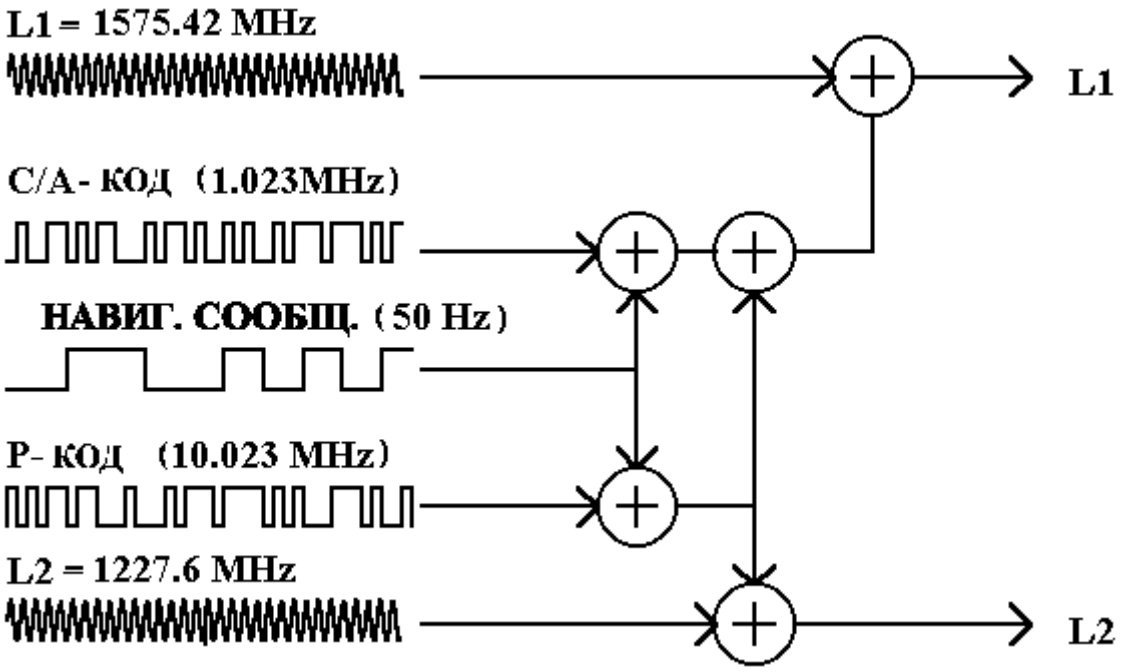

Рисунок 1.1 – Формирование спутникового сигнала [6]

После получения всех необходимых данных формируется пакет. Полученная разница, вместе с синхронизацией по четвертому спутнику и корректировке времени, дает и синхронизацией по четвертому спутнику и корректировке времени, дает искомое расстояние.

1.6 Описание протокола NMEA0183

В разрабатываемом многофункциональном GPS-спидометре GPS-приемник выдает сигнал на микроконтроллер в формате NMEA0183 [7]. NMEA – это формат передачи сообщений между корабельными приборами. Он включает в себя систему сообщений для обмена информацией между навигационными GPS-приемниками и потребителями навигационной информации. Все команды и сообщения передаются в ASCII коде. Первый байт сообщения, для идентефикации типа навигационной системы, указывается $GP, что означает что приемник провзаимодействовал с GPS и передает пакет. В последнем поле сообщения может быть указана контрольная сумма текущего сообщения d CRC-коде, разделеная от основного сообщения символом «*». Контрольная сумма всех символов сообщения, включая пробелы, расположенных между разделителями «$» и «*», не включая последних.

1.6.1 GPGGA

Сообщение содержит данные о местоположении, вре-мени определения местоположения, качестве данных, количестве использо-ванных спутников,

фактор ухудшения точности координат, информацию о дифференциальных поправках и их возраст. Пример сообщения: *$GPGGA*, 004241.47, 5532.8492, *N*, 03729.0987, *E*, 1, 04, 2.0, –0015,*M*,,,,*31, расшифров-ка представлена в таблице 1.1

Таблица 1.1 – Содержание сообщения *$GPGGA*

Тип сообщения	1	2	3	4	5	6	7
$GPGGA	*hhmmss.ss*	1111.11	a	*yyyyy.yy*	*a*	*x*	*xx*
8	9	10	11	12	13	14	15
x.x	*xxx*	*M*	*x.x*	*M*	*x.x*	*xxxx*	**hh*

где 1– гринвичское время на момент определения местоположения;

2 – географическая широта местоположения;

3 – север/юг (N/S);

4 – географическая долгота местоположения запад/восток (E/W);

5 – индикатор качества GPS-сигнала: 0 – Определение местоположения невозможно или не верно; 1 – GPS-режим обычной точности, возможно определение местоположения; 2 – Дифференциальный GPS-режим, точность обычная, возможно определение местоположения;

6 – количество используемых спутников

7 – фактор ухудшения точности плановых координат (HDOP);

8 – высота антенны приемника над уровнем моря;

9 – единица измерения высоты расположения антенны, метры;

10 – геоидальное различие – различие между земным эллипсоидом WGS 84 и уровнем моря – уровень моря ниже эллипсоида;

11 – единица измерения различия, метры; 12 – возраст дифференциальных данных GPS – время в секундах с момента последнего обновления SC104 типа 1 или 9.

1.6.2 GPGLL

Сообщение содержит данные о географической широте, долготе и времени определения координат. Пример сообщения: $GPGLL, 5532.8492, N, 03729.0987, E, 004241.469, A*33. Расшифровка содержания приведена в таблице 1.2.

Таблица 1.2 – Содержания сообщения *$GPGLL*

Тип сообщения	1	2	3	4	5	6	7
$GPGLL	1111.11	*a*	*yyyyy.yy*	*a*	*hhmmss.ss*	*A*	**hh*

где 1 – географическая широта местоположения;
2 – север/юг (*N/S*);
3 – географическая долгота местоположения;
4 – запад/Восток (*E/W*);
5 – гринвичское время на момент определения местоположения;
6 – статус: *A* – данные верны, *V* – данные не верны;
7 – контрольная сумма строки.

1.6.3 GPVTG

Сообщение содержит информацию о направлении и скорости движения. Пример сообщения*: $GPVTG*, 360.0, *T*, 348.7, *M*,000.0, *N*, 000.0, *K**43 Расшифровка сообщения приведена в таблице 1.7

Таблица 1.3 – Содержание сообщения *$GPVTG.*

Тип сообщения	1	2	3	4	5	6	7	8	9
$GPVTG	*x.x*	*T*	*x.x*	*M*	*s.ss*	*N*	*s.ss*	*K*	**hh*

где 1 – направление движения в градусах;
2 – относительно Северного полюса;
3 – направление движения в градусах (может не использоваться);
4 – относительно северного магнитного полюса (может не использоваться);
5 – скорость;
6 – единица измерения скорости, узлы;
7 – скорость;
8 – единица измерения скорости, км/ч;
9 – контрольная сумма строки.

Проведя анализ раздела обзора литературы, были получены выводы свидетельствующие, о полном изучении преимуществ, свойств и специфики измерения скорости посредством технологии GPS. Было приведено сравнение достоинств и недостатков иных способов измерений.

1.7 Обзор архитектуры AVR

Контроллер является ядром системы, именно он управляет драйверами устройств индикации, получает сообщения от GPS модуля и обрабатывает их, а также исполняет основной алгоритм программы, записанный в его флеш-память.

Ввиду того, что необходимо использовать дешевую, малогабаритную, и в тоже время крайне распространенную архитектуру, выбор был сделан в пользу семейства AVR.

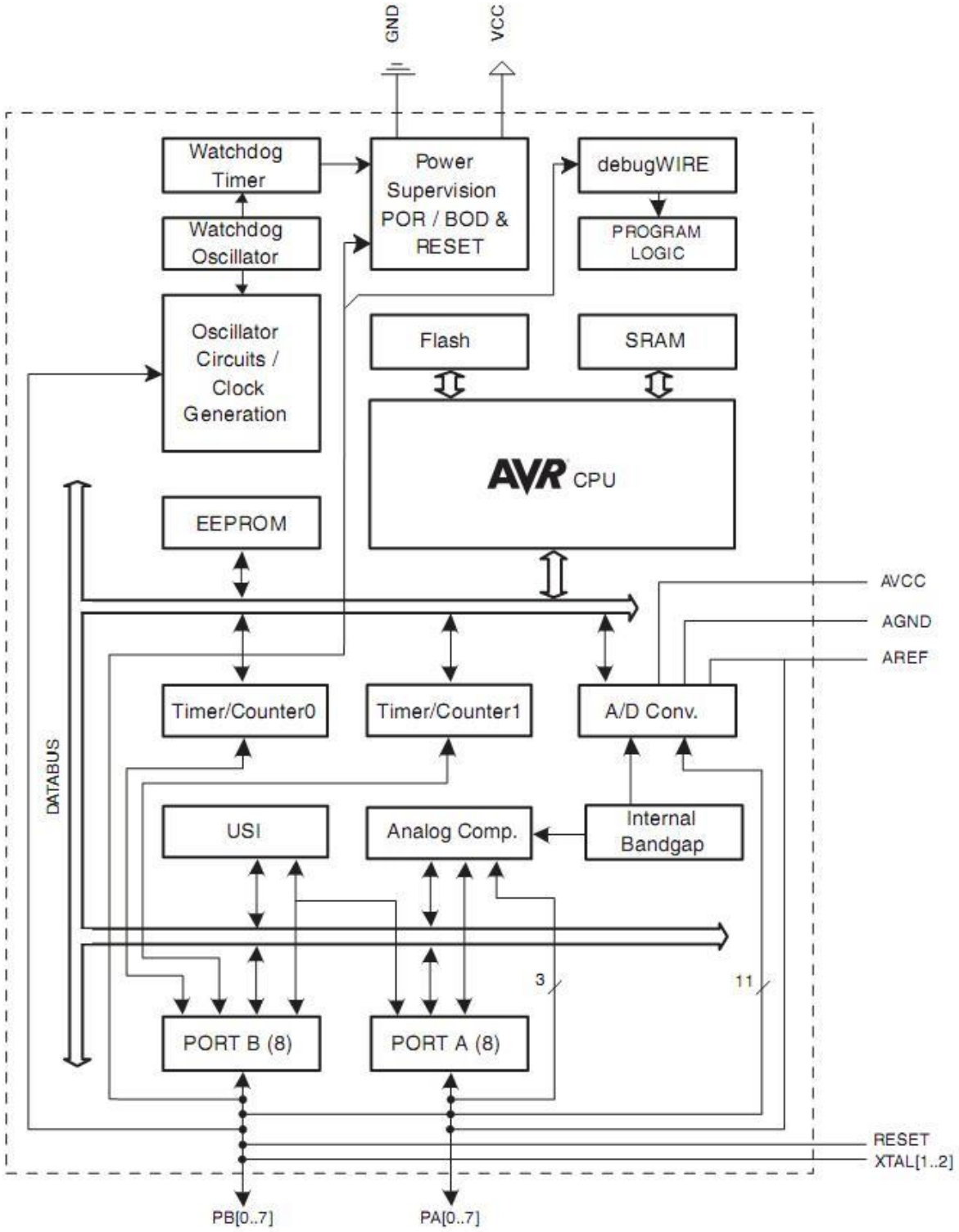

Рисунок 1.2 – Архитектура ATtiny865 [7]

Наибольшим преимуществом семейства AVR – это наличие 32 оперативных регистров. Несмотря на их разные в правах доступа, это позволяет не обращаться в оперативную память и стек. На этом фоне семейство PIC крайне малопригодны для конструирования и проектирования – они скорее больше «заточены» под несложные устройства с высоким

тиражированием. Контролеры от производителей ST и Texas Instruments будут слишком дорогими и избыточными [6]. Также имеет место простота использования в программировании данной системы, хотя больших отличий с более развитых систем компаний названых выше мало.

Рассмотрим архитектуру и принцип работы на примере микроконтроллера ATtiny865. Это 8-разрядный микроконтроллер на базе RISC архитектуры из семейства AVR с низким энергопотреблением, объединяющий 16 КБ программируемой флэш-памяти, 1 КБ SRAM, 512B EEPROM, 8-канальный 10-разрядный аналого-цифровой преобразователь и интерфейс JTAG для включения отладки. Устройство поддерживает пропускную способность 16 MIPS на 16 МГц и работает от 4,5 до 5,5 В. Выполняя инструкции за один такт, устройство достигает пропускной способности, приближающейся к 1 MIPS на МГц, балансируя энергопотребление и скорость обработки. Ядро AVR объединяет богатый набор команд с 32 рабочими регистрами общего назначения. Все 32 регистра напрямую подключены к арифметическому логическому блоку (АЛУ), что позволяет получить доступ к двум независимым регистрам в одной инструкции, выполняемой за один такт (Рисунок 1.2). Получающаяся архитектура более эффективна в отношении кода, обеспечивая при этом пропускную способность до десяти раз быстрее, чем у обычных CISC-микроконтроллеров. Обладает следующими характеристиками:

1) Усовершенствованная архитектура RISC - 2-тактный множитель на кристалле.
2) Высокопрочные, энергонезависимые сегменты памяти.
3) Интерфейс JTAG (совместимый с IEEE 1149.1).
4) Два 8-битных таймера / счетчика с отдельными пределителями и режимами сравнения.
5) Один 16-битный таймер / счетчик с отдельным пределителем, режимом сравнения и режимом захвата.
6) Сброс при включении питания (POR) и программируемое обнаружение отключения.
7) Внутренний калиброванный RC генератор.
8) Внешние и внутренние источники прерываний.

1.8 Интерфейс I_2C

Интерфейс I_2C был придуман и стандартизирован фирмой Philips. В терминологии Atmel, из-за патентной чистоты, этот интерфейс называется, как *Двухпроводной последовательный интерфейс* (Two-Wire Serial Interface, TWI) [8]. Это двухпроводная шина, состоящая из двух проводов для последовательной передачи данных *serial data* (SDA), и синхросигнала *serial clock* (SCL). Распределение ролей в данной системе классическое: *Ведущий* (Master) и *Ведомый* (Slave).

Master инициирует передачу данных по шине, Slave только отвечает на

запросы мастера и только по его вызову. Шина может работать с несколькими главными устройствами (Multi-Master), когда в каждый любой момент времени на шине активно только одно Master устройство. Однако чаще всего используют простую конфигурацию шины только с одним Master устройством, все остальные устройства работают как Slave (подчиненных устройств может быть одно или несколько).

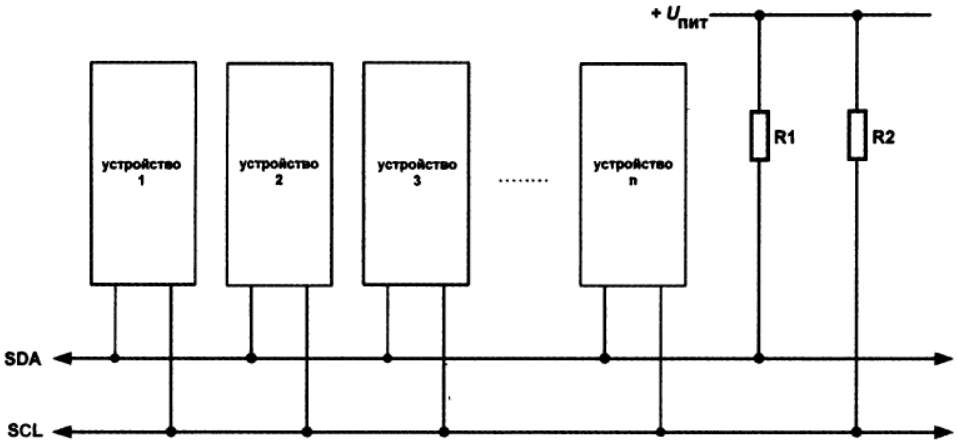

Рисунок 1.3 – Соединение устройств по интерфейсу I_2C [9]

Все устройства на шине соединены друг с другом параллельно (Рисунок 1.3), используя драйверы с открытым стоком для формирования на шине цифровых сигналов. По этой причине на шине необходимо присутствие два верхних подтягивающих резистора. Они должны быть достаточно большие, чтобы не перегружать выходные стоки драйверов устройств, которые подключены к шине. Чем выше частота передачи, тем меньше номинал нагрузочных резисторов.

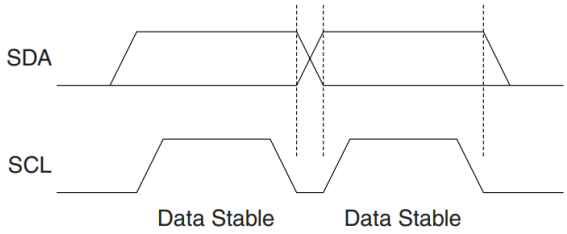

Рисунок 1.4 – Валидность данных [9]

Данные передаются по принципу MSB (Старшим разряд первый поступает на линию SDA). Каждая транзакция байта сопровождается START и STOP сигналами на линиях. Данные на линии SDA валидны в момент, когда на линии SCL высокий логический уровень (Рисунок 1.4).

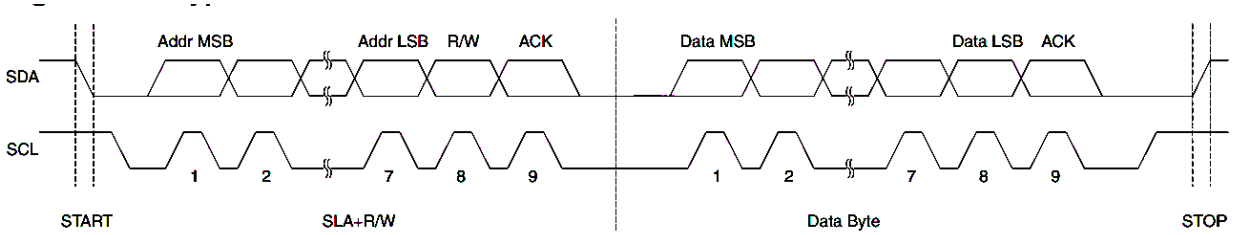

Рисунок 1.5 – Диаграмма передачи данных на ведомое устройство [9]

Для того, чтобы отличать подчиненные устройства на шине, используется 7-битный адрес (так изначально определила компания Philips), передаваемый как первый байт после сигнала START (Рисунок 1.5). Биты адреса передаются в старших семи битах байта. Младший бит LSB байта адреса имеет бит чтения/записи. В конце передачи адреса Мастер ждет подтверждение от ведомого. После его получения мастер начинает передавать байт данных, и последним битом, после квитирования отправленных данных, передается сигнал STOP.

1.9 Обзор аналогов

GPS-спидометр RL-SM001. Его внешний вид представлен на рисунке 1.6. Этот GPS-спидометр является компактным прибором, с минимальным количеством функций. Он способен отображать скорость, время, температуру. Его исполнение (цилиндр круглого сечения) подразумевает монтаж в

Рисунок 1.6 – GPS-спидометр RL-SM001
Источник изображения: *https://driveboat.by/katalog/rl-sm001.html*

приборную панель. Он имеет стандарт защиты IP65 и подсветку дисплея зеленого цвета. Но, однако это не позволяет использовать его портативно и

быстро на разных транспортных средствах и объектах, что дает ему большой минус.

Рисунок 1.7 – GPS-спидометр C60 Hud
Источник изображения*: https://www.pinterest.es/pin/776589529476809577/*

Для пользователя доступно меню прибора, где он может задать единицы измерения. Преимущества такого спидометра в том, что он компактен и имеет пылевлагозащитную конструкцию корпуса, что позволяет его устанавливать на катамараны, лодки и т.п. Из недостатков – малый набор измеряемых величин, принимающая сигнал антенна спрятана в корпусе, что снижает чувствительность.

Еще одним рассматриваемым аналогом является GPS-спидометр C60 Hud. Его внешний вид представлен на рисунке 1.7. Данный прибор имеет предназначение для измерения текущей скорости, высоту относительно уровня мирового океана, скорость в двух мерах (км/ч и м/ч)., а также одометр. Большим минусом является отсутствие выносной антенны – она встроена в корпус как у и предыдущего аналога. А также время на поиск и анализ спутников.

Сравнив представленные аналоги, можно выделить, что в разрабатываемый в дипломном проектировании прибор, должен обладать большим количеством преимуществ, которыми являются:

– наглядное представление информации;

– наличие звукового оповещения;

– наличие выносной антенны, что позволяет обеспечить лучший прием для антенны GPS-приемника.

2 РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ

После изучения теоретических основ и формирования полного представления о том, что должна делать система, можно перейти к проектированию на уровне блоков и соответственно, к построению структурной схемы. Схема структурная электрическая раскрывает структуру системы управления комплексом по измерению скорости объекта с точки зрения крупноблочного проектирования и приведена на чертеже ГУИР.400201.034 Э1.

Комплекс измерения скорости объекта включает в себя следующие модули:

1) Микроконтроллер.
2) Модуль GPS.
3) Источник питания.
4) Каскад согласования измеряемого напряжения.
5) Датчик освещенности.
6) Преобразователь напряжения.
7) Модуль индикации.
8) Часы реального времени.

Рассмотрим каждый из перечисленных модулей подробнее.

2.1 Микроконтроллер

Микроконтроллер является ядром системы, именно он управляет всеми остальными модулями и осуществляет управление системой в целом, путем исполнения прописанной в него программы. Он связан практически со всеми системами односторонней связью (кроме часов реального времени). Любые изменения на одном из блоков немедленно поступают на микроконтроллер, который непрерывно все это обрабатывает. И вносит изменения на блок индикации.

Модуль GPS соединен через последовательный USART интерфейс. После получения пакетов с информацией о местоположении, контролер, приняв эту информацию начинает обрабатывать и вносить изменения на блоке индикации. Также, необходимо предусмотреть возможность немедленно перепрограммировать контролер, без извлечения его из устройства.

Часы реального времени необходимы для реализации часов в данном устройстве. Для большей точности и внесения корректировок в их ход этот модуль соединен с микроконтроллером двухсторонней связью. Для внесения изменений от полученных данных времени по спутнику, а затем отображение их на блок индикации.

Также контролер получает через определенный промежуток времени данные с каскада согласования преобразователя напряжения.

2.2 Модуль GPS

Модуль GPS, используемый в составе прибора, служит приема сигнала от спутника GPS. Он принимает пакеты данных от спутников, и после анализа формирует пакеты, которые далее будут отправлены микроконтроллеру, посредством интерфейса UART. В модуле необходимо использовать активную антенну, для того чтобы позволить повысить чувствительность приемника к приему сигнала. Модуль выдает результаты в формате NMEA.

2.3 Источник питания

Источником питания служит аккумуляторная батарея автомобиля на 11В – 16В. Она подключена напрямую к преобразователю напряжения и каскаду согласования измеряемого напряжения.

2.4 Каскад согласования измеряемого напряжения

Каскад согласования измеряемого напряжения состоит из стабилизирующей и понижающей аппаратуры. Необходим для измерения текущего напряжения на источнике питания автомобиля через аналого-цифровой преобразователь микроконтроллера.

2.5 Датчик освещенности

Датчиком освещенности необходим для корректировки яркости на блоке индикации. Этот модуль состоит из фоторезистора, подключенного к системе компараторов, сравнивающих текущий свет с эталонным значением, которое подтянуто от преобразователя напряжения. Полученный сигнал постоянно получает микроконтроллер и периодически опрашивая свои порты ввода/вывода, реагирует на изменения состояния этого модуля.

2.6 Модуль индикации

Модуль индикации состоит из звукового устройства, матричного дисплея и трех жидкокристаллических OLED дисплеев. Электромагнитный излучатель звука используется для звуковой сигнализации каких-либо запланированных событий. OLED дисплей предназначен в устройстве для вывода текстовой информации. Подключено через последовательный интерфейс. Также есть отдельный матричный дисплей. На котором будет отображаться текущая скорость. Звуковое устройство будет использоваться как звуковой сигнализатор, дабы уменьшить воздействие глаз водителя и постоянно не отвлекать его.

2.7 Преобразователь напряжения

Преобразователь напряжения необходим для создания 5 вольт, которые используются для питания микроконтроллера, часов реального времени, модуля индикации и GPS-модуля. Совокупность этих частей является значительным потребителем энергии. Поэтому, применение линейных стабилизаторов не желательно, так как на них будет выделяться большое количество тепла, что внесет большую проблему при эксплуатации в автомобиле. Исходя из этого, необходимо установить импульсный понижающий преобразователь с большим значением КПД. Внутри он состоит из делителя напряжения с возможностью точной коррекции необходим для того, чтобы согласовать уровень значения напряжения питания с уровнем, обрабатываемым АЦП микроконтроллера. Это необходимо для корректного измерения входного уровня напряжения питания. Делитель задает соотношение 1:10. В нем предусмотрен подстрочный многооборотный резистор, сопротивлением которого выстраивается точность измерений, которая нарушается из-за разброса номиналов деталей.

2.8 Часы реального времени

Часы реального времени с последовательным интерфейсом – это малопотребляющие полные двоично-десятичные часы-календарь, включающие 56 байтов энергонезависимой статической ОЗУ. Адреса и данные передаются последовательно по двухпроводной двунаправленной шине. Часы-календарь отсчитывают секунды, минуты, часы, день, дату, месяц и год. Последняя дата месяца автоматически корректируется для месяцев с количеством дней меньше 31, включая коррекцию високосного года. Часы работают как в 24-часовом, так и в 12-часовом режимах с индикатором AM/PM. Часы реального времени имеют встроенную схему наблюдения за питанием, которая обнаруживает перебои питания и автоматически переключается на питание от батареи. Именно низкое потребление (порядка 500 нА) от батареи, при отключении внешнего питания наиболее лучше характеризует эти часы и их применение в схеме. Так как преимущественно большее время они будут отключены от питания (когда автомобиль не эксплуатируется). Для автономной работы этих часов, на плате предусмотрена посадочная колодка под элемент питания.

3 РАЗРАБОТКА ФУНКЦИОНАЛЬНОЙ СХЕМЫ

В данном разделе будет описана более подробно, с функциональной точки зрения, ранее спроектированная в предыдущем разделе структурная схема аппаратно-программного комплекса измерения скорости.

Схема функциональная электрическая раскрывает функционал системы управления комплексом по измерению скорости объекта с точки зрения алгоритмов и систем взаимодействия и приведена на чертеже ГУИР.400201.034 Э2.

Комплекс измерения скорости объекта включает в себя следующие модули:

1) Микроконтроллер.
2) Модуль GPS.
3) Источник питания.
4) Каскад согласования измеряемого напряжения.
5) Датчик освещенности.
6) Преобразователь напряжения.
7) Модуль индикации.
8) Часы реального времени.

3.1 Микроконтроллер

Микроконтроллер является устройством, которое принимает, обрабатывает и отправляет сигналы от и на внутренние блоки и модули. В результате осуществляется управление и функционирование всей системы.

Микроконтроллер должен содержать в себе следующие функции:

– сбор цифровых данных с периферийных модулей;

– сбор аналоговых данных с преобразователей напряжения и периферийных аналоговых датчиков;

– анализ принятых данный (аналоговых и цифровых), их вычисления и преобразования;

– формирование сигнала для отправки вычисленных сигналов на периферийные устройства в виде полезной информации;

– передача данных по интерфейсу с использованием внутренних средств обработки с данными интерфейсами;

– передача команд исполнительным устройствам;

– тактирование периферийных устройств.

Изображенная блок схема микроконтроллера на рисунке 3.1 включает в себя следующие подсистемы:

– подсистема ввода/вывода *General Purpose Input/Output* (*GPIO*);

– вычислительное ядро (*core*);

– аналогово-цифровой преобразователь (*Analog Digital Converter ADC*);

– подсистему устройства последовательной асинхронной передачи и

приема данных (universal asynchronous serial receiver and transmitter – далее *UART*);

– двухпроводной последовательный интерфейс *I2C*;

– последовательный периферийный интерфейс (serial peripheral interface – далее SPI);

– запоминающие устройства энергозависимые (*SRAM*), энергонезависимые (*EEPROM*) и перепрограммируемые (*FLASH*);

– подсистема прерываний (Interrupt unit – далее *INT*);

– подсистема тактирования (oscillator – далее *OSC*).

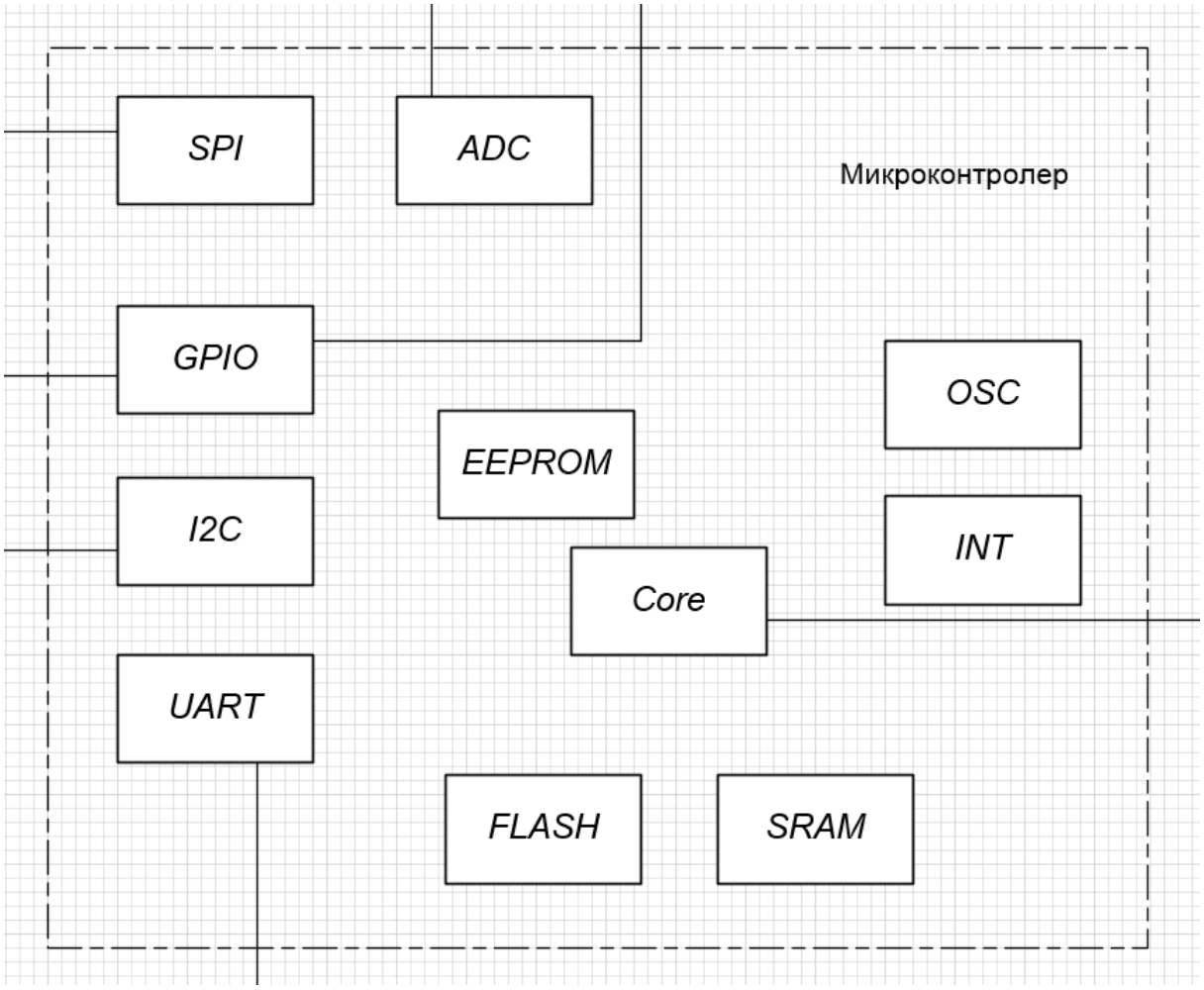

Рисунок 3.1 – Блок-схема микроконтроллера

Ниже будет более подробнее рассмотрена применение и назначение каждой указанной подсистемы в данном проекте.

3.1.1 Подсистема ввода/вывода GPIO

Все порты имеют настоящую функциональность Read-Modify-Write при использовании в качестве общего цифрового порта ввода/вывода. Это

означает, что направление одного вывода порта может быть изменено без непреднамеренного изменения направления любого другого вывода с помощью инструкций SBI и CBI.

То же самое относится к включению или выключению подтягивающих резисторов (которые необходимы для работы в режиме приема). Каждый выходной буфер имеет симметричный диск характеристик как с высокой нагрузкой, так и с возможностями управлять светодиодными дисплеями напрямую. Все выводы порта должны имеют индивидуально выбираемые подтягивающие резисторы с инвариантным сопротивлением напряжения питания. Все выводы ввода/вывода должны иметь защитные диоды к обоим выводам питания Vcc и заземление, как показано на рисунке 3.2.

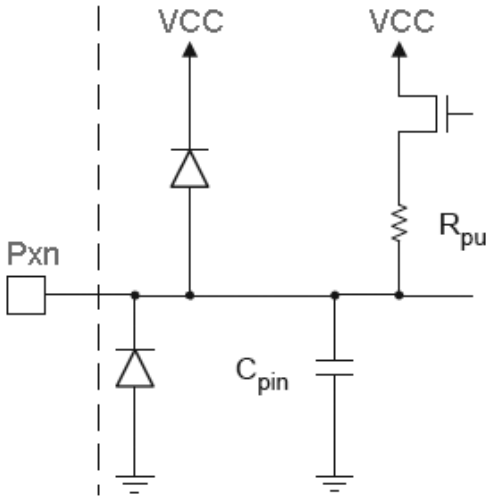

Рисунок 3.2 – Схема портов GPIO

Все регистры и битовые ссылки в этом разделе записаны в общем виде. Строчные буквы «X» обозначает букву нумерации порта, а строчная буква «n» обозначает бит число. Однако при использовании регистра или бита в программе указывается точная форма необходимо использовать. PORTB3 для бита 3 в порте B, документально, как правило, PORTxn.

Три адреса памяти ввода-вывода выделены для каждого порта:

– регистр данных – PORTx;
– регистр направления данных DDRx;
– входные контакты порта PINx.

Расположение ввода/вывода входных контактов порта доступно только для чтения, в то время как регистр данных и направление данных регистр для чтения/записи.

Принимая и передавая данные и сигналы с подсистемы ввода/вывода, мы можем получать их для анализа, и для проведения необходимых вычислений в ядре, а также для управления иными периферийными устройствами.

Также, в подсистеме GPIO существует возможность подключать альтернативные функции. Это необходимо для использования иных внутренних периферийных блоков напрямую, в обход стандартных подсистем приема и буфера ввода/вывода.

3.1.2 Вычислительное ядро

Вычислительное ядро представляет из себя главный компонент микроконтроллера. именно он занимается вычислительной мощностью всего комплекса. Преимущественно состоит из арифметико-логического устройства (АЛУ), общих регистров назначения (РОН), регистров статуса (SR), регистра инструкций, счетчика команд (program counter), стека и декодера команд.

Программный счетчик, устанавливается в ноль, тем самым указывая на начальный адрес инструкций, записанных в память FLASH, находящуюся вне ядра. Прочитав оттуда первую инструкцию, она сохраняется в регистре инструкций и декодируется в декодере команд. Когда устройство управление распознало команду, начинает ее выполнять, задействую регистры общего назначения и АЛУ. результат вычислений и манипуляций выносится на системную шину, расположенную вне ядра. Далее эта информация попадает на внутреннею периферию или в оперативное запоминающее устройство (ОЗУ).

3.1.3 Аналогово-цифровой преобразователь

Аналого-цифровой преобразование — это процесс преобразования входной физической величины в ее числовое представление. Аналого-цифровой преобразователь или АЦП (от англ. *analog to digital converter*, сокращенно *ADC*) – это устройство, выполняющее такое преобразование. Формально, входной величиной АЦП может быть любая физическая величина – напряжение, ток, сопротивление, емкость, частота следования импульсов, угол поворота вала и т.п. Однако, для определенности, в дальнейшем под АЦП мы будем понимать исключительно преобразователи напряжение-код.

АЦП имеет множество характеристик, из которых основными можно назвать частоту преобразования и разрядность. Частота преобразования обычно выражается в отсчетах в секунду (*samples per second*, SPS), разрядность – в битах. Современные АЦП могут иметь разрядность до 24 бит и скорость преобразования до единиц GSPS (конечно, не одновременно). Чем выше скорость и разрядность, тем труднее получить требуемые характеристики, тем дороже и сложнее преобразователь. Скорость преобразования и разрядность связаны друг с другом определенным образом, и мы можем повысить эффективную разрядность преобразования, пожертвовав скоростью.

На ADC микроконтроллера поступает опорный сигнал, равный Vcc. С каскада согласования измеряемого напряжения поступает сигнал равный

некой неизвестной величине, меньшей чем Vcc. Далее они сравниваются аналоговым компаратором и компаратором и получившиеся кривая раскладывается на бинарные значения специальным кодером. Величина полученного значения будет зависеть от разрядности этого конвертера.

Далее полученное значение будет программно изменено в реальное путем пропорционального соотношения и выведено на знакосинтезирующий экран.

3.1.4 UART

Последовательным асинхронным приемопередачиком является интерфейс, который использует одну линию. подключение с периферийными устройства по этому интерфейсу осуществляется скрещено.

Основные рабочие линии – RXD и TXD, или просто RX и TX. Передающая линия – TXD (*Transmitted Data*), а порт RXD (*Received Data*) – принимающая.

Данная подсистема обычно разделяют три основные части:

- тактовый генератор;
- передатчик;
- приемник.

Передатчик состоит из одного буфера записи, регистра последовательного сдвига, генератора четности и логики управления для обработки различных форматов последовательных кадров. Буфер записи позволяет осуществлять непрерывную передачу данных без задержки между кадрами. Приемник является наиболее сложной частью модуля UART благодаря своим модулям синхронизации и восстановления данных. Блоки восстановления используются для асинхронного приема данных. В дополнение к блокам восстановления, приемник включает проверку четности, логику управления, регистр сдвига и двухуровневый буфер приема.

В данном проекте будет использоваться 1 стоп бит в кадре, а размер информации 8 бит.

В данном комплексе будет использоваться только прием информации, поэтому будет задействован только входной буфет приема данных. Данные последовательно поступают на сдвиговый регистр с TXD линии, после которого при появлении стоп бита защелкивается и загружается в регистр приема. После этого в регистре статуса UARTустанавливается флаг, который говорит от том что данные получены. После прочтения появится флаг что буфер пуст, и следующее принятое сообщение будет загружено в приемный буфер. Но так как скорость этого интерфейса в разы меньше скорости обработки внутри контролера, большую часть времени контролер будет простаивать на приеме. Буте выгоднее по затратам времени пользоваться прерываниями этого подсистемы, которые срабатывают по флагу заполняемости буфера.

Также необходимо определить скорость работы данной подсистемы. Вычисление коэффициента, который позже будет использован при настройке контролера, находится по формуле (3.1):

$$UBBB = \frac{f_{osc}}{(16 \cdot BAUD)} - 1 \qquad (3.1)$$

где UBBB – коэффициент, используемый для конфигурации подсистемы UART;

f_{osc} – частота тактирования контролера, которая равна, МГц;

BAUD – скорость с которой модуль GPS передает контролеру, бод/с.

Воспользовавшись формулой (3.1), и зная, что частота тактирования контролера f_{osc} = 8 МГц, а скорость приема данных должна быть BAUD = 9600 бод/с, следовательно, коэффициент UBBB = 51.

3.1.10 Выбор архитектуры

После рассмотренных всех необходимых обязательных внутренних подсистем и внутренних модулей необходимо выбрать какое семейство использовать микроконтроллеров из микроархитектур гарвардского типа RISC. Кроме всех вышеуказанных требований, также необходимое требование — это размер памяти. Ввиду того, что в таких микроархитектурах адресное пространство разделено (код и данные в разных областях памяти) необходимо выбрать именно так, чтобы памяти хватало на выполнение поставленных целей, и в тоже время, контролер не должен быть перенасыщен внутренними периферийными устройствами. По распространенности и простоте программирования лучше всех подойдет семейство AVR, которое был описано в разделе 1.7.

3.2 Модуль GPS

Модуль GPS представляет из себя устройство, состоящее из антенны приемника, устройства управления, обработчика сигналов и контролера передачи данных. Оно осуществляет прием и анализ данных, полученных от приемников. Также, именно этот модуль осуществляет подключение и производит аналитическо-вычислительные процессы, которые в дальнейшем формируются в пакет данных, поступающих на микроконтроллер.

Навигационные ИСЗ постоянно знают свои координаты, а также вместе с полезным сигналом излучают его на закодированной частоте *L1* и *L2*. При перехвате модулем этого сигнала, он дешифруется (только сигнал с частотой *L1* – дешифровка *L2* невозможна на гражданских модулях, так как он закодирован Р-кодом и имеет военное предназначение), и вносится в выделенную устройствам управления область памяти модуля.

Принимающая антенна модуля принимает поступающие от ИСЗ закодированные сигналы С/А кодом на частоте *L1*. Приняв и обработав сигнал от первого найденного спутника, осуществилось позиционирование приемника относительно спутника. Приняты сигнал высчитывает время приема, и сравнивая с временем передачи сообщения узнает время прохождения сигнала от спутника до приемника. Однако полученное расстояние, равное произведению скорости света (299 792 км/с) на полученное время дает расстояния от спутника до приемника, притом с большой погрешностью! Но неизвестно остается направление этого расстояния. Найден лишь радиус круга, на концах которого находится приемник (ось X) относительно спутника. Модуль продолжает поиск второго спутника из другой орбитали, для более точного позиционирования на плоскости *X*-*Y*.

При перехвате сигнала от второго спутника с другой орбиты, приемник фиксирует номер спутника, его орбиту, и далее производит вычисления точно такие же которые были произведены с первым найденным спутником. И также с большой погрешностью. Однако, направление уже известно. Найден еще один радиус круга, на концах которого находится приемник (ось Y). Пересечение радиусов осуществляется в одной точке. Именно в этой точке и находится приемник.

Приемник продолжает поиск третьего ИСЗ. Все это необходимо для того, чтобы можно было позиционировать приемный модуль относительно координат этих спутников также и по высоте (ось Z). Произведя такие же процедуры приема и расчета расстояния, как было сделано с двумя предыдущими спутниками, будет найдена высота относительно уровня моря. Итак, будет позиционировать в трех плоскостях положение приемного модуля относительно трех спутников. Однако, остается нерешенным проблема правильности данных измерений. Поэтому, необходим четвертый спутник, который поможет приемнику скорректировать текущее время по Гринвичскому часовому поясу, по причине наличия на каждом ИСЗ атомных часов.

После установления связи с четвертым спутником происходит прием данных о текущем времени от четвертого спутника. Но так как на прием и расшифровку сообщения уходило время, принятое значение также окажется некорректным. Поэтому в течении некоторого времени производится прием всех сообщений и производится расчет по специальной формуле. Найденные коэффициенты отслеживаются с течением времени и если в течении этого времени результат не меняется – то приемник находит специальный коэффициент, который и влияет на погрешность. И далее он вносит его в принятое время, и далее истинное значение текущего времени найдено.

Далее, после нахождения текущего времени, оно используется для вычисления расстояний до трех первых найденных спутников. И вот теперь они являются истинными.

После получения всех данных от спутника начинается формирование пакета, который будет отправлен микроконтроллеру по последовательному

USART порту. Модуль передает пакеты постоянно, однако в процессе поиска и вычислений всех истинных параметров он непрерывно отправляет пакеты, содержащие в себе пустые поля (или поля заполнены специальными знаками, как это реализовано в основном GPS протоколе NMEA, предназначенном для передачи данных от приемника к вычислительному устройству). Далее сформированные пакеты, передаются по последовательному порту к контролеру (на порт приема RX). Данные, передаваемые модулем GPS, и принимаемые микроконтроллером представляют из себя двоичные сигналы, равные одному байту, и передающие определенное количество байт за единицу времени. Контролер воспринимает эти байты информации как текст, зашифрованный в ASCII код. Далее микроконтроллер сохраняет принятые байты информации, переданные модулем, идентифицирует, и начинает читать и обрабатывать полученную информацию. Структура принимаемых пакетов описана в подразделе 1.6.

3.3 Источник питания

Источником питание является автомобильный бортовой аккумулятор, или иной источник питания, с диапазоном напряжения от 4 до 16 вольт. Несмотря на то, что преобразователь напряжения будет рассчитана на напряжение входное от 4 до 40 вольт, это создаст нагрузку и некоторые проблемы на модуле аналогово-цифрового преобразователя микроконтроллера.

3.4 Каскад согласования измеряемого напряжения

Каскадом согласования измеряемого напряжения необходим для уменьшения напряжения, необходимого для измерения на аналогово-цифровом преобразователе. Это связано с тем, что опорное напряжение будет равно напряжению питания микроконтроллера. Ведь аналого-цифровой преобразователь не сможет принять настолько большое напряжение и разложить его.

Каскадом согласования измеряемого напряжения служит входной каскад, состоящий из элемента сглаживая уровня напряжения и обратной защиты, и регулируемый делитель напряжения, который уменьшает напряжение, подаваемое с бортового источника питания до величины опорного напряжения. Получение напряжение будет подаваться на аналогово-цифровой преобразователь (АЦП) контролера, которое будет раскладываться в сравнении с опорным напряжением.

На входном каскаде необходимо использовать емкость, которая будет сглаживать колебания, возникающие на источнике питания, ввиду того что он может, например, при работающем двигателе автомобиля заряжаться, тем самым напряжение на его концах повысится.

3.5 Датчик освещенности

Датчиком освещенности служит для анализа окружающего света и обработка полученных сигнала от светочувствительного прибора, относительно некоторого опорного сигнала.

Датчик освещенности должен будет иметь возможность определять уровень освещенности извне корпуса комплекса и передавать сигнал на порт общего назначения (а именно ввода/вывода) микроконтроллера и затем обрабатываться им. Допустимое питание датчика освещенности должно входить в диапазон от 1.8 до 5 вольт.

Датчик освещенности обычно состоит из светорезестивного элемента с нагрузочным элементом, а также устройства сравнения – компаратора. Также необходим развязка для работы с компаратором. В качестве нагрузки компаратора можно использовать любую нагрузку с током потребления не более 50 мА. Это могут быть непосредственно обмотки реле, резисторы, светодиоды индикации и оптронов исполнительных устройств, с ограничивающими ток резисторами. Индуктивные нагрузки желательно шунтировать диодами от обратного выброса напряжения. Напряжение питания компаратора может быть от 5 до 36 вольт однополярного (или сумма двухполярного) напряжения. Ввиду того, что сопротивление светорезестивного элемента не фиксировано, то необходимо использовать аналоговый компаратор.

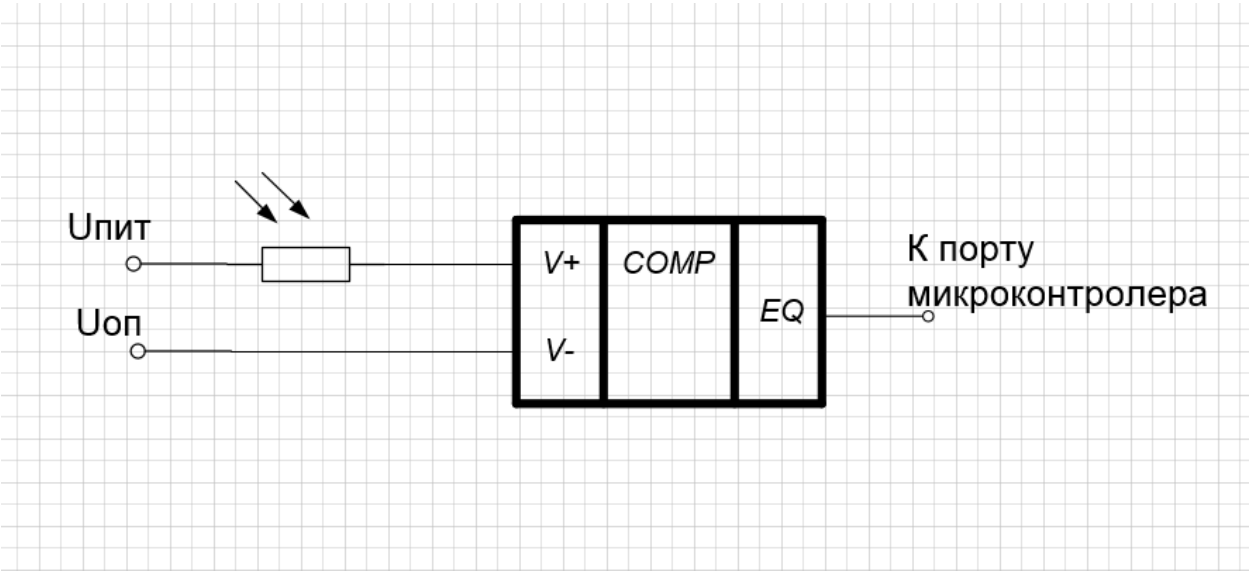

Рисунок 3.3 – Подключение аналогового компаратора

Классическая схема подключения аналогового компаратора с коллектором представлена на рисунке 3.3.

Чтобы иметь устойчивый сигнал, сигнализирующий об освещенности бинарной величиной (логическим нулем или единицей), в зависимости от

степени освещенности фоторезистора, нам необходимо установить порог переключения. Для этого служит не инвертирующий вход (V+) на который необходимо подать опорное (неизменяемое) напряжение. Это опорное напряжение будет взято от преобразователя напряжения и равное напряжению питания всей схемы.

Компаратор будет сравнивать два уровня напряжения (на выводах V+ и V-). Если напряжение на входе V- будет больше чем на входе V+, то на выходе данного модуля будет установлена логический ноль, который в свою очередь будет сигнализировать микроконтроллеру об изменении интенсивности света. Как только напряжение на входе V- опустится (при освещении фоторезистора) ниже уровня напряжения на входе V+, на выходе появится положительный потенциал и установится логическая единица, которая также будет сигнализировать о том, что необходимо произвести некие действия. В данном проекте, это будет использоваться для настройки яркости на модуле индикации (а конкретнее – в матрице светодиодной).

3.6 Преобразователь напряжения

Преобразователь напряжения служит для трансформации входного напряжения в необходимые величины.

Данный функциональный блок должен обеспечивать питанием все датчики и устройства, подключенные к микроконтроллеру напрямую. При рассмотрении документации было установлено, что все датчики и модули, необходимые для работы исполнительных устройств могут работать от 5 вольт. Следовательно, питание, которое будет приходить извне должно быть преобразовано до 5 вольт.

Преобразователь напряжения внутри состоит из входного каскада, преобразователя напряжения и выходного каскада.

Наиболее подходящим вариантом является схема преобразователя, реализованная на импульсном преобразователе. Выбор данной системы связан именно с тем, что линейные преобразователи имеют малую КПД, а также требуют дополнительную систему охлаждения, что будет влиять на габариты.

3.7 Модуль индикации

Модуль индикации — это комплекс устройств, необходимых для сигнализации пользователя о производимых событиях и отображении результата вычислительного процесса аппаратно-программного комплекса.

В состав модуля входит звуковая и знакосинтезирующая индикация. Знакосинтезирующая индикация состоит из трех органических светодиодных дисплеев и одной светодиодной матрице.

3.7.1 Звуковая индикация

Звуковая индикация – это генерация звука, сигнализирующая о каком-то происходящем событии. В данном проекте, этими событиями являются:

– подача питания на прибор;
– нахождение необходимого для навигации числа спутников;
– потеря связи со спутника;
– отсутствие спутников.

Отличительной особенностью каждого события будет разная длительность и генерация периодических звуковых сигналов. Генерация разной длинны и частоты осуществляется с помощью подачи на определенный порт логической единицы командой `PORTx |= (1<<Pxy)` или отключение, путем подачи логического нуля командой `PORTx &= ~(1<<Pxy)`, и задержка, вызываемая функцией `_delay_ms`, определенной в библиотеке `delay.h`. Эта функция ничего не возвращает, но принимает следующий параметр:

1. `double __ms`.

Аргумент `__ms` имеет тип `double` и является временем задержки выполнения текущего кода. Значение `__ms` ограничена в размерности типа данных с плавающей запятой.

3.7.2 Знакосинтезирующая индикация

Знакосинтезирующая индикация – это генерация изображений на средствах, предназначенных для знакоотображения. Такой информацией, которую необходимо отображать, является:

– текущее время (часы и минуты с разделительным символом двоеточия, который мигает с частотой 0,5 Гц);
– текущая скорость в километрах за час (км/ч);
– количество подключенных спутников;
– напряжение аккумуляторной батареи автомобиля в вольтах.

Для большей эффективности, эргономичности и стоимости проекта, был сделан выбор в пользу комбинированной индикации. Отображение скорости является приоритетной, поэтому ее отображать будет знакосинтезирующая светодиодная матрица. А прочая, то есть второстепенная информация, будет отображаться на трех органические светодиодные экраны и большим разрешением. Рассмотрим каждый поподробнее.

3.7.2.1 Знакосинтезирующая светодиодная матрица

Знакосинтезирующая светодиодная матрица представляет собой набор светодиодов, включенных параллельно с общим катодом или анодом. Стандартная матрица имеет 64 светодиода на одном сегменте, квадратной формы с восьмью светодиодами на каждой стороне. Более оптимальный для

человеческого глаза цвет, который бы не раздражал и не слепил водителя в темноте, был выбран именно красный. Также, яркость будет обеспечиваться коррекцией интенсивности путем отправки команд с микроконтроллера, который в свою очередь получает и обрабатывает информацию с датчика освещенности. При темноте интенсивность минимальная, однако при дневном свете полная яркость. Регистр управления, отвечающий за яркость определен как `INTENSITY` и равен шестнадцати битовому адресу `0x0A00`. Младшая половина адреса указывает на принадлежность к командам и необходимости записи в регистр управления, старшая половина адреса указывает собственно адрес регистра, который управляет интенсивностью матричного сегмента. Для организации управлением интенсивности матричным модулем используется функция `SetIntensity`, возвращаюшее пустое значение, и принимающая в качестве параметра аргумент типа `uint8_t` который определен в стандартной библиотеке `stdint.h` языка С. Ниже приведен текст этой функции:

```
void SetIntensity(uint8_t a)  // 0 down to 15
{
     SendLed((INTENSITY >> 8), (SHUTDOWN | a));
}
```

Аргумент `a` это интенсивность сегмента, способное принимать значение от минимального `0x00` до максимального `0x0F`.

Добавление данных будет осуществляться путем записи в драйвер сегментов матрицы данных, которые будут записаны в регистр данных. Под данными, в свою очередь, подразумевается запись в ряд необходимое состояние всех светодиодов в этом ряду. Представление осуществляется в бинарном (8 битном) виде. Единица соответствует состоянию, когда светодиод загорелся, ноль – выключился светодиод. Для организации управлением интенсивности матричным модулем используется функция `WriteNum`, возвращаюшее пустое `void` значение, и принимающая в качестве параметра следующие аргументы:

- `char* z;`
- `char* y;`
- `char* x.`

Аргумент `z` является старшей частью трехзначного числа, имеет тип `char*` и является указателем, который возвращает функция `pgm_read_byte`, который принимает в виде параметра указатель типа `static const char` с атрибутом `PROGMEM`, из файла `pgmspace.h` на массив символа, сохраненного в энергонезависимой памяти `FLASH`. Может принимать значения от `EMPTY` и `0` и до `9`. Аргумент `y` является средней частью трехзначного числа, имеет тип `char*` и является указателем, который возвращает функция `pgm_read_byte`, который принимает в виде параметра указатель типа `static const char` с атрибутом `PROGMEM`, из файла `pgmspace.h` на массив символа, сохраненного в

энергонезависимой памяти FLASH. Может принимать значения от EMPTY и 0 и до 9. Аргумент x является младшей частью трехзначного числа, имеет тип char* и является указателем, который возвращает функция pgm_read_byte, который принимает в виде параметра указатель типа static const char с атрибутом PROGMEM, из файла pgmspace.h на массив символа, сохраненного в энергонезависимой памяти FLASH. Может принимать значения от EMPTY и 0 и до 9. Ниже приведен пример кода на языке C:

```
void WriteNum(char *z, char *y, char *x)
{
     for(int i = 0; i < 8; i++)
     {
          PORTB &= ~(CS);

          SPI_WriteByte(i + 1);
          SPI_WriteByte(pgm_read_byte(&(z[i])));

          SPI_WriteByte(i + 1);
          SPI_WriteByte(pgm_read_byte(&(y[i])));

          SPI_WriteByte(i + 1);
          SPI_WriteByte(pgm_read_byte(&(x[i])));

          PORTB |= CS;
     }
}
```

Наличие цикла for соответствует записи в восемь информационных регистров состояния светодиодной линии в сегменте. Ввиду того, что большинство драйверов светодиодных матриц внешний интерфейс реализован в виде интерфейса *Serial Peripheral Interface* (SPI), то при его использовании возникет необходимость выделения для каждого сегмента отдельного вывода с микроконтроллера, который бы управлял бы по отдельности записью (по концепции SPI это вывод *Chip Select* (CS) на принимаемом устройстве). Наличие одновременной записи сразу трех знакосимволов соответствует решению, принятому с целью экономии выводов контролера и количества дорожек. В итоге, на светодиодную матрицу было задействовано три вывода вместо пяти (без учета двух выводов для питания).

Запись осуществляется через систему интерфейса SPI в режиме *Master* (ведущий) и *Slave* (ведомый). Ведущим в данной ситуации является микроконтроллер, который управляет всем аппаратно-программным комплексом, а ведомым является драйвер светодиодной матрицы, состоящий из параллельно подключенных драйверов, каждый из которых представляющий из себя шеснадцатибитный сдвиговый регистр, соединенные в один единый 48-битный сдвиговый регистр, каждый отдельный драйвер из

которых управляет отдельным сегментом светодиодной сборки, состоящей как, было сказано выше, из 64 светодиодов в одной сегменте.

При выборе элементной базы (а именно микроконтролера) стоит учесть то, что знакосинтезирующая матрица должна будет генерировать цифры в одном сегменте (от 0 до 9 включая пустое поле), символы (G, P, S), при условии, что каждый символ для знакосинтезирующей матрицы имеет размер 8 байт. Итого, для матричного дисплея необходимо зарезервировать место в FLASH памяти 14 байт под цифры и символы.

3.7.2.2 Знакосинтезирующие органические светодиодные экраны

Знакосинтезирующими органическими светодиодными экранами (*organic light-emitting diode*, далее OLED) называются полупроводниковые приборы, изготовленные из органических соединений, эффективно излучающих свет при прохождении через них электрического тока. Управление дисплея осуществляется с помощью специального драйвера, представляющего из себя устройство управления, устройство ввода/вывода и дампа памяти, каждая ячейка которой соединена с элементами OLED дисплея.

Таким параметром, как разрешения дисплея, считается в данном случае количество OLED элементов на дисплее, а также размер дампа памяти, способного работать с данным дисплеем. Размеры дисплеев сильно варьируются, и потому чтобы не терять разрешение или наоборот не увеличивать его тем самым удорожая его, принято делать драйверы под определенный дисплей как абсолютно нелогичное и затратное занятие. Чтобы найти компромисс в данной ситуации, было найдено простое решение: делать дамп памяти дисплея кратным размерам стандартных дисплеев, а использовать его с дисплеями меньшей размерности. Конечно, это повлечет за собой большие неудобства при разработке и работе с такими дисплеями, однако на сегодняшний день это единственное решение. Но разработчики добавили гибкую управляемость адресным пространством, что дает возможность назначать программному указателю драйвера начальное смещение относительно начала памяти, тем самым приняв его за начальное значение дампа. Также есть драйвера с расширенным функционалом, например, управление интенсивностью свечения, инверсия изображения путем инверсии данных во всех ячейках памяти, а также включение и отключение OLED дисплея. В данном проекте самым подходящим будет разрешение 128*32, соответственно дамп памяти будет не меньше 8192 ячеек.

Управлением дампом памяти осуществляется устройством управления, в лице которого является драйвер дисплея. Также, оно конфигурирует и управляет настройками отображения данных, переданного в дамп. Данные, получаемые драйвером, делятся на команды и данные. Данные имеют размер двух байт, первый байт является идентификатором данных, второй бай собственно полезной информацией. Такая раздельная передача вызвана тем, что большинство регистров передачи данных с микроконтроллера из

семейства AVR, имеют разрядность восемь бит. Основные команды определены в файле `OLED.h` в виде макросов.

В связи с тем, что дамп памяти имеет размер 8192, а необходимый размер ячеек 4096, значит имеет значение использование половины дампа. Важно это помнить при написании программного кода, так как можно ошибиться и записать область памяти, которая не будет отражаться на дисплее (невидимая область памяти).

Также, важной проблемой является использование трех дисплеев на интерфейсе I2C. Ввиду того, что они имеют одинаковые адреса, а зачастую на микроконтроллерах имеется только одно устройство I2C, возникает проблема индивидуального использования. Так как по правилам подключения нескольких устройств на шину I2C, подключение производится параллельно. Необходимо добавить устройство демультиплексирования или проще говоря – устройство управлением потоком данных, которое будет переключать информационный и синхронизирующий потоки к необходимому дисплею.

При выборе элементной базы (а именно микроконтроллера) стоит учесть то, что OLED экран должен будет генерировать цифры (от 0 до 9 включая символ двоеточия), значки и эмблемы (в количестве четырех), при условии, что каждый символ для имеет размер 100 байт. Итого, для матричного дисплея необходимо зарезервировать место в FLASH памяти 1 килобайт под цифры и около 2 килобайт под символы, при условии, что каждый занимает 100% дисплея при отрисовке.

3.8 Часы реального времени

Часами реального времени называют модуль, который отсчитывает текущее время, а также имеет возможность настройки и считывания данных.

Модуль часов реального времени состоит из основного устройства реального времени (*real time clock*, далее *RTC*), тактового генератора, долговременного памяти и резервного питания.

На рисунке 3.4 предоставлена блок схема RTC, которая представляет из себя устройство, состоящее из блоков тактирования (*oscillation block*), управления памятью (*power control*), шины интерфейса (*serial bus interface*), устройством управления (*control logic)*, внешнего тактирования (*mux/buffer)*, память *RAM*, часы/календарь и регистры управления (*control register and calendar).* Шина интерфейса является основным связующим с микроконтроллером. Команды идут в обход основного устройства управления (это чем-то походе на работу как в DMA). Память используется для хранения записываемой информации, и имеет зависимость от энергопитания. Часы/календарь и регистры управления служат для занесения команд и данных, а также в их формировании. В формировании участвует блок тактирования, являющийся по сути счетчиком отсчитывающий каждую секунду, и способный формировать сигнал на выходе с частотой 1 Гц. С учетом распространенных номиналов такой счетчик должен быть 15-ти

разрядным, чтобы за одну секунду он точно мог посчитать от 0 до 32767.

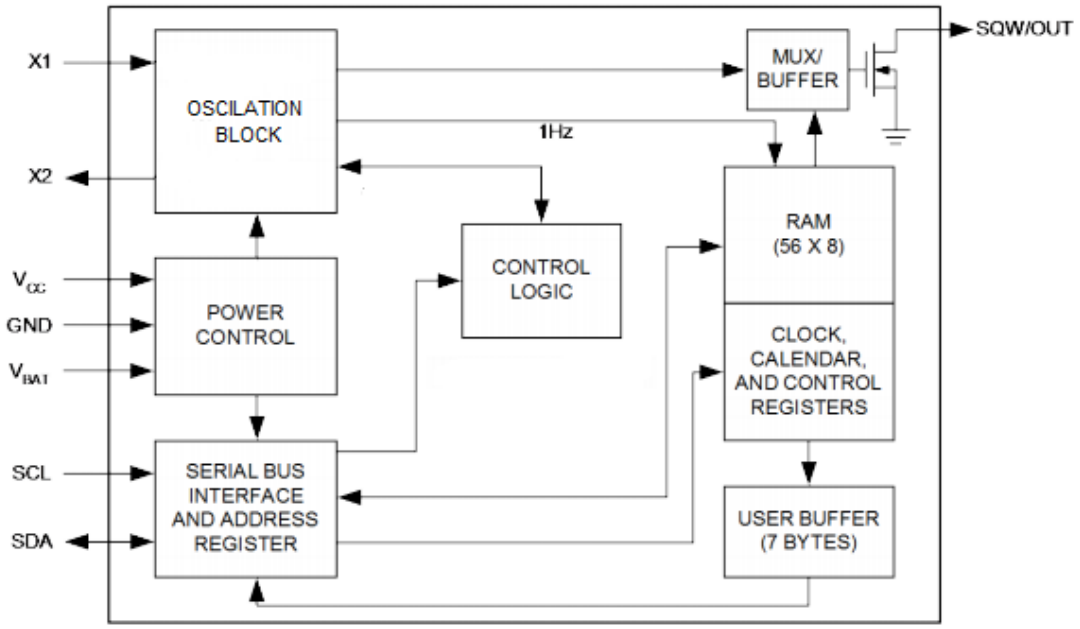

Рисунок 3.4 – Real Time clock

Долговременная память используется для записи и хранения данных, хранение которых необходимо при отсутствии основного питания. Однако, напрямую без источника питания с ней RTC взаимодействовать не сможет. Так как система однопоточная, необходимо участие однотактного устройства управления типа контролера.

Резервное питание необходимо для обеспечения работой основную схему при отсутствии основного питания и поддержания хода часов.

Работу часового механизма обеспечивает тактовый генератор, генерирующий тактированные сигналы с определенной частотой. Частотой, при которой счетчик в блоке тактирования сможет посчитать свое максимальное значение за 1 секунду является 32768 Гц. Именно такое количество колебаний тактового сигнала, пришедших на блок тактирования, ровно за 1 секунду переполнят счетчик от минимального до максимального значения.

3.9. Алгоритм функционирование комплекса

Функционирование всего комплекса выглядит следующим образом.

При подаче питания с источника питания автомобиля оно поступает на преобразователь напряжения и каскад согласования измеряемого напряжения. Так как Множество систем начинают пользоваться этим напряжением, появляется ток. Далее происходит небольшое проседание напряжения (приблизительно на 0,5 В – 1,5 В). Это связано с законом Ома: увеличивается

ток, сопротивление неизменно (сопротивление в данном случае являются все наши потребители (нагрузка) так называемые). Но это не мешает функционированию системы. Далее на выходе каскада преобразования измеряемого напряжения формируется пониженное напряжение относительно напряжения питание в +5 вольт. А на преобразователе напряжения происходит формирование пониженного напряжения равное напряжению питания микроконтроллера равное +5 В. Процесс формирования пониженного напряжения описан в подразделе 3.6.

После преобразователя питание поступает на все системы и подсистемы комплекса. Подается питание на модуль GPS, на котором в сою очередь стабилизирующий преобразователь понижает его до + 3.3 В, которыми происходит запитывание приемника и всех подсистем модуля. И сразу же на линии передачи UART (а именно *TX*) происходит передача сформированных пакетов на модуле GPS. Принцип формирования описан в подразделе 3.2. Но ввиду того что на раннем этапе формируются невалидные пакеты, микроконтроллер игнорирует эти сигналы. Также, питание поступает на систему индикации и часы реального времени. Но не один из этих модулей не начнет работу, так как необходимо настроить их драйвера. Также, на микроконтроллер поступает трансформированное напряжение с каскада согласования измеряемого напряжения.

При подаче питания микроконтроллер начинает выполнять код программы, расположенной в сегменте кода памяти FLASH. На начальном этапе производится инициализация и настройка всех систем и внутренней периферии контролера. В первую очередь происходит поиск адреса с главной исполнительной функцией. Далее идет вход в адрес начала этой функции и выполнение ее инструкций.

Для начала идет настройка подсистемы GPIO, далее идет сигнализированние о том, что комплекс включен и начал работу, посредством подачи логической единицы на порт, к которому подключен динамик. Далее следует двухсекундная задержка.

Далее происходит настройка подсистемы интерфейсов SPI, UART и I2C. При настройке последовательного порта UART настраивается обработчик прерывания от данной подсистемы и настраивается обработка байтов, приходящих с этого модуля на ядро микроконтроллера. Однако, сразу включается запрет прерываний, чтобы не мешать настраивать остальные подсистемы, тем более что первые пакеты, приходящие от модуля GPS невалидны, и расходования системного времени на их обработку считаю нецелесообразной и излишней.

После настройки интерфейсов осуществляется настройка системы индикации посредством передачи команд:

– по SPI знакосинтезирующей матрице команд на настройку внутренних систем и драйвера, а также очистка внутреннего дампа памяти и включение отображение этой памяти;

– по I2C интерфейсу трем знакосинтезирующим OLED дисплеям, предварительно передав необходимый адрес на устройство разадресации системы индикации.

Далее, по интерфейсу I2C происходит инициализация часов реального времени. Устанавливается начальное время 00:00:00 и запускается тактирование.

Далее свою работу начинает подсистема АЦП. Осуществляется ее инициализация и анализ принятого от каскада согласования измеряемого напряжения от бортового источника питания относительно источника питания. После задержки на вычисления полученной величины появляется величина имеющая размер от 0 до 1024. Оно сохраняется в оперативной памяти SRAM, и далее заносится в OLED дисплей и отображается. Также настраивается прерывание от АЦП. При изменении измеряемой величины происходит вызов обработчика прерывания, в котором осуществляется изменение переменной, которая хранит данные об напряжении источника питания, и устанавливается флаг о том, что данные изменены. Прерывание оканчивается. Именно такое краткое событие необходимо потому, что нахождение программного счетчика (или как его еще называют «*Instruction Pointer*», «*Program Counter*») в теле прерывания плохо скажется на работе свей системы в целом. Это считается «дурным тоном» не только потому, что контролер большую часть действий делает в прерывании, но и еще то что происходит резкая смена контекста, а старый контекст сохранен в стеке, пока прерывание не будет окончено. Но стек не бесконечен, как и время его задержки. В процессе вычислительных операций стек также заполняется, что может повлечь нарушение контекста и потерю данных. Поэтому самым безопасным и рекомендованным способом от ведущих производителей микропроцессорных систем является установка особого, предварительно созданного разработчиком флага, который будет опрашивается в процессе нахождения в бесконечном цикле главной программы. Это более безопасно, и проще для вычислительных возможностей микроконтроллера, да и инструкций данный подход занимает наименьшее количество.

Далее осуществляется отображения логотипов и изображений спутника и знака разделения часов и минут. Осуществляется разрешение внешних прерываний от подсистем контролера. Далее следует бесконечный цикл

Вызывается прерывание от подсистемы UART о приеме байта данных. После получения кучи байт и сложении их в один пакет выставляется флаг о приходе пакета. Далее в цикле, при опросе всех существующих пользовательских флагов вызывается функция обработки пакета. Внутри нее анализируется тип пакета, и из тела этой функции вызывается следующая, в зависимости от того какой тип пакета необходимо обработать. Такое разделение вызвано тем, что размеры и значения пакетов сильно разняться и каждый раз динамически вычислять одной функцией очень избыточно по растратам процессорного времени. При нахождении валидных пакетов

начинается их парсинг полученного пакета, и обработка данных. Далее результат обработки посылается на устройства индикации. Скорость – на знакосинтезирующую матрицу, а время и количество спутников – на знакосинтезирующие OLED дисплеи.

Также, осуществляет свою работу датчик освещенности, построенный на аналоговом компараторе и фоторезисторе. При наличии света на нем формируется логический ноль. При таком наборе данных микроконтроллер посылает команду на увеличение яркости на светодиодные матрицы и увеличение контраста на знакосинтезирующих дисплеях. Как только количество света уменьшается настолько, что провоцирует изменение сигнала на аналоговом компараторе, то на вход микроконтроллера меняется данные на логическую единицу. Это сигнал к тому, что необходимо отправить команду на знакосинтезирующие матрицу и дисплеи о понижении яркости и контраста. Если этого не выполнить, то свечение этих знакосинтезирующих элементов будет слепить пользователя и мешать ему осуществлять движение на транспортном средстве в темное или сумрачное время суток.

4 РАЗРАБОТКА ПРИНЦИПИАЛЬНОЙ СХЕМЫ

На данном этапе будут выбраны конкретные модели интегральных микросхем, выбраны схемы их включения, рассчитаны вторичные источники питания, выбраны интерфейсы для сопряжения с MPU. На электрической принципиальной схеме ГУИР.400201.034 Э3 реализованы блоки коммутации пассивных элементов к микроконтроллеру, подключение средств индикации к микроконтроллеру, а также коммутацию внешних модулей.

4.1 Обоснование выбора схемы и расчет дополнительных элементов

Основным элементом электрической принципиальной схемы является микроконтроллер.

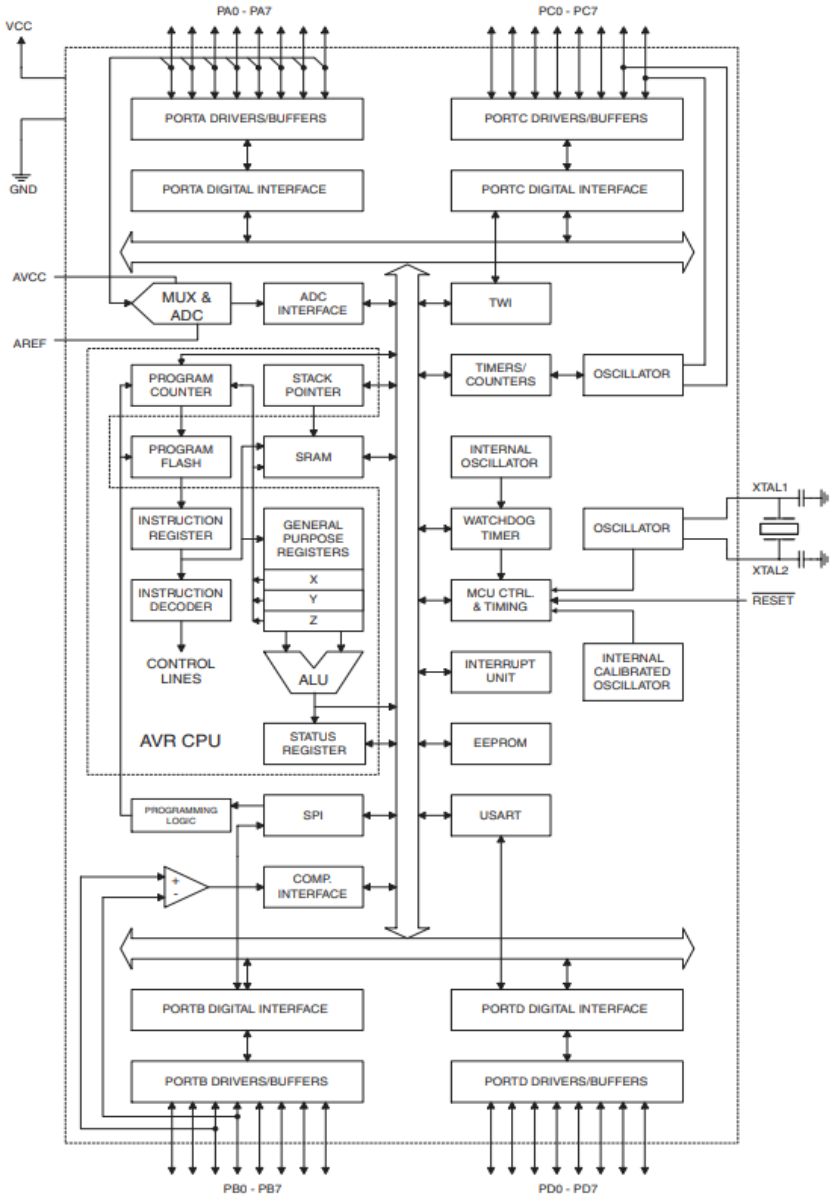

Рисунок 4.1 – Блок-схема микроконтроллера ATmega16

После того как были определены основные задачи, возложенные на данный микроконтроллер и архитектура которая будет использована, было принято решение использовать микроконтроллер архитектуры AVR из семейства Mega, который имеет все необходимые интерфейсы для взаимодействия с периферийными устройства и знако-звукооповещения, прием и передача данных на высокоскоростных и последовательных интерфейсах SPI и I2C, а также с последовательным интерфейсом UART. Исходя из требований подпункта 3.1.10, необходимо выбрать чип с высокой (от 10 Кб памяти FLASH (с учетом обязательных данных символов и чисел, но и код и промежуточные переменные, и константы) и от 1 Кб ОЗУ). Наиболее подходящей является микроконтроллер ATmega16A, имеющий 40 пинов, 1 Кб ОЗУ и 16 Кб памяти FLASH, имеет все необходимые внутренние периферии. Также имеет возможность использовании встраиваемых операционных систем, что дает возможность на модернизацию и продолжение усовершенствования комплекса, как минимум, повысив быстродействие всей системы, путем использование так называемого «параллелизма» и многопоточности.

Также, к микроконтроллеру будет подключен внешний кварцевый резонатор для точного тактирования оного. Частота работы – 8 МГц. Применение внешнего кварцевого резонатора вызвана тем что внутренний кварцевый резонатор подвержен сильно к температурным зависимостям. Это с одной стороны некритично, но многие связи и взаимодействие завязано на тактовом сигнале, выдаваемое микроконтроллером. И поэтому, для более совершенной конструкции и точности, будет использоваться внешний, в обвязке с конденсаторами на 20 пФ.

4.1.1 Преобразование питания

Данный функциональный блок должен обеспечивать питанием все датчики и устройства, подключенные к микроконтроллеру напрямую. При рассмотрении документации было установлено, что все датчики и модули, необходимые для работы исполнительных устройств могут работать от 5 вольт. Следовательно, питание, которое будет приходить извне должно быть преобразовано до 5 вольт. Полученное напряжение от источника питание будет варьироваться от 11 до 16 вольт.

Наиболее подходящим вариантом является схема преобразователя, реализованная на импульсном преобразователе. Выбор данной системы связан именно с тем, что линейные преобразователи имеют малую КПД, а также требуют дополнительную систему охлаждения, что будет влиять на габариты.

Для этих целей будет использоваться DC/DC импульсный преобразователь LM2596. Выбор данного преобразователя обусловлен те, что диапазон его входных напряжений от 4 до 40 В, выходное напряжение, регулируемое от 1,25 до 35 В.

Преобразователь напряжения внутри состоит из входного каскада,

преобразователя напряжения и выходного каскада.

Входной каскад служит для удаления пульсации с источника бортового питания, а также к медленной динамической работе, в случае резкого отключения питания. Удалять такие биение проще всего простыми фильтрами, которыми могут выступить конденсаторы.

Далее следует собственно сам преобразователь, который реализуется в стабилизационной микросхеме. Ее роль является понижение входного напряжения с бортового питания путем кратковременных импульсов, которые после сглаживания попадают на нагрузку, то есть на всю нашу систему. Через резистор R1 регулируется количество и частота этих импульсов. Импульсный преобразователь имеет большой плюс в отсутствии использования высокомощных элементов обвязки, а также экономит место в корпусе и на плате. Однако есть минут – низкая помехоустойчивость. Но в нашем проекте нету такого требования как высокоточная напряжения питания, так что этим можно пренебречь.

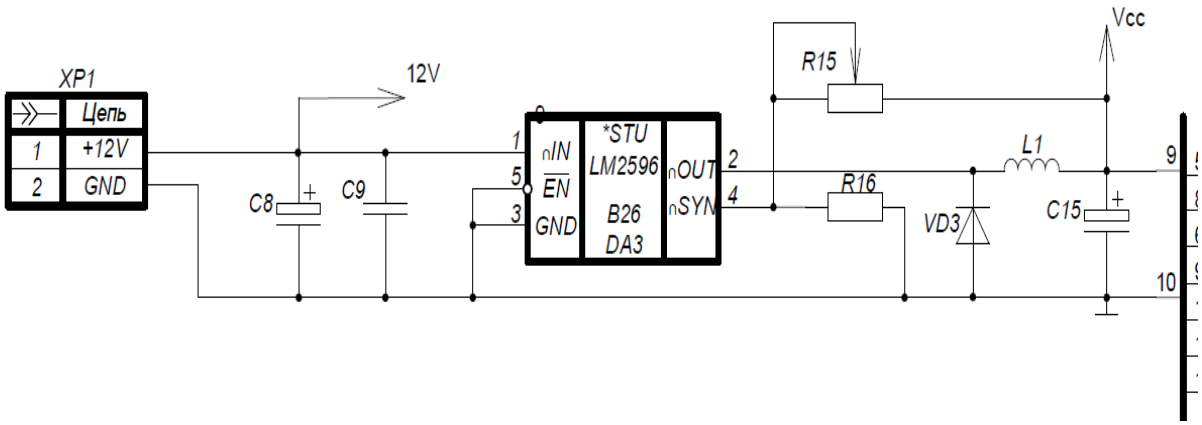

Рисунок 4.2 – Подключение преобразователя напряжения LM2596

Далее следует выходной каскад, состоящий внутри из подстроченного резистора, защитного диода, фильтрующего конденсатора и катушки индуктивности. При прохождении пульсирующего напряжения, полученного с импульсного стабилизатора, через катушку возникает ЭДС самоиндукции. Далее оно стабилизируется конденсатором, который сглаживает пульсацию. Далее напряжение попадает на делитель напряжения, состоящий из R1 и VR1. С помощью подстроенного резистора можно на месте, во время сборки сделать коррекцию получаемого напряжения. Так как далее оно попадает на вход стабилизатора, который отвечает за внесение изменений в выходной сигнал (изменение заполняемости, длительности, ведь работа импульсного преобразователя схожа с частотной модуляцией). Это называется обратная связь. Наличие диода необходимо для отсечение отрицательной полуволны пульсирующего напряжения.

4.1.2 Расчет делителя для каскада согласования измеряемого напряжения

Как описывалось ранее в подразделе 3.4, каскад согласования измеряемого напряжения необходим для уменьшения напряжения, необходимого для измерения на аналогово-цифровом преобразователе микроконтроллера. Это связано с тем, что максимальное опорное напряжение, которое можно подать на вход АЦП микроконтроллера будет равно напряжению питания микроконтроллера (5 Вольт). Ведь аналого-цифровой преобразователь не сможет принять настолько большое напряжение и разложить его.

Для классической схемы делителя, изображенной на рисунке 4.3, необходимо рассчитать сопротивления R1 и R2, из формулы (4.1):

$$U_{вых} = U_{вх} \cdot \frac{R2}{(R1+ R2)} \qquad (4.1)$$

где $U_{вых}$ – выходное напряжение, В;
$U_{вх}$ – входное напряжение, В;
R1 – сопротивление на уменьшение напряжения, Ом;
R2 – сопротивление на увеличение напряжения, Ом.

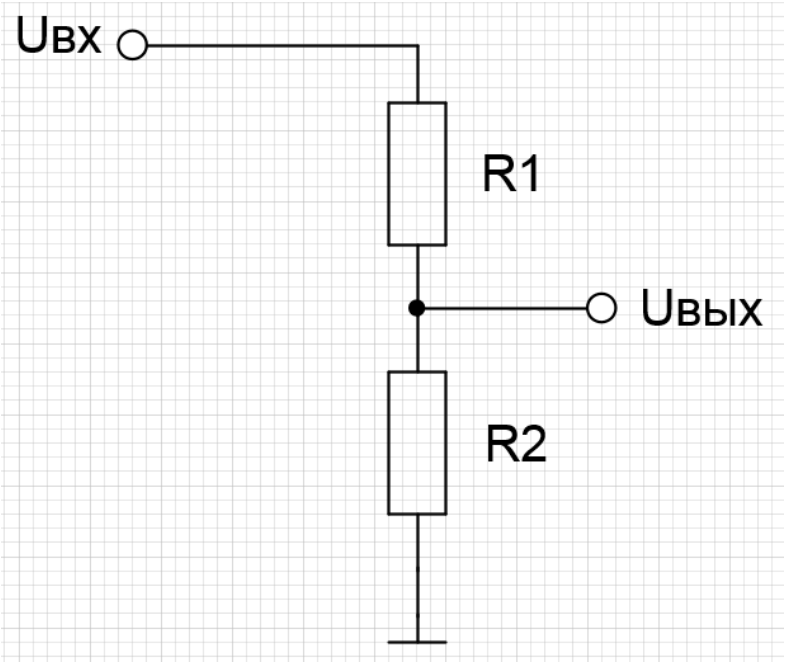

Рисунок 4.3 – Схема делителя напряжения

Исходя из формулы (4.1) и исходит из максимального входного напряжения в 15 В и минимального 11 В, наиболее подходящим вариантом будет размещение R2 построечное сопротивление около 1 кОм для более

точной настройки каскада, а сопротивление R1 = 2.2 кОм. Такая компоновка более эффективна, ввиду того что дискретные элементы имеют допуск погрешности, однако изменение значений может привести к неточным данным на АЦП, и соответственно ошибке и неверному замеру напряжения. Также считаю необходимым подключить на выходе каскада конденсатор, чтобы убрать скачки и пульсации напряжения.

4.1.3 Расчет номиналов для датчика освещенности

Датчик освещенности состоит из фоторезистора с нагрузочным элементом, а также устройства сравнения – компаратора. Также необходим развязка для работы с компаратором. В качестве нагрузки компаратора можно использовать любую нагрузку с током потребления не более 50 мА. Это будут резисторы или светодиоды. Напряжение питания компаратора может быть от 5 до 36 вольт однополярного (или сумма двухполярного) напряжения. Ввиду того, что сопротивление светорезестивного элемента не фиксировано, то необходимо использовать аналоговый компаратор. Самый распространенный и подходящий под наши требования это LM393

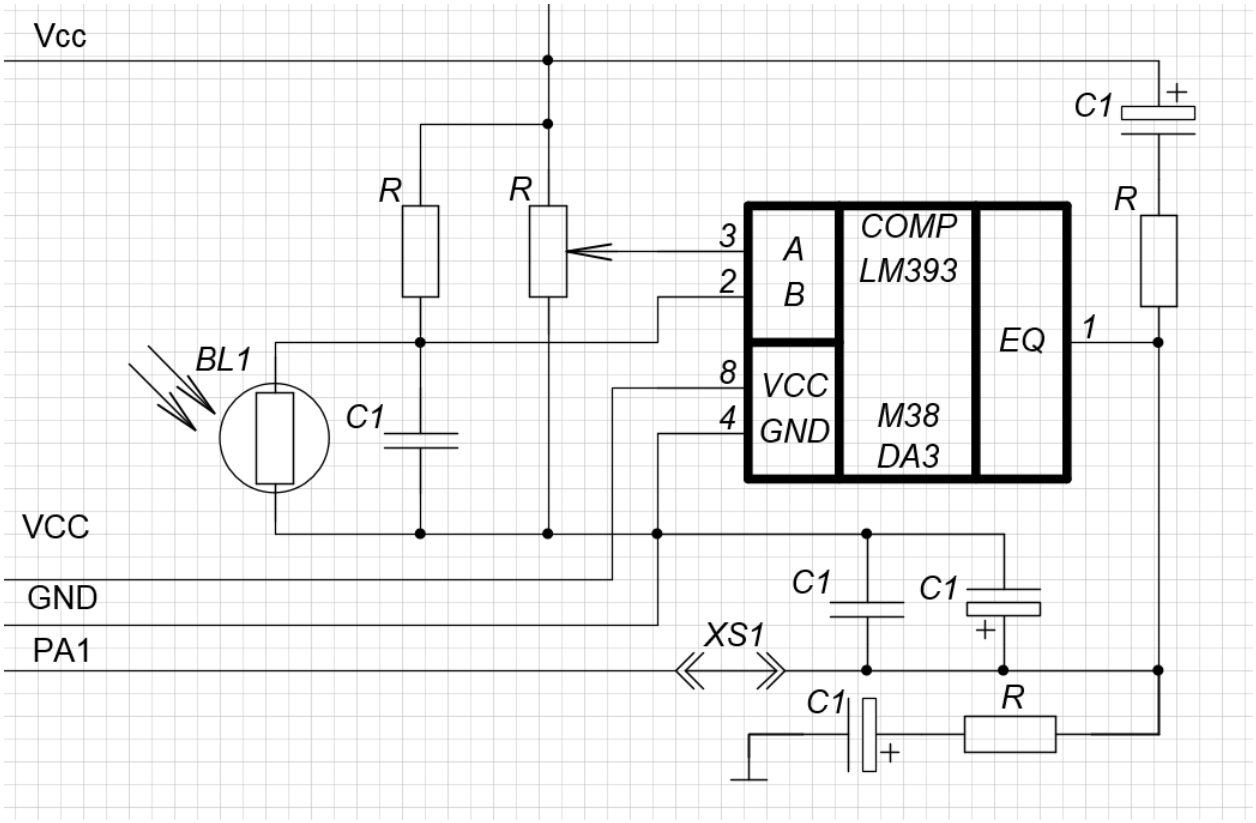

Рисунок 4.4 – Схема датчика освещенности

Глядя на схему (рисунок 4.4), мы видим, что оба входа компаратора подключены к делителям напряжения. Первый делитель напряжения, подключенный к инвертирующему входу (вход B), состоит из постоянного

резистора на 33 кОм и светорезестивного элемента. Как известно сопротивление неосвещенного светорезестивного элемента (фоторезистора) имеет очень большое сопротивление (более 1МОм), и малое при освещении. Поэтому в ночное время суток, согласно логике работы делителя напряжения, напряжение на инвертируемом входе В компаратора будет выше, чем в дневное время суток. Второй делитель напряжения — это построечный резистор на 10 кОм, который соединен выводом с переменного сопротивления и не инвертирующем входом (вход А). Также на схеме видны подключенные конденсаторы, которые сглаживают резкие перепады уровня напряжения на входе и на выходе компаратора, а также на измеряемом фоторезисторе BL1. Также, имеется перемычка монтажная свободного (jumper pins) для подключения выхода компаратора на порт микроконтроллера, или отключении, в случае ненадобности данной функции.

4.1.4 Выбор элементной базы для модуля индикации

Блок индикации, исходя из функциональной схемы, состоит из звуковой индикации, из составной трехразрядной знакосинтезирующей матрицы и трех органических светодиодных дисплея OLED. Начнем с подбора и выбора элементной базы для звуковой индикации.

Схема звуковой индикации изображена на рисунке 4.5 Рассмотрим ее подробнее и опишем принцип работы.

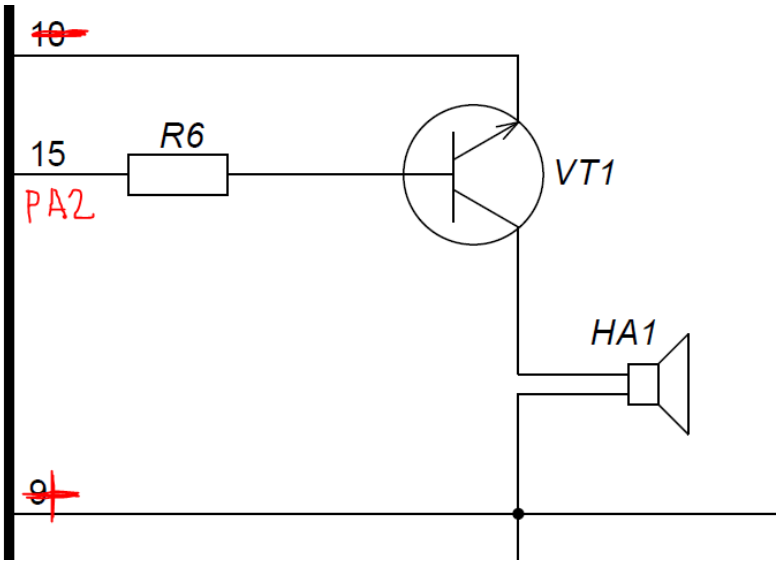

Рисунок 4.5 – Схема звуковой индикации

На данной схеме видно, что используется простая схема включения динамика. Включение и отключение сигнала управляется с помощью выхода микроконтроллера PA2. В условиях применения в автомобиле, или в кабинах машин, в которых возможен шум двигателя, необходимо использовать довольно мощный акустический сигнализирующий электрический компонент.

В связи с этим возникает проблема того, что токоотдача микроконтроллера может не хватить, а также напряжение номинальное больше напряжения высокого уровня, которое может сформировать микроконтроллер. Был выбран звонок магнитоэлектрический HMC1212A на 12 В с частотой 2,3 Гц и подключен он будет одним контактом к положительному напряжению питания 12В. Запускать его будем через транзисторный ключ, реализованный через самый обиходный и универсальный n-p-n транзистор КТ315Б.Транзистор будет замыкать наш звонок с минусом. Открывать и закрывать будет порт микроконтроллера PA2, подключенный к базе транзистора через резистор на 1 кОм, выполняющую роль токоограничения, чтобы транзистор не сгорел.

4.2 Расчет потребляемой мощности

Основными потребителями тока в схеме являются: модуль GPS, интегральные микросхемы и светодиодные матрицы.

Микроконтроллера ток потребления варьируется в очень больших пределах и зависит от множества параметров. Условно, можно считать, что потребление не превысит 500 мА по каждому из напряжений.

Микросхема RTC имеет напряжение питания 5В и имеет ток потребления максимальный 1,5 mA.

Микросхемы переключателя имеет напряжение питания 5В и имеет ток потребления максимальный 0,5 mA.

Микросхемы MAX7219 имеют напряжение питания 5В и ток потребления максимальный 40 mA на одну микросхему. Но так как их три, и подключены они параллельно, а также параллельно работают, следовательно, общий ток потребления около 120 mA.

Учитывая все необходимые условия по питанию микросхем, можно использовать в блоке первичное питание 5 В и вторичные источники питания. Все токи от вторичных источников питания не превышают 0,5 mA. Так как для получения вторичных напряжений используется импульсный стабилизатор напряжения, то остальное напряжение падает на самих стабилизаторах. Суммарная потребляемая мощность модуля составляет:

$$P=5 * (0,5 + 0,0015 +0,00\ 1 + 0,12 + 0,3) = 4,612 \text{ Вт.}$$

Учитывая все дополнительные потери, можно округлить потребление до 5 Вт. Из этого следует, что модуль потребляет почти 1А тока от первичного напряжения в 5 В. Подобранный блок преобразователя рассчитан на выходной ток 3А и мощность 15 Вт.

5 РАЗРАБОТКА ПРОГРАММНОЙ ЧАСТИ

В этом разделе будут описаны основные библиотечные функции и переменные, которые были использованы в проекте, а также описание собственных разработанных методов и шаги использования данных методов и константных переменных в коде основной программы, которая должна исполнять главный алгоритм аппаратно-программного комплекса по измерению скорости объекта.

Весь проект написан на языке программирования С и поэтому также использует встроенные в среду разработки ATMEL модули и возможности. В данном разделе будут описаны внутренние алгоритмы ключевых методов и сущностей, о которых было рассказано в предыдущем разделе.

5.1 Разработка алгоритмов работы с знакосинтезирующей матрицей

В матрицах используется микросхема MAX7219, основным интерфейсом является SPI, поэтому необходимо проинициализировать и настроить внутреннею SPI периферию микроконтроллера. И далее начать настраивать светодиодную матрицу. Данный код описан в файле `LED_MAX7219.c`

Перед началом работы необходимо настроить конфигурации дисплея и его отображение, а также создание функции отправки пакета по интерфейсу SPI (для более лучшего удобства).

```
    SPCR = (0<<SPIE) | (1<<SPE) | (0<<DORD) | (1<<MSTR) |
(0<<CPOL) | (0<<CPHA) | (0<<SPR1) | (0<<SPR0);
    SPSR =(0<<SPI2X);
```

Далее мы опишем четыре основных функции, которые будут использоваться в передаче данных: `SPI_WriteEndByte`, `SPI_WriteStartByte` `SPI_WriteEndByte`, `SPI_WriteByte`. Это необходимо для упрощенной работы и использование их в дальнейшем.

```
void SPI_WriteStartByte(char data)
{
     PORTB &= ~(CS);
     SPDR = data;
     while(!(SPSR & (1<<SPIF)));
}
```

Данная функция осуществляет прием параметра data и отправляет в буфер обмена, а оттуда в сдвиговый регистр интерфейса. Также, необходимо разрешить прием данных на интерфейсе посредством установки низкого уровня на линии `CS` (`PB4`). Где далее по синхросигналу идет передача этого байта из буфера за восемь синхросигналов. Именно по этой причине

используется условие (`! (SPSR & (1<<SPIF))`) ожидания в бесконечном цикле.

Далее необходима передачи последнего байта. Отличие от большинства функций в наличии подачи большого уровня сигнала на порт выбора устройства:

```
void SPI_WriteEndByte(char data)
{
      SPDR = data;
      while(!(SPSR & (1<<SPIF)));
      PORTB |= (CS);
}
```

Данная функция осуществляет прием параметра data и отправляет в буфер обмена, а оттуда в сдвиговый регистр интерфейса. По синхросигналу идет передача этого байта из буфера за восемь синхросигналов. Именно по этой причине используется условие (`!(SPSR & (1<<SPIF))`) ожидания в бесконечном цикле. После передачи всего сообщения и освобождении буфера приемный регистр на стороне слейва (драйвер матрицы) необходимо сохранить (защелкнуть) данные, посредством установки высокого уровня на линии `CS` (`PB4`).

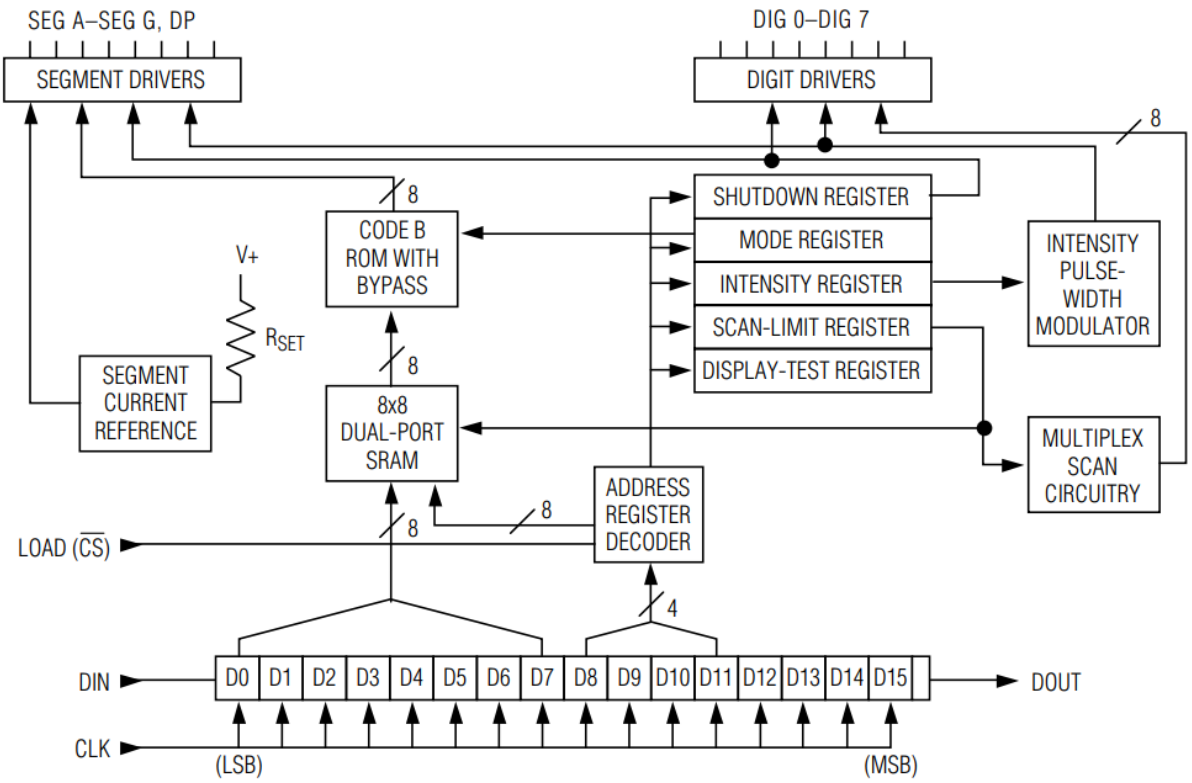

Рисунок 5.1 – Структура принимаемых данных MAX7219

Далее необходима для стандартной передачи байта. Отличие от большинства функций в отсутствии изменении уровня сигнала на порту

выбора устройства:

```
void SPI_WriteByte(char data)
{
      SPDR = data;
      while(!(SPSR & (1<<SPIF)));
}
```

Далее мы опишем четыре функции, которые будут использоваться в передаче конкретных команд на микросхему драйвера MAX7219: `SendLed`, `ClearDisplay`, `WriteNum`, `SetIntensity`. Рассмотрим их ниже поподробнее.

Функция `SendLed` используется для передачи команд следующего формата (см. рисунок 5.1). На рисунке мы видим, что четыре старших бита (D12 – D15) не используются и наличие там информации бесполезно. В D8, D9, D10 и D11 бите располагается адрес необходимого ряда, в младших байтах D0 – D7 указаны позиции двойном коде, где необходимо зажечь светодиод. Допустимые значения от 1 до 8. Более старшие значения используются в управлении драйвером. Число 0 является командой `noop`.

```
void SendLed(char adr, char data)
{
      int i = 0;
      while(i < 3)
      {
            SPI_WriteStartByte(adr);
            SPI_WriteEndByte(data);
            i++;
      }
}
```

Данная функция осуществляет прием параметров `data` и `adr`, которые используются для передачи адреса и данных в этот адрес. отправляет в буфер обмена, а оттуда в сдвиговый регистр интерфейса. По синхросигналу идет передача этого байта адреса из буфера за восемь синхросигналов. После ожидания осуществляется передача 8 битных данных. После передачи всего сообщения и освобождении буфера приемный регистр на стороне слейва (драйвер матрицы) необходимо сохранить (защелкнуть) данные, посредством установки высокого уровня на линии `CS` (`PB4`). Данная операция проводится три раза, так как все три драйвера подключены последовательно после каждой записи необходимо защелкивать данные в памяти драйвера. А далее эти данные перемещаются(сдвигаются) в другой драйвер, так как это получается, как один большой сдвиговый регистр.

Далее необходима для стандартной передачи байта. Отличие от большинства функций в отсутствии изменении уровня сигнала на порту выбора устройства:

Функция `InitLed` используется для инициализации дисплеев:

```
void InitLed(void)
{
     SendLed((DISPLAY_TEST >> 8), (DISPLAY_TEST | 0x00));
     SendLed((INTENSITY >> 8), (INTENSITY | 0x0f));
     SendLed((SCAN_LIMIT >> 8), (SCAN_LIMIT | 0x07));
     SendLed((NO_DECODE_MODE >> 8), (NO_DECODE_MODE | 0x00));
     SendLed((SHUTDOWN >> 8), (SHUTDOWN | 0x00));
}
```

Сначала посылается команда на отключения режима «Тест» в драйвере по адресу `DISPLAY_TEST` (0x0F00), затем по адресу `INTENSITY` устанавливается яркость свечения светодиодной матрицы в диапазоне от 0 до 255. Далее идет включения режима на 7 и отключение режима декодирования из BCD.После инициализации сразу включать отображение бессмысленно, так как ячейки памяти драйвера находятся в нестабильном состоянии.

Функция `ClearDisplay` используется для очистки дампа памяти драйверов:

```
void ClearDisplay()
{
     for(char j = 1; j <= 8; j++)
     {
          SendLed(j,0);
     }
     SendLed((SHUTDOWN >> 8), (SHUTDOWN | 0x01));
}
```

Она осуществляет запись в каждую ячейку каждого сегмента пустое значение (равное нулю), путем вызова функции `SendLed`. Далее после этой процедуры производится команда включения дисплея. Это необходимо для того, чтобы процесс очистки не был уловим человеческому глазу, чтобы был эффект естественно движения.

Далее рассмотрим метод `WriteNum` предназначенный для передачи трех символов на знакосинтезирующую матрицу.

```
void WriteNum(char *z, char *y, char *x)
{
     for(int i = 0; i < 8; i++)
     {
          PORTB &= ~(CS);

          SPI_WriteByte(i + 1);
          SPI_WriteByte(pgm_read_byte(&(z[i])));

          SPI_WriteByte(i + 1);
          SPI_WriteByte(pgm_read_byte(&(y[i])));

          SPI_WriteByte(i + 1);
```

```
            SPI_WriteByte(pgm_read_byte(&(x[i])));

            PORTB |= CS;
        }
}
```

Рассмотрим работу метода подробнее. Метод принимает три параметра:

– `char *z` (первый старший порядковый символ);

– `char *y` (второй старший порядковый символ);

– `char *x` (третий младший порядковый символ).

Далее осуществляется запись в драйверы матрицы сначала адрес расположения куда будут записаны данные, потом сами данные. И так три записи из каждого разряда числа, которое необходимо отразить на матрице. И далее этот цикл производится 8 раз – именно такой размер имеют матрицы. Массив данных, который используется в изображении на матрице (например, числа «3») имеет следующий вид:

```
static const unsigned char PROGMEM THREE[8] =
    {
        0b01111110,
        0b01100110,
        0b01100000,
        0b00011100,
        0b01100000,
        0b01100000,
        0b01100110,
        0b00111100
    };
```

Функция `SetIntensity` используется для установки яркости свечения матриц и принимает в качестве параметра число типа `uint8_t`

```
void SetIntensity(uint8_t a)  // 0 down to 15
{
    SendLed((INTENSITY >> 8), (SHUTDOWN | a));
}
```

Также, вместе с установкой яркости, осуществляется отправка команды о включении отображении памяти драйвера.

5.2 Разработка алгоритмов работы с светодиодным дисплеем

Для более простого позиционирования и упрощенного представления введена глобальная переменная `oled_pointer`, имеющая размер одного незнакового байта `uint8_t`. Она необходима для сохранения текущего адреса в дампе памяти драйвера, который управляет отображением в OLED дисплее.

При работе с OLED дисплеями необходимо помнить, что для обращения

к ним (как и к блоку RTC) используется интерфейс I_2C (в терминологии Atmel это TWI). Поэтому, для начала рассмотрим алгоритмы и методы работы с данным интерфейсом и подсистемами микроконтроллера, которые позволяют работать с данным интерфейсом. Прототипы на нижеописанные методы находятся в написанной библиотеке `I2C.h` и `OLED.h`.

5.2.1 Инициализация подсистем, прием и отправка пакетов

Для начала рассмотрим функцию инициализации внутренних подсистем микроконтроллера, которые позволяют пользоваться данным интерфейсом – `i2cInit`.

```
void i2cInit(void)
{
     TWBR = TWBR_VALUE;
     TWSR = 0;
}
```

Настойка осуществляется путем установки регистра статуса TWI в ноль, а также установка частоты и режима устройства по отношению к внешним устройствам (ведущий или ведомый). Значение коэффициента `TWBR_VALUE` = 1. Это самое максимальное число. Также необходимо создать переменную `uint8_t twi_status_register`, которая будет хранить в себе текущее состояние TWI интерфейса.

Чтобы инкапсулировать переменную `oled_pointer`, чтобы защитить ее от случайного изменения или предотвращения постоянного копирования из ОЗУ, необходимо локальна указать ее в файле исполнителе. Но для внешнего доступа необходимо создать примитивный интерфейсы (методы `SetPointer(uint8_t a)` и `GetPointer()`). метод ввода принимает байт, который необходимо присвоить указателю, а метод вывода возвращает значение, которое хранит в себе указатель дисплея.

```
void SetPointer(uint8_t a)
{
     oled_pointer = a;
}
uint8_t GetPointer()
{
     return oled_pointer;
}
```

5.2.2 Разработка алгоритмов работы с интерфейсом I_2C (TWI)

Далее рассмотрим методы для работы с интерфейсом, который необходим для инициализации и настройки, чтения и записи OLED дисплей и часы реального времени.

Для начала рассмотрим метод отправки старт сигнала `i2cstart(uint8_t ask)`, который в качестве параметра принимает семи битный адрес устройства к которому необходимо получить доступ (последний младший бит указывает на тип текущее операции: «0» это чтение, а «1» это запись). И возвращаемое значение это значение регистра статуса передачи по данному интерфейсу.

```
uint8_t i2cstart(uint8_t address)
{
     TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
     while(!(TWCR & (1<<TWINT)))
          ;
     twi_status_register = TWSR & TWSR_MASK;
     if ((twi_status_register != TWI_START) &&
               (twi_status_register != TWI_REP_START))
     {
          return twi_status_register;
     }
     while (!(TWCR & (1<<TWINT)))
          ;
     TWDR = address;
     TWCR = (1<<TWINT) | (1<<TWEN);

     while(!(TWCR & (1<<TWINT)))
          ;
     twi_status_register = TWSR & TWSR_MASK;
     if ((twi_status_register != TWI_MTX_ADR_ACK) &&
               (twi_status_register != TWI_MRX_ADR_ACK)){
          return twi_status_register;
     }
     return 255;
}
```

Данная функция настраивает на передачу данных из буфера обмена и посылает специальный строб-сигнал, говорящий о том, чтобы устройств на линии начали принимать первый байт и анализировать его ожидание от флага статуса. Потом при очередном включении включается передача байта адреса устройство и ожидается прием сигнала квитирования. При завершении успешном возвращается значение статус. Иначе число 255.

Далее рассмотрим функцию чтения `i2cread(uint8_t ask)` имеющую возвращаемый тип незнаковый байт:

```
uint8_t i2cread(uint8_t ask)
{
     uint8_t data;
     while(!(TWCR & (1<<TWINT)))
          ;
     TWCR = (1<<TWINT)|(1<<TWEN)|((ask & 1) << TWEA);
     while(!(TWCR & (1<<TWINT)));
```

```
        data = TWDR;
        twi_status_register = TWSR & TWSR_MASK;

        if ((ask && (twi_status_register != TWI_MRX_DATA_ACK))||
          (!ask && (twi_status_register != TWI_MRX_DATA_NACK)))
        {
             return twi_status_register;
        }
        return data;
}
```

Данная функция настраивает на прием данных и ожидании флагов, которые сигнализируют об окончании приема данных от слейва.

Далее рассмотрим метод отправки стоп сигнала `i2cstop ()`, который не принимает ничего как параметр, и возвращает тоже ничего.

```
void i2cstop(void)
{
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
        while(TWCR & (1<<TWSTO))
             ;
}
```

Здесь мы видим, что пока не закончилось передача стоп сигнала, и квитирование его от слейва, PC будет находится в цикле, ожидая квитанцию подтверждения от ведомого.

Далее рассмотрим метод передачи данных `i2cwrite()`, а точнее записи в ведомого информации. В качестве параметра принимается однобайтная переменная `data` типа `uint8_t`.

```
uint8_t i2cwrite(uint8_t data)
{
        TWDR = data;
        TWCR = (1<< uint8_t);

        while(!(TWCR & (1<<TWINT)))
             ;

        twi_status_register = TWSR & TWSR_MASK;
        if (twi_status_register != TW_MT_DATA_ACK)
        {
             return 1;
        }
        else
        {
             return 0;
        }
}
```

Сначала происходит установка флагов, которые сигнализируют об включении и собственно включают тактирование на линии передачи данных. А также заполняется буферный регистр `TWDR` который потом при тактировании и разрешении передачи передаст свое значение на сдвиговый регистр передачи, а оттуда на линию.

Далее рассмотрим метод `InitOLED()`, который необходим для инициализации и настройки аппаратных, адресных и иных подсистем OLED дисплея. Рассмотрим вызываемые методы по порядку.

```
i2cstart(SSD1306_ADDR);
i2cwrite(CODE_COMMAND);
```

Вначале мы отправляем старт сигнал методом `i2cstart()` который был описан ранее. Затем в драйвер, после успешного установления с ним связи, записывается значение `CODE_COMMAND` = 0x00. Запись его означает, что сейчас записывается будет команда управления.

Далее необходимо отправить команду о выключении отображении дисплея `SSD1306_DISPLAYOFF` = 0xAE, чтобы человеческий глаз не видел все процессы, которые будут мешать. И отправить команду `SSD1306_SETDISPLAYCLOCKDIV` = 0xD5, которая необходима для установки частоты отображения значений из памяти.

```
i2cwrite(SSD1306_DISPLAYOFF);
i2cwrite(SSD1306_SETDISPLAYCLOCKDIV);
i2cwrite(0x80);
```

Далее запустим функцию мультиплексирования:

```
i2cwrite(SSD1306_SETMULTIPLEX);                  // 0xA8
i2cwrite(SSD1306_LCDHEIGHT - 1);
```

Далее установим смещение по указателю:

```
i2cwrite(SSD1306_SETDISPLAYOFFSET);              // 0xD3
i2cwrite(0x0);                                  // no offset
i2cwrite(SSD1306_SETSTARTLINE | 0x0);            // line #0
i2cwrite(SSD1306_CHARGEPUMP);                    // 0x8D
if (0x2 == SSD1306_EXTERNALVCC)
{
     i2cwrite(0x10);
}
else
{
     i2cwrite(0x14);
}
i2cwrite(0x20);
i2cwrite(0x00);
i2cwrite(SSD1306_SEGREMAP | 0x1);
```

```
i2cwrite(SSD1306_COMSCANDEC);
i2cwrite(SSD1306_SETCOMPINS);                    // 0xDA
i2cwrite(0x02);
i2cwrite(SSD1306_SETCONTRAST);                   // 0x81
i2cwrite(0xFF);
i2cwrite(SSD1306_SETPRECHARGE);                  // 0xd9
if (0x2 == SSD1306_EXTERNALVCC)
{
     i2cwrite(0x22);
}
else
{
     i2cwrite(0xF1);
}
i2cwrite(SSD1306_SETVCOMDETECT);                 // 0xDB
i2cwrite(0x40);
i2cwrite(SSD1306_DISPLAYALLON_RESUME);           // 0xA4
i2cwrite(SSD1306_NORMALDISPLAY);                 // 0xA6
i2cwrite(SSD1306_DEACTIVATE_SCROLL);
i2cwrite(SSD1306_DISPLAYON);//--turn on oled panel
i2cstop();
oled_pointer = 0x00;
}
```

После записи и настройки всех параметров необходимо включить отображения дисплея и установить переменную, которая является указателем на текущую позицию в ноль.

5.2.3 Разработка методов работы с OLED дисплеев

Далее рассмотрим метод `SetPointOLED`, который в качестве параметра принимает следующие значения:

– `uint8_t Start_Collumn` является безнаковым байтным значением, который является начальной позицией по столбцам;

– `uint8_t End_Collumn` указывает на конечное положение по колонке;

– `uint8_t Start_Page` необходимо для указания начальной страницы;

– `uint8_t End_Page` указывает конечную страницу.

```
void SetPointOLED(uint8_t Start_Collumn, uint8_t End_Collumn,
uint8_t Start_Page, uint8_t End_Page)
{
     i2cstart(SSD1306_ADDR);
     i2cwrite(CODE_COMMAND);
     i2cwrite(SSD1306_COLUMNADDR);
```

Для начала установим связь с дисплеем, и затем укажем ему о том, что необходима запись на ведомое устройство. И что первым байтом информации будет команда. Затем отправляем собственно саму команду –

SSD1306_COLUMNADDR, которая указывает на установку адреса столбца дисплея. А ниже мы видим, что сразу поле передачи сигнала о том, что сейчас будет команда об изменении адреса начального для столбца, посылается собственно число, которое и воспринимается как новый адрес по столбцу параметром Start_Collumn :

```
        i2cwrite(Start_Collumn);
        i2cwrite(End_Collumn);
        i2cstop();
        i2cstart(SSD1306_ADDR);
        i2cwrite(CODE_COMMAND);
        i2cwrite(SSD1306_PAGEADDR);
        i2cwrite(Start_Page);
        i2cwrite(End_Page);
        i2cstop();
}
```

А после установки конечного адреса для столбца и начало и конец для страницы (то есть строки дисплея) мы отправляем стоп-сигнал об окончании транзакций.

Далее опишем отдельно созданный метод для записи команд OLED_Command, который в качестве параметра принимает команду, которую необходимо записать.

```
void OLED_Command(uint8_t data)
{
        i2cstart(SSD1306_ADDR);
        i2cwrite(CODE_COMMAND);       // Co = 0, D/C = 0
        i2cwrite(data);
        i2cstop();
}
```

Далее рассмотрим функцию очистки дисплея. А точнее его дампа памяти данных, которые необходимо отображать.

```
void ClearOLED()
{
        SetPointOLED(0x00, 0x7F, 0x04, 0x07);

        i2cstart(SSD1306_ADDR);
        i2cwrite(CODE_DATA);

        for(int kk = 0; kk < 4; kk++)
        {
                for(int k = 0; k < 128; k++)
                {
                        i2cwrite(0x00);
                }
        }
```

```
        i2cstop();
        oled_pointer = 0x00;
}
```

В начале видно, что мы устанавливаем драйвер в начало на нулевой адрес. А затем двигаться до 127 адреса по горизонтали. И так все четыре страницы памяти дисплея. Далее начинаем запись в дисплей что следующими придут данные, которые необходимо записать в дамп памяти. Далее производится в двух циклах запись (всего 512 ячеек необходимо изменить). И после мы заканчиваем транзакцию и обнуляем указатель дисплея.

Далее рассмотрим метод `SelectDisplay` который необходим для выбора дисплея, и принимающий в качестве параметра значение номера дисплея. Причина использования данного метода и целесообразность его использования необходима по той причине, что три дисплея, подключенные правильно по параллельной схеме, необходимо мультиплексировать. Для этих целей аппаратное было установлен на схеме электронный коммутатор CB4066, который управляется с помощи трех пинов микроконтролера.

```
void SelectDisplay(int i)
{
        switch(i)
        {
                case 3:
                {
                        PORTB |= (PortB0 | PortB1 | PortB2);
                        break;
                }
                case 0:
                {
                        PORTB |= (PortB0);
                        PORTB &= ~(PortB1 | PortB2);
                        break;
                }
                case 1:
                {
                        PORTB |= (PortB1);
                        PORTB &= ~(PortB0 | PortB2);
                        break;
                }
                case 2:
                {
                        PORTB |= (PortB2);
                        PORTB &= ~(PortB0 | PortB1);
                        break;
                }
        }
}
```

Выше хорошо видно, что управление завязано на подаваемом числе.

Максимальное – это подключение всех дисплеев.

Далее рассмотрим метод записи и отображения на дисплее картинок и логотипов, которые записаны в памяти. В качестве параметра `IMAGE_OLED a` принимается параметр, указывающий на параметры изображения, а `unsigned char *b` является указателем на сам рисунок, находящийся в памяти.

```
void Set_OLED_Image(IMAGE_OLED a, unsigned char *b)
{
     OLED_Command(SSD1306_DISPLAYOFF);
     SetPointOLED(oled_pointer,
          oled_pointer + a.long_image - 1,
          0x04, (0x04 + a.height_image));
```

Видно, что сначала осуществляется отключение отображения дисплея, а затем установка указателей в определенные позиции для записи изображения

```
     i2cstart(SSD1306_ADDR);
     i2cwrite(CODE_DATA);
     for(long int kk = 0; kk < a.array_size; kk++)
     {
          i2cwrite(pgm_read_byte(&(b[kk])));
     }
     i2cstop();
}
```

А далее осуществляется запись по байтно в дамп памяти это изображение.

Далее рассмотрим метод записи и отображения на дисплее чисел от 0 до 9 и разделительных символов, которые записаны в памяти. В качестве параметра `IMAGE_OLED a` принимается параметр, указывающий на параметры числа которое нужно отобразить на дисплее, а `unsigned char *b` является указателем на сам рисунок числа, находящийся в памяти.

```
void Set_OLED_Num(unsigned char *a)
{
SetPointOLED(oled_pointer, (oled_pointer + 0x19), 0x04,
0x07);
i2cstart(SSD1306_ADDR);
i2cwrite(CODE_DATA);
for(int kk = 0; kk < 104; kk++)
{
     i2cwrite(pgm_read_byte(&(a[kk]))); //LSB вверху, MSB
снизу
}
i2cstop();
}
```

А далее осуществляется запись по байтно в дамп памяти это

изображение числа. Хорошо видно, что эту функции очень похожи и несильно уникальны. Единственная их особенность – размер записанных чисел по заполняемости дисплея одинаковый. И фиксирования позиция чисел.

Далее рассмотрим метод определения чисел от 0 до 9 и разделительных символов, которые записаны в памяти. В качестве параметра число, указывающий на число, которое необходимо отобразить. Возвращает указать на структуру этого числа.

```
uint8_t* GetNum(int i)
{
	switch(i)
	{
		case 0:
		{
			return zero_logo;
			break;
		}
		case 1:
		{
			return one_logo;
			break;
		}
		case 2:
		{
			return two_logo;
			break;
		}
		 .
		 .
		 .
		case 9:
		{
			return nine_logo;
			break;
		}
	}
}
```

5.3 Разработка алгоритмов работы с часами реального времени

Для работы с данным модулем будет использоваться методы, описанные в пунктах 5.2.1 – 5.2.3, потому что для связи с ЧРВ будет использоваться также интерфейс I2C. В качестве глобальных переменных используется три переменные типа `uint8_t`:

– `buffer0` используется для промежуточного хранения секунд;

– `buffer1` используется для промежуточного хранения минут;

– `buffer2` используется для промежуточного хранения десятой части часа;

Ниже приведен листинг кода, где инициализируется и включается модуль ЧРВ. Посылаемые данные и команды подтянуты из документации на микросхему DS1307.

```
void DS1307_Init(uint8_t rs)
{
     buffer0 = 0xFF;
     buffer1 = 0xFF;
     buffer2 = 0xFF;
     rs &= 0x03;
     i2cstart(DS1307_ADDR);
     i2cwrite(DS1307_CONTROL_REGISTER);
     i2cwrite(0x13);
     i2cstop();
     i2cstart(DS1307_ADDR);
     i2cwrite(0x00);
     i2cwrite(0x00);
     i2cstop();
}
```

На данном коде хорошо видно как используются стандартные команды чтения и записи по интерфейсу I2C. Которые были описаны ранее. В листинге кода видно, что осуществляется инициализация и запуск тактирования путем отправки команд. Это старт ЧРВ.

Далее рассмотрим метод чтения данных о дате и времени из ЧРВ.

```
uint8_t DS1307_ReadRegister(uint8_t deviceRegister)
{
     uint8_t boo = 0;
     i2cstart(DS1307_ADDR);
     i2cwrite(deviceRegister);
     i2cstop();
     i2cstart(DS1307_ADDR | 0x01);
     boo = i2cread(0x00);
     i2cstop();
     return boo;
}
```

В качестве параметра функции `DS1307_ReadRegister` принимается регистр, из которого необходимо прочесть данные. Возвращаемое значение – байт.

Далее расмотрим метод установки времени в модуль ЧРВ

```
void DS1307_SetTime(uint8_t hour, uint8_t minutes)
{
     i2cstart(DS1307_ADDR);
     i2cwrite(DS1307_MINUTES_REGISTER);
     i2cwrite(minutes);
     i2cwrite(hour);
```

```
        i2cstop();
}
```

Метод `DS1307_SetTime` принимает в качесте параметров два значения:

– `uint8_t  hour` указывпает на число часов в BCD формате, которое необходимо записать;

– `uint8_t minutes` указывпает на число минут в BCD формате, которое необходимо записать;

Далее расмотрим метод записи времени полученого из ЧРВ в дисплей OLED.

```
void GetTime(void)
{
uint8_t second = 0x00,
        minute = 0x00,
        hour = 0x00,
        a = 0x00,
        b = 0x00;
```

Сначала необходимо проинициализировать и объявитьт промежуточные переменные, в которые будет сохранен результат вызова метода который читает ЧРВ.

Для начала получаем секунды из модуля ЧВР

```
second = DS1307_ReadRegister(DS1307_SECONDS_REGISTER);
```

Затем осущевствляем парсинг полученых данных (а именно, секунд) из BCD кода в человеческое значение, путем сдвигов и сохранение для отдельных мест частей, одновременно разбивая на единицы и десятки.

```
a = (second>>4);
b = second;
a&= 7;
b&= 15;
```

Далее осущевствяется сравнение,, было ли изменение. Стоит ли занимать квант времени на запись дисплея. Если так получилось, что изменения произошло (а именн она 1 секунду), то соответственно необходимо обновить данные. Данные будут обновляться по мере их изсменения. Такой подход реализован по причине того, что интерфейс I2C очень медленный, и любое его использование будет сильно просаживать системное время и задерживать на этом больше ресурсов и все больше потреблять кванты времени.

```
if(b != buffer0)
{
        SelectDisplay(0);
```

```
     if(b == 0x00 ||
          b == 0x02|| b == 0x04 || b == 0x06||
          b == 0x08)
     {
          SetPointer(0x3C);
          Set_OLED_Image(Ncolon_struct, Ncolon_logo);
     }
     else
     {
          SetPointer(0x3C);
          Set_OLED_Image(colon_struct, colon_logo);
     }
```

Далее проводится все тоже самое для минут

```
buffer0 = b;
SelectDisplay(1);
SetPointer(0x5B);
Set_OLED_Num(GetNum(b));
SetPointer(0x41);
Set_OLED_Num(GetNum(a));

minute = DS1307_ReadRegister(DS1307_MINUTES_REGISTER);

a = (minute>>4);
b = minute;
a&= 7;
b&= 15;

if(b != buffer1)
{
     SelectDisplay(0);
     SetPointer(0x5B);
     Set_OLED_Num(GetNum(b));
     SetPointer(0x41);
     Set_OLED_Num(GetNum(a));
     buffer1 = b;
```

И точно также, все расписывается для часов. Принцип прост: движение и отслеживание изменений от младших частей к старшим.

```
hour = DS1307_ReadRegister(DS1307_HOURS_REGISTER);
a = (hour>>4);
b = hour;
a&= 3;
b&= 15;
if(b != buffer2)
{
     SetPointer(0x22);
     Set_OLED_Num(GetNum(b));
     SetPointer(0x08);
```

```
            Set_OLED_Num(GetNum(a));
            buffer2 = b;
        }
    }
    }
    else
    {
        return;
    }
}
```

Иначе, если изменений не произошло, необходимо немедлено покаинуть ифункцию и перейти к следующей, и не задерживать время.

Далее расмотрим функции для работы с BCD кодом. Ввиду того, что эта инструкция может занимать довольно большой отрезок системного времени, то было принято решение написать эту часть на ассемблере и сделать ассемблерную вставку. Функция перевода в BCD формат:

```
uint8_t In_BCD(uint8_t n)
{
asm volatile(
     "ld   r26,y+"  "\n\t"
     "clr  r30"          "\n\t"
"bin2bcd0:"              "\n\t"
     "subi r26,10"  "\n\t"
     "brmi bin2bcd1" "\n\t"
     "subi r30,-16" "\n\t"
     "rjmp bin2bcd0" "\n\t"
"bin2bcd1:"              "\n\t"
     "subi r26,-10" "\n\t"
     "add  r30,r26" "\n\t"
     "ret" "\n\t"
::);
}
```

Функция перевода из BCD формата:

```
uint8_t Out_BCD(uint8_t n)
{
asm volatile(
"ld   r30,y"        "\n\t"
"swap r30"               "\n\t"
"andi r30,0xf"      "\n\t"
"mov  r26,r30"      "\n\t"
"lsl  r26"               "\n\t"
"lsl  r26"               "\n\t"
"add  r30,r26"      "\n\t"
"lsl  r30"               "\n\t"
"ld   r26,y+"       "\n\t"
"andi r26,0xf"      "\n\t"
```

```
"add  r30,r26"      "\n\t"
"ret" "\n\t"
::);
}
```

5.4 Разработка алгоритмов работы с модулем GPS

В модуле GPS используется интерфейс UART. Поэтому, для обмена информацией (а именно прием пакетов от модуля) необходимо настроить внутреннею периферию микроконтроллера на прием данных. В первую очередь необходимо создать глобальный массив данных, куда будет собираться вся принятая информация. Пакет имеет размер 1 байт. А сообщение имеет до 80 байт размер.

```
char rx_buffer[RX_BUFFER_SIZE];
```

И эту переменную необходимо локализовать, чтобы за пределы UART ей никто не мог пользоваться без объявления прототипа.

```
char Get_flagRX(void)
{
return flagRX;
}
```

Также необходимо создать функцию приема данных с периферийного внутренего модуля микроконтролера. Которая будет активироватся по выставлению специального флага в главной функции:

```
void USARTReceiveChar(void)
{
//  Устанавливается, когда регистр свободен
char status, data;
int i = 0;
rx_wr_index = 0;
while(!(UCSRA & (1<<RXC)))
     ;
rx_buffer[rx_wr_index++] = UDR;
if(rx_buffer[0] != '$') /*&& rx_wr_index == 0*/
{
     flagEr &= 0;
     flagRX &= 0;
     UCSRB |= (1<<RXCIE);
     return;
}
else
{
     while(1)
     {
          while(!(UCSRA & (1<<RXC)))
```

```
status = UCSRA;
data = UDR;
if ((status & (FRAMING_ERROR |
      PARITY_ERROR | DATA_OVERRUN)) == 0)
{
     rx_buffer[rx_wr_index++]=data;
     #if RX_BUFFER_SIZE == 255
     // special case for receiver buffer size=256
     if (++rx_counter == 0)
     {
          rx_buffer_overflow = 1;
     }
     #else
     if (rx_wr_index == RX_BUFFER_SIZE)
     {
          rx_wr_index = 0;
     }
     if (++rx_counter == RX_BUFFER_SIZE)
     {
          rx_counter = 0;
          //rx_buffer_overflow = 1;
     }
     #endif
}
if (data == '*')
{
     break;
}
```

Выше приведенная была функция приема данных по специальному символу «$», который, согласно протоколу NMEA, сигнализирует систему о том что пришло начало сообщения от модуля GPS. И осуществляет прием побайтно каждого символа. Прерывание срабатывает при каждом появлении сигнала на канале приема. прерывание срабатывает, запускается обработчик прерывания (смотрите ниже). Потом при появлении символа конца сообщения «*» прием заканчивается, флаг выставляется в ноль. Начинается обработка полученного сообщения, определения его типа, передача управления другому методу, который завязан на определенном типе сообщений (в разделе 1 указано, что их около семи видов).

Анализ полученых сообщений приведен в коне ниже:

```
}
if((rx_buffer[4] == 'M' && flag_Set_Time == 0) ||
     (rx_buffer[4] == 'S') ||
     (rx_buffer[4] == 'T') ||
          rx_buffer[2] == 'G') ||
          (rx_buffer[2] == 'L'))
{

}
```

```
}
ProcessingBufferRx();
flagEr = 0;
flagRX &= 0;
UCSRB |= (1<<RXCIE);
}
```

После завершения приема и анализа принятого сообщения осущевствляется переход к методам.

```
void ProcessingBufferRx(void)
{
     if(rx_buffer[3] == 'M')
     {
          if(rx_buffer[16] == 'A')
          {
     DS1307_SetTime(In_BCD(rx_buffer[6]*10 + rx_buffer[7]),
In_BCD(rx_buffer[8]*10 + rx_buffer[9]));
               flag_Set_Time = 1;
          }
          else return;
     }
     else if(rx_buffer[3] == 'T')
     {

     }
     else if(rx_buffer[3] == 'S')
     {

     }
 }
```

Также, необходимо использовать прерывание, ведь постоянный ропрос флагов будет расходовать время работы контролера и тормозить его. Поэтому системное прерывание на прием от модуля последоввательного интерфейса есть лучшее решение.

```
ISR(USARTRXC_vect)
{
     if(flagEr == 0)
     {
          flagRX = 1;
          UCSRB &= ~(1<<RXCIE);
     }


}
```

Эта функция описывает инициализацию и включения модуля USART внутренней периферии контролера. На этапе инициализации необходимо

настроить управляющие регистры периферийного модуля, задать частоту обмена, режимы работы, а также разрешить прерывания на прием.

```
void USART_Init(unsigned int speed
{
     UBRRH = (unsigned char)(speed>>8);
     UBRRL = (unsigned char)speed;

     UCSRB = (1<<RXEN) | (1<<RXCIE);
     UCSRB |= (1<<RXCIE);
     UCSRA = 0;
     UCSRA |= (1<<U2X);
     UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
     sei();
}
```

Вконце обязательно после инициализации необходимо отключить прерывания. Иначе еще на этапе инициализации оно прервется, так как сработает прерывание, котррое вызовет обработчик.

5.6 Настройка внутренней периферии и портов микроконтроллера

Настройка портов осуществляется ни низком уровне путем управления напрямую состояниями портов ввода/вывода.

Настройка выводов порта A:

```
     DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3)
| (1<<DDA2) | (0<<DDA1) | (0<<DDA0);
          // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T
Bit0=T
          PORTA=(0<<PORTA7)   |   (0<<PORTA6)   |   (0<<PORTA5)   |
(0<<PORTA4)   |   (0<<PORTA3)   |   (0<<PORTA2)   |   (0<<PORTA1)   |
(1<<PORTA0);
```

Все выводы, кроме вывода PA2, настроены на выход.

Настройка выводов порта B:

```
     DDRB = PortB7 | (~PortB6) | PortB5 | PortB4 | PortB3 | PortB2
| PortB1 | PortB0;
          // State: Bit7=0 Bit6=T Bit5=0 Bit4=0 Bit3=T Bit2=T Bit1=T
Bit0=T
     PORTB &= ~(PortB7 | PortB6 | PortB5 | PortB4 | PortB3 | PortB2
| PortB1 | PortB0);
```

Все выводы, кроме вывода PB6, настроены на выход. PB6 настроен на вход.

Настройка выводов порта C:

```
     DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3)
| (0<<DDC2) | (0<<DDC1) | (0<<DDC0);
          // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T
Bit0=T
      PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4)
| (0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);
```

Все вывода этого порта настроены на выход.

Настройка выводов порта D:

```
     DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3)
| (0<<DDD2) | (0<<DDD1) | (0<<DDD0);
          // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T
Bit0=T
     PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4)
| (0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);
     }
```

5.7 Разработка алгоритмов работы комплекса

Прежде всего, необходимо подключить основные библиотеки и заголовочные файлы:

– `#define __AVR_ATmega16A__` этот макрос необходим для того чтобы библиотека `iom16a.h` могла точно определить и сконфигурировать файл прошивки и настроить бутлодер;

– `#include "library/OLED.h"` необходим для работы с OLED дисплеями;

– `#include "library/LED_MAX7219.h"` необходим для работы со знакосинтезирующими светодиодными матрицами;

– `#include "library/ports.h"` необходим для насторойки периферии и выходных портов микроконтролера;

– `#include "library/ds1307.h"` необходим для настройки и работы с модклем ЧРВ;

– `#include "library/uart.h"` необходим для настройки и работы с UART;

В начальный момент необходимо проинициализировать выходные порты ввода/вывода, а также запустить интерфейсы I2C и SPI

```
init_ports();
i2cInit();
_delay_ms(100);
PORTA |= PortA2;
_delay_ms(500);
PORTA &= ~PortA2;
```

Также можно заметить, что как только подали питание на устройство, срабатывает звуковая индикация, сигнализирующая о подаче питания и начала работы комплекса.

Далее необходимо проинициализировать дисплей, очистить его содержимое памяти от хаотичного набора, а также провести быстрый тест с задержкой для восприятия для человеческих глаз:

```
InitLed();
ClearDisplay();
SetIntensity(0x0F);
WriteNum(ONE, TWO, THREE);
_delay_ms(100);
WriteNum(SIX, FIVE, FOUR);
_delay_ms(100);
WriteNum(NINE, EITHT, SEVEN);
_delay_ms(500);
WriteNum(EMPTY, ZERO, ZERO);
_delay_ms(100);
WriteNum(G, P, S);
_delay_ms(100);
```

После окончания тестирования матриц, переходим к настройке и тестированию OLED дисплеев. Сразу всех трех экранов.

```
SelectDisplay(3);
InitOLED();
OLED_Command(SSD1306_DISPLAYON);
ClearOLED();
```

Далее переходим к тесту дисплеев поочередно для проверки их работоспособности.

```
SetPointer(0x00);
SelectDisplay(0);
Set_OLED_Image(ansgrem_struct, ansgrem_logo);
_delay_ms(100);
```

Далее отображаем с задержкой эмблему бгуира, для того чтобы подчеркнуть принадлежность данной разработки к нашему заведению.

```
SelectDisplay(1);
Set_OLED_Image(bsuir_struct, bsuir_logo);
_delay_ms(100);
```

Далее на всех трех дисплеях отображается на 0,5с эмблема технолонии, по которой функционирует данная система (соблюдение авторских прав).

```
SelectDisplay(3);
Set_OLED_Image(avr_struct, avr_logo);
_delay_ms(500);
ClearOLED();
```

Прсле оеончания всех тестов переходим к выводу изображения спутника и настройке частов.

```
DS1307_Init(0);
DS1307_SetTime(0x20, 0x48);

SelectDisplay(1);
SetPointer(0x00);
Set_OLED_Image(sputnik_struct, sputnik_logo);
_delay_ms(1000);

SelectDisplay(0);
SetPointer(0x3C);
Set_OLED_Image(colon_struct, colon_logo);
SetIntensity(0x00);
```

Далее необходимо инициализировать UART и разрешить прерывания от этого модуля.

```
USART_Init(MYUBRR);
sei();
```

Далее следует бесконечный цикл, в которм сообственно и происходит вычисление и работа всего програмно аппаратоно комплексаю.

```
while(1)
{
     GetTime();
     _delay_ms(200);
     if(Get_flagRX() == 1)
     {
          USARTReceiveChar();
     }
     if((PINA & (PortA0)) == 0)
     {
          _delay_ms(3000);
          PORTA |= PortA2;
     }
}
return 0;
}
```

6 ТЕСТИРОВАНИЕ АППАРАТНО-ПРОГРАММНОГО КОМПЛЕКСА

Тестированием аппаратно-программного комплекса по измерению скорости объекта является процесс проверки выполнения разработаного алгоритма вычисления и индикации скорости, а также проверка реакции на внешние и внештатные воздействия на комплекс.

Для начала проведем тестирование яркости в ночное время суток. На рисунке видно (рисунок 6.1), что яркость имеет довольно приятную интенсивность для глаз, и изменилась посредством отсутствия солнечного света.

Рисунок 6.1 – Тест на работу датчика освещености

Также, интенсивность свечения красного цвета уменшена путем достижения обклейки затемненной лентой знакосинтезирующие светодиодные матрицы.

Рисунок 6.2 – Тест на точность измерений

Далее протестируем работу и точность измеряемую аппаратно-программным комплексом. На рисунке 6.2 изображен результат теста в сравнении с спидометром автомобиля, на скорости около 80 км/ч. Если брать в расчет, что автомобильный спидометр добавляет к текущей скорости в среднем 5-7 км/ч, то истиная скорость находится на интервале от 73 до 75км/ч. Видно, что прототип отобразил среднее знпчение из этого интервала, что есть неплохой результат.

Следующим тестом будет проверка на отключение питания. При отключении питания происходит сброс и оключение функционирования всех подсистем, кроме часов реального времени без отображения хода часов. Это связано с тем что в изделии отсутвует основного резервного источника питания. Резервыный источник питания присутствует только у ЧРВ.

Следующий тест на воздействие ударных и механических воздействий.

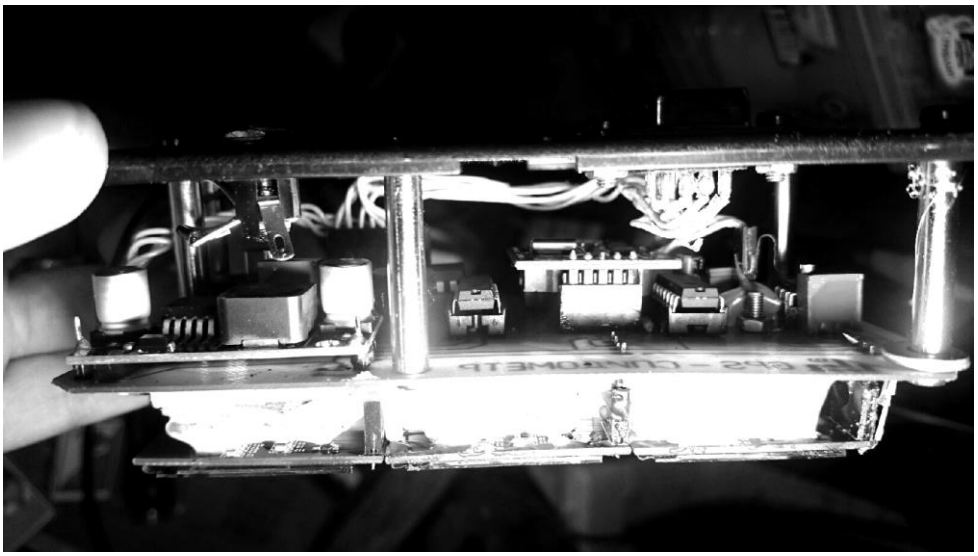

Рисунок 6.3 – Механическое воздействие на прототип

В следствии ударных и механических воздействий было выявлено ненадежность PLS соединений. Контакт нарушается и модули плохо коммутируются с основной платой. Исправлено путем развальцовывания штыревых разъемов PLS. Однако лучшим решение будет в дальнейшем использованием посадочных разъемов типа «цанга».

7 ТЕХНИКО - ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ПРОИЗВОДСТВА КОМПЛЕКСА ИЗМЕРЕНИЯ СКОРОСТИ ОБЪЕКТА

7.1 Характеристика аппаратно-программного комплекса

Разрабатываемый в данном дипломном проекте «Комплекс измерения скорости объекта» является функциональным прибором, который позволяет получать данные о скорости объекта относительно земли, к которому прикреплен комплекс. Данный прибор может быть использован в любой сфере, связанной с использованием транспортных средств (кроме подземных). Его пользователями могут быть как обычные водители, имеющие свой собственный автомобиль или иное транспортное средство, так и предприятия, эксплуатирующие транспортные средства.

Его преимущество — это отсутствие большой погрешности относительно стандартных приборов измерения, которыми снабжают в базовой комплектации. Основные средства по контролю скорости имеют погрешность, а также намерено заниженные показатели. И не отражают реальную обстановку объекта по отношению к окружающему миру. Данный комплекс обеспечивает недорого и точно получать данные по скорости.

7.2 Расчет экономического эффекта от производства аппаратно-программного комплекса

Для производства данного комплекса были использованы следующие материалы (см. таблицу 7.1) и комплектующие (см. таблицу 7.2):

Таблица 7.1 – Расчет затрат на основные и вспомогательные материалы

Наименование материала	Ед. Изм.	Норма расхода материала	Цена, р.	Сумма, р.
1. Припой ПОС 61, 100г	г	0,28	21,00	6,88
2. Флюс ЛТИ-120, 20 мл	мл	0,4	4,40	1,76
3. МГТФ 0.12 кв.мм, Провод монтажный, 1м	м	2,8	0,76	2,13
3. Текстолит двухстороний 1,5мм, 100х100мм	мм	0,9	5,00	4,50
5. Кембрик, 1м	м	1,2	0,59	0,71
6. Стойка дистанценирующая	шт	4	0,34	1,36
Итого				16,34
Всего с учетом транспортных расходов, $Р_м$				17,97

Таблица 7.2 – Расчет затрат на комплектующие изделия.

Наименование материала	Количество на изд, шт.	Норма расхода материала	Цена, р.	Сумма, р.
1. Микроконтролер Atmega16	1	1	17,00	17,00
2. GPS модуль	1	1	22,24	22,24
3. LCD модули 8x8	1	1	19,00	19,00
4. OLED дисплей 128*32	3	3	10,00	30,00
5. Резистор (1 кОм)	3	3	0,09	0,27
6. Кварц 8МГц	1	1	0,35	0,35
7. Разъем BH10	1	1	2,00	2,00
8. Фоторезистор	1	1	3,80	3,80
9. Посадочное место для МК	1	1	1,00	1,00
10. Компаратор	1	1	8,00	8,00
11. Корпус пластиковый	1	1	9,96	9,96
Итого:				113,62
Всего с учетом транспортных расходов, Рк				124,98

Все материалы и комплектующие взяты по ценам, сложившимся на рынке на текущую дату. Коэффициент транспортных расходов взят 1,1.

Расчет общей суммы прямых затрат на производство аппаратной части изделия предоставлен в таблице 7.3.

Таблица 7.3 – Расчет общих прямых затрат.

Показатель	Сумма, р.
1. Сырье и материалы	17,97
2. Покупные комплектующие изделия	118,60
Всего прямые затраты на производство аппаратной части ($З_р^{ач}$)	136,57

Расчет затрат на основную заработную плату разработчиков программной части комплекса представлена в таблице 7.4.

При Расчете зарплаты использовались данные среднемесячной зарплаты в Республике Беларусь для сотрудников IT отрасли. Премия не начисляется.

Основная зарплата определяется по формуле (7.1):

$$З_о = К_{пр} \sum_{i=1}^{n} З_{чi} \cdot t_i, \quad (7.1)$$

где $К_{пр}$ – коэффициент премий (в нашем Расчете)

n – категории исполнителей, занятых разработкой;

$З_{чi}$ – часовая заработная плата исполнителя i-й категории, р.;

t_i – трудоемкость работ, выполняемых исполнителем i-й категории, определяется исходя из сложности разработки программного обеспечения и объема выполняемых им функций, ч.

Часовая заработная плата каждого исполнителя определяется путем деления его месячной заработной платы (оклад плюс надбавки) на количество рабочих часов в месяце (Расчетная норма рабочего времени на 2020г. для 5-дневной недели составляет 168 ч по данным Министерства труда и социальной защиты населения на момент проведения Расчетов).

Таблица 7.4. – Расчет основной зарплаты разработчиков программной части

Категория разработчика	Месячная заработная плата, р.	Часовая заработная плата, р.	Трудоемкость работ, ч	Итого, р.
Инжинер - программист	2150,00	12,80	168	2150,00
Техник - программист	1500,00	8,93	168	1500,00
Итого				3650,00
Премия (50–100 %)				1,00

Дополнительная зарплата разработчиков определяется по формуле (7.2)

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (7.2)$$

где $Н_д$ – норматив дополнительной зарплаты, 15%

Отчисления в фонд социальной защиты населения и обязательное страхование БелГосстрах ($З_{сз}$) определяется в соответствии с действующим законодательством по формуле (7.3)

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100} \quad (7.3)$$

где $Н_{соц}$ – ставка отчислений в фонд социальный защиты населения (ФСЗН)

и БелГосстрах (в соответствии с действующим законодательством на 01.01.2020 г. составляет 34,6%)

Расчет общей суммы затрат на разработку программной части программно-управляемого комплекса в таблице 7.5.

Таблица 7.5 – Расчет затрат на разработку программного средства

Наименование статьи затрат	Расчет по формуле	Сумма, р.
1. Основная заработная плата разработчиков	Табл.7.4	3650,00
2. Дополнительная заработная плата разработчиков	Формула (7.1)	547,50
3. Отчисления на социальные нужды	Формула (7.1)	1452,33
Итого		5649,84

Формирование отпускной цены программно-аппаратного комплекса представлена в таблице 7.6.

Таблица 7.6 – Расчет отпускной цены

Показатель	Расчет по формуле (в таблице)	Сумма, р.
1. Затраты на производство аппаратной части ($З_р^{ач}$)	Табл.7.3	136,57
2. Затраты на разработку программной части ($З_р^{пч}$)	Табл.7.5	5649,84
3. Сумма затрат на производство программно-аппаратного комплекса	$З_{пр}$ =136,57+5649,84	5786,41
4. Накладные расходы	$Р_{накл}$ = 5786,41·0,56	3240,39
5. Расходы на реализацию	$Р_{рел}$= 5786,41·0,02	115,73
6. Полная себестоимость	$С_п$=5786,41+3240,39+115,73	9142,52
7. Плановая прибыль, включаемая в цену	$П_{ед}$=9142,52·25/100	2285,63
8. Отпускная цена	$Ц_{отп}$=9142,52+2285,63	11428,15

Результатом производства аппаратно-программного комплекса является прирост чистой прибыли, полученный от их реализации.

$$\Delta П_ч = П_{ед} \cdot N_п \left(1 - \frac{Н_п}{100}\right), \quad (7.4)$$

где $N_п$ – прогнозируемый годовой объем производства и реализации, шт.;

$П_{ед}$ – прибыль, включаемая в цену, р.;

$Н_п$ – ставка налога на прибыль согласно действующему законодательству, % (по состоянию на 01.01.2020 г. – 18 %).

$$\Delta П_ч = 2285{,}63 \cdot 50\ (1\text{-}0{,}18) = 93710{,}85 \text{ руб}$$

7.3 Расчет инвестиций в проектирование и производство аппаратно-программного комплекса

Инвестиции в производство программно-аппаратного комплекса включают в общем случае:

– инвестиции на его разработку;

– инвестиции в прирост основного капитала (затраты на приобретение необходимого для производства нового изделия оборудования, станков и т.п.);

– инвестиции в прирост собственного оборотного капитала (затраты на приобретение необходимых для производства нового изделия материалов, комплектующих, начатой, но незавершенной продукции и т.п.).

7.3.1 Расчет инвестиций на разработку аппаратно-программного комплекса

Инвестиции рассчитываем ($И_р$) по затратам на разработку нового изделия. Для начала необходимо рассчитать заработную плату технологам и инженерам предприятия-производителя, ввиду того что именно они являются первой статьей первичных затрат.

Расчет заработной платы разработчиков нового изделия в таблице 7.7.

Таблица 7.7 – Расчет заработной платы разработчиков нового изделия

Категория исполнителя	Количество, чел.	Месячная зарплата, р.	Дневная зарплата, р.	Время, д.	Заработная плата, р.
1	2	3	4	5	6
1.Руководитель проекта	1	2350	111,90	84	9400,00
2. Инженер по электронной технике	1	2200	104,76	84	8800,00
3. Техник - технолог	2	1580	75,24	22	3310,48

Продолжение таблицы 7.7

1	2	3	4	5	6
4. Инженер - системотехник	1	2300	109,52	80	8761,90
Итого					30272,38
Премия, 30%					1,3
Всего основная заработная плата (Зо)					39354,10

Расчета затрат на разработку нового изделия в таблице 7.8.

Таблица 7.8 – Расчет инвестиций на разработку нового изделия

Наименование статьи затрат	Расчет по формуле (в таблице)	Сумма, р.
1. Основная заработная плата разработчиков $З_о$	табл. 7.7	39354,10
2. Дополнительная заработная плата разработчиков	$З_д = З_о \cdot 0,15$	5903,11
3. Отчисления на социальные нужды	$Р_{соц} = (З_о + З_д) \cdot 0,346$	15658,99
4. Инвестиции на разработку нового изделия	$И_р = З_о + З_д + Р_{соц}$	60916,20

Инвестиции в прирост основного капитала не требуются, т. к. производство нового изделия планируется осуществлять на действующем оборудовании в связи с наличием на предприятии-производителе свободных производственных мощностей.

7.3.2 Расчет инвестиций в прирост оборотного капитала

1) Определяется годовая потребность в материалах по формуле:

$$П_м = Р_м \cdot N_п, \quad (7.5)$$

где $Р_м$ – затраты на материалы на единицу изделия, р. (см. таблицу 7.1),
Nп - прогнозируемый годовой объем.

$$П_м = 17,97 \cdot 50 = 898,5 \text{ р.}$$

2) Определяется годовая потребность в комплектующих изделиях по

формуле:

$$П_к = Р_к \cdot N_п, \quad (7.6)$$

где $Р_к$ – затраты на комплектующие изделия на единицу продукции, р.

$$Пк = 118{,}60 \cdot 50 = 5930 \text{ руб}$$

3) Определяются инвестиции в прирост собственного оборотного капитала в процентах от годовой потребности в материалах и комплектующих изделиях (исходя из среднего уровня по экономике: 20–30 %) по формуле:

$$\Delta И_{сок} = 0{,}20(П_м + П_к). \quad (7.7)$$

$$\Delta И_{сок} = 0{,}20(898{,}5 + 5930) = 1365{,}7 \text{ руб}$$

Оценка экономической эффективности разработки изделия зависит от результата сравнения инвестиции в разработку и прирост собственных оборотных средств и полученного годового прироста чистой прибыли.

Рассчитаем рентабельность инвестиций по формуле

$$Р_и = \frac{\Delta П_ч}{И_{пр}} \cdot 100\ \%, \quad (7.8)$$

где $\Delta П_ч$ – прирост чистой прибыли, руб.;

$И_{пр}$ – инвестиции в производство ($И_{сок}$ + Ир), руб.

$$Р_и = (93710{,}83/62281{,}92) \cdot 100\% = 150{,}46\%$$

Сравнивая инвестиции в разработку изделия и прирост собственных оборотных средств, с приростом годовой чистой прибыли можно сделать вывод, что инвестиции окупаются в течении года.

Рентабельность инвестиций в производство превысила 100%, следовательно, разработка данного продукта является целесообразным.

ЗАКЛЮЧЕНИЕ

Разработанный в ходе дипломного проектирования аппаратно-программный комплекс измерению скорости объекта – электронный прибор, предназначенный для обработки протокола NMEA0813 с последующим отображением данных на индикации. Дополнительными измеряемыми величинами являются такие как текущее время и напряжение питания. Прибор снабжен также звуковой сигнализацией.

По результатам обзора литературных источников и расчетов были составлены электрическая структурная, электрическая функциональная и электрическая принципиальная схемы. На основании этих схем, была разработана конструкция печатной платы и сборочная документация для прибора, что позволило создать в рамках дипломного проекта опытный образец комплекса по измерению скорости объекта.

Также была разработана методика по эксплуатации устройства, обеспечивающая в полном объеме описание, настройку и использование прибора.

В ходе проведения измерений были получены данные, подтверждающие соответствие разработанного прибора требованиям технического задания.

Проведенный экономический расчет позволил определить цену разработки нового устройства в соответствии с ценами действительными на май 2020 года.

СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ

[1] Дивин, А.Г. Методы и средства измерений, испытаний и контроля: учебное пособие. В 5 ч. / А.Г. Дивин, С.В. Пономарев, Г.В. Мозгова. – Тамбов: Изд-во ФГБОУ ВПО «ТГТУ», 2012. – Ч. 2. – 108 с.

[2] ЕЭК ООН №34 – [Электронный ресурс]. – Электронный данные. – Режим доступа: http://www.unece.org/fileadmin/DAM/t/R039r2r.pdf/ – Дата доступа: 12.04.2020

[3] GPS – [Электронный ресурс]. – Электронный данные. – Режим доступа: https://ru.wikipedia.org/wiki/GPS – Дата доступа: 12.04.2020

[4] Яценков В.С. Основы спутниковой навигации / В.С. Яценков – М.: Атомиздат, 2008 – 124с.

[5] Принцип работы системы GPS [Электронный ресурс]. – Режим доступа: http://kunegin.com/ref6/gps/pringps1.htm Дата доступа: 12.04.2020

[6] Описание протокола *NMEA*0813 [Электронный ресурс]. – Режим доступа: https://www.irz.ru/uploads/files/226_1.pdf – Дата доступа: 12.04.2020

[7] Описание микроконтроллера ATtiny861 – [Электронный ресурс]. – Электронный данные. – Режим доступа : http://ww1.microchip.com/downloads/en/devicedoc/doc2466.pdf – Дата доступа: 12.04.2020

[8] Описание протокола I2C – [Электронный ресурс]. – Электронный данные. – Режим доступа: http://microsin.net/programming/avr/example-using-the-twi-i2c.html – Дата доступа: 12.04.2020

[9] Ревич, Ю.В. Практическое программирование микроконтроллеров Atmel AVR / Ю.В. Ревич. – СПб.: БХВ-Петербург, 2014 – 368 с.

ПРИЛ,ОЖЕНИЕ А

(обязательное)

Код программы

Файл main.c

```
#include "main.h"
#define __AVR_ATmega16A__
uint16_t i;



void ssd1306_command(uint8_t data);

int main(void)
{

	init_ports();
	i2cInit();

	_delay_ms(1000);

	PORTA |= PortA2;
	_delay_ms(1000);
	PORTA &= ~PortA2;

	InitLed();
	ClearDisplay();
	SetIntensity(0x0F);
	WriteNum(ONE, TWO, THREE);
	_delay_ms(1000);
	WriteNum(SIX, FIVE, FOUR);
	_delay_ms(1000);
	WriteNum(NINE, EITHT, SEVEN);
	_delay_ms(5000);
	WriteNum(EMPTY, ZERO, ZERO);
	_delay_ms(1000);
	WriteNum(G, P, S);
	_delay_ms(1000);

	SelectDisplay(3);
	InitOLED();
	OLED_Command(SSD1306_DISPLAYON);
	ClearOLED();

	SetPointer(0x00);
	SelectDisplay(0);
	Set_OLED_Image(ansgrem_struct, ansgrem_logo);
	_delay_ms(1000);

	SelectDisplay(1);
```

```
	Set_OLED_Image(bsuir_struct, bsuir_logo);
	_delay_ms(1000);

	SelectDisplay(3);
	Set_OLED_Image(avr_struct, avr_logo);
	_delay_ms(5000);
	ClearOLED();

	//initSymbolOLED();
	DS1307_Init(0);
	DS1307_SetTime(0x20, 0x48);

	SelectDisplay(1);
	SetPointer(0x00);
	Set_OLED_Image(sputnik_struct, sputnik_logo);
	_delay_ms(1000);

	SelectDisplay(0);
	SetPointer(0x3C);
	Set_OLED_Image(colon_struct, colon_logo);
	SetIntensity(0x00);

	//USART_Init(MYUBRR);
	//sei();
	while(1)
	{
		GetTime();
		WriteNum(ZERO, SEVEN, FOUR);
		_delay_ms(10000);
		WriteNum(ZERO, ZERO, ZERO);
		_delay_ms(2000);
		//if(Get_flagRX() == 1)
		//{
			//USARTReceiveChar();
		//}
		//if((PINA & (PortA0)) == 0)
		//{
			//_delay_ms(3000);
			//PORTA |= PortA2;
		//}
	}
	return 0;
}


void initSymbolOLED(void)
{
	OLED_Command(SSD1306_DISPLAYOFF);

	SelectDisplay(0);
	SetPointer(0x08);
	Set_OLED_Num(GetNum(0));
```

```
     //_delay_ms(1000);
     SetPointer(0x22);
     Set_OLED_Num(GetNum(1));
     //_delay_ms(1000);
     SetPointer(0x41);
     Set_OLED_Num(GetNum(2));
     //_delay_ms(1000);
     SetPointer(0x5B);
     Set_OLED_Num(GetNum(3));
     SelectDisplay(2);
     SetPointer(0x08);
     Set_OLED_Num(GetNum(4));
     //_delay_ms(1000);
     SetPointer(0x22);
     Set_OLED_Num(GetNum(5));
     //_delay_ms(1000);
     SetPointer(0x41);
     Set_OLED_Num(GetNum(6));
     //_delay_ms(1000);
     SetPointer(0x5B);
     Set_OLED_Num(GetNum(7));
     SelectDisplay(2);
     SetPointer(0x08);
     Set_OLED_Num(GetNum(8));
     SetPointer(0x22);
     Set_OLED_Num(GetNum(9));

     OLED_Command(SSD1306_DISPLAYON);
     ClearOLED();
}

#include "uart.h"
#include "init_perif.h"
#include "OLED.h"
#include "LED_MAX7219.h"
#include "ds1307.h"

char rx_buffer[RX_BUFFER_SIZE];


int rx_wr_index = 0, rx_rd_index = 0;
unsigned char rx_counter = 0;
uint8_t buffer;
char flagEr = 0, flagRX = 0, flag_Set_Time = 0;

char Get_flagRX(void)
{
     return flagRX;
}

void USARTReceiveChar(void)
{
```

```
	//  Устанавливается, когда регистр свободен
	char status, data;
	int i = 0;
	rx_wr_index = 0;
	while(!(UCSRA & (1<<RXC)))
		;
	rx_buffer[rx_wr_index++] = UDR;
	if(rx_buffer[0] != '$') /*&& rx_wr_index == 0*/
	{
		flagEr &= 0;
		flagRX &= 0;
		UCSRB |= (1<<RXCIE);
		return;
	}
	else
	{
		while(1)
		{
			while(!(UCSRA & (1<<RXC)))
				;
			status = UCSRA;
			data = UDR;
			if  ((status  &  (FRAMING_ERROR  |  PARITY_ERROR  |
DATA_OVERRUN)) == 0)
			{
				rx_buffer[rx_wr_index++]=data;
				#if RX_BUFFER_SIZE == 255
				// special case for receiver buffer size=256
				if (++rx_counter == 0)
				{
					rx_buffer_overflow = 1;
				}
				#else
				if (rx_wr_index == RX_BUFFER_SIZE)
				{
					rx_wr_index = 0;
				}
				if (++rx_counter == RX_BUFFER_SIZE)
				{
					rx_counter = 0;
					//rx_buffer_overflow = 1;
				}
				#endif
			}
			if (data == '*')
			{
				break;
			}

		}
		if((rx_buffer[4] == 'M' && flag_Set_Time == 0) ||
			(rx_buffer[4] == 'S') ||
```

```
                (rx_buffer[4] == 'T')) /*rx_buffer[2] == 'G' /*||
rx_buffer[2] == 'L' ||*/
          {

          }
     }


                //status = UCSRA;
                //data = UDR;
                //if ((status & (FRAMING_ERROR | PARITY_ERROR |
DATA_OVERRUN))==0)
                //{
                     //rx_buffer[rx_wr_index++] = data;
                //}

                //if(data == 0x43)
                //{
                     //flagEr = 255;
                     //break;
                //}

                //return UDR;




     ProcessingBufferRx();
     flagEr = 0;
     flagRX &= 0;
     UCSRB |= (1<<RXCIE);
}

void ProcessingBufferRx(void)
{
     //if(rx_buffer[2] == 'G')
     //{
          //if(rx_buffer[
     //}
     if(rx_buffer[3] == 'M')
     {
          if(rx_buffer[16] == 'A')
          {
                DS1307_SetTime(In_BCD(rx_buffer[6]*10           +
rx_buffer[7]), In_BCD(rx_buffer[8]*10 + rx_buffer[9]));
                flag_Set_Time = 1;
          }
          else return;
     }
     else if(rx_buffer[3] == 'T')
     {
```

```
	}
	else if(rx_buffer[3] == 'S')
	{

	}
}

ISR(USARTRXC_vect)
{
	if(flagEr == 0)
	{
		flagRX = 1;
		UCSRB &= ~(1<<RXCIE);
	}
	//char b = USARTReceiveChar();
	//rx_buffer[i] = b;
	//i++;

	//char status,data;
	//status = UCSRA;
	//data = UDR;
	//if   ((status   &   (FRAMING_ERROR   |   PARITY_ERROR   |
DATA_OVERRUN))==0)
	//{
		//rx_buffer[rx_wr_index++] = data;
		//#if RX_BUFFER_SIZE == 256
		//// special case for receiver buffer size=256
		//if (++rx_counter == 0) rx_buffer_overflow=1;
		//#else
		//if (rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
		//if (++rx_counter == RX_BUFFER_SIZE)
		//{
			//rx_counter = 0;
			//rx_buffer_overflow = 1;
		//}
		//#endif
	//}

}

void USART_Init(unsigned int speed)//Инициализация модуля USART
{
	UBRRH = (unsigned char)(speed>>8);
	UBRRL = (unsigned char)speed;

	UCSRB = (1<<RXEN) | (1<<RXCIE); //Включаем прием и передачу
по USART
	UCSRB |= (1<<RXCIE); //Разрешаем прерывание при передаче
	UCSRA = 0;
	//UCSRA |= (1<<U2X); // Для 8 мгц
```

```
     UCSRC   =   (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);//   Обращаемся
именно к регистру UCSRC (URSEL=1),
     sei();
}

//unsigned char rx_counter = 0;
//
//bit rx_buffer_overflow;
//
//// USART Receiver interrupt service routine
//interrupt [USART_RXC] void usart_rx_isr(void)
//{
////rx_buffer[i++] = USARTReceiveChar();
//unsigned char rx_wr_index = 0, rx_rd_index = 0;
//char status,data;
//status = UCSRA;
//data = UDR;
//if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
     //{
     //rx_buffer[rx_wr_index++] = data;
//#if RX_BUFFER_SIZE == 256
     //// special case for receiver buffer size=256
     //if (++rx_counter == 0) rx_buffer_overflow=1;
//#else
     //if (rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
     //if (++rx_counter == RX_BUFFER_SIZE)
          //{
          //rx_counter = 0;
          //rx_buffer_overflow = 1;
          //}
//#endif
     //}
//}
//#pragma vector = USART_RXC
//__interrupt void usart_rx_isr(void)
//{
//  rx_buffer[i++] = USARTReceiveChar();
//}


#include "I2C.h"

uint8_t twi_status_register;

void i2cInit(void)
{
     TWBR = TWBR_VALUE;
     TWSR = 0;
}

uint8_t i2cread(uint8_t ask)
{
```

```
	int i = 0;
	uint8_t data;
	while(!(TWCR & (1<<TWINT)))
	{
		i++;
		if(i == 300)
		{
			break;
		}
	}

	TWCR = (1<<TWINT)|(1<<TWEN)|((ask & 1) << TWEA);
	while(!(TWCR & (1<<TWINT)))
	{
		i++;
		if(i == 300)
		{
			break;
		}
	}

	data = TWDR;
	twi_status_register = TWSR & TWSR_MASK;

	if ((ask && (twi_status_register != TWI_MRX_DATA_ACK))||(!ask
&& (twi_status_register != TWI_MRX_DATA_NACK)))
	{
		return twi_status_register;
	}
	return data;
}

uint8_t i2cstart(uint8_t address)
{
	int i = 0;
	TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
	while(!(TWCR & (1<<TWINT)))
	{
		i++;
		if(i == 300)
		{
			break;
		}
	}
	twi_status_register = TWSR & TWSR_MASK;
	if	((twi_status_register	!=	TWI_START)	&&
(twi_status_register != TWI_REP_START)){
		return twi_status_register;
	}
	while (!(TWCR & (1<<TWINT)))
	{
		i++;
```

```
          if(i == 300)
          {
               break;
          }
     }

     TWDR = address;
     TWCR = (1<<TWINT) | (1<<TWEN);

     while(!(TWCR & (1<<TWINT)))
     {
          i++;
          if(i == 300)
          {
               break;
          }
     }

     twi_status_register = TWSR & TWSR_MASK;
     if     ((twi_status_register    !=    TWI_MTX_ADR_ACK)    &&
(twi_status_register != TWI_MRX_ADR_ACK)){
          return twi_status_register;
     }
     return 255;
}

uint8_t i2cwrite(uint8_t data)
{
     int i = 0;
     TWDR = data;
     TWCR = (1<<TWINT) | (1<<TWEN);

     while(!(TWCR & (1<<TWINT)))
     {
          i++;
          if(i == 300)
          {
               break;
          }
     }

     twi_status_register = TWSR & TWSR_MASK;
     if (twi_status_register != TW_MT_DATA_ACK)
     {
          return 1;
     }
     else
     {
          return 0;
     }
}
```

```
void i2cstop(void)
{
     int i = 0;
     TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
     while(TWCR & (1<<TWSTO))
     {
          i++;
          if(i == 300)
          {
               break;
          }
     }
}

#include "ports.h"

void init_ports()
{
    // Input/Output Ports initialization
    // Port A initialization
    // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=Out
Bit1=In Bit0=In
    DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3)
| (1<<DDA2) | (0<<DDA1) | (0<<DDA0);
    // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T
Bit0=T
    PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) |
(0<<PORTA3) | (1<<PORTA2) | (0<<PORTA1) | (1<<PORTA0);

    // Port B initialization
    // Function: Bit7=Out Bit6=In Bit5=Out Bit4=Out Bit3=Out
Bit2=Out Bit1=Out Bit0=Out
    DDRB = PortB7 | (~PortB6) | PortB5 | PortB4 | PortB3 | PortB2
| PortB1 | PortB0;
    // State: Bit7=0 Bit6=T Bit5=0 Bit4=0 Bit3=T Bit2=T Bit1=T
Bit0=T
    PORTB &= ~(PortB7 | PortB6 | PortB5 | PortB4 | PortB3 | PortB2
| PortB1 | PortB0);

    // Port C initialization
    // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In
Bit1=In Bit0=In
    DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3)
| (0<<DDC2) | (0<<DDC1) | (0<<DDC0);
    // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T
Bit0=T
    PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) |
(0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

    // Port D initialization
    // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In
Bit1=In Bit0=In
```

```
    DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3)
| (0<<DDD2) | (0<<DDD1) | (0<<DDD0);
    // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T
Bit0=T
    PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) |
(0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);
}

#include "LED_MAX7219.h"

void SPI_WriteStartByte(char data);
void SPI_WriteEndByte(char data);
void SPI_WriteByte(char data);
void SPI_WriteByte(char data);
void SendLed(char adr, char data);
void InitLed()
{
     SPCR=(0<<SPIE) | (1<<SPE) | (0<<DORD) | (1<<MSTR) | (0<<CPOL)
| (0<<CPHA) | (0<<SPR1) | (0<<SPR0);
     SPSR=(0<<SPI2X);

     // инициализация дисплея
     SendLed((DISPLAY_TEST >> 8), (DISPLAY_TEST | 0x00));
     SendLed((INTENSITY >> 8), (INTENSITY | 0x0f));
     SendLed((SCAN_LIMIT >> 8), (SCAN_LIMIT | 0x07));
     SendLed((NO_DECODE_MODE >> 8), (NO_DECODE_MODE | 0x00));
     SendLed((SHUTDOWN >> 8), (SHUTDOWN | 0x00));
}
void SPI_WriteStartByte(char data)
{
     PORTB &= ~(CS);
     SPDR = data;
     while(!(SPSR & (1<<SPIF)));
}

void SPI_WriteEndByte(char data)
{
     SPDR = data;
     while(!(SPSR & (1<<SPIF)));
     PORTB |= (CS);
}

void SPI_WriteByte(char data)
{
     SPDR = data;
     while(!(SPSR & (1<<SPIF)));
}

void SendLed(char adr, char data)
{
     int i = 0;
     while(i < 3)
```

```
     {
          SPI_WriteStartByte(adr);
          SPI_WriteEndByte(data);
          i++;
     }
}
void ClearDisplay()
{
     for(char j = 1; j <= 8; j++)
     {
          SendLed(j,0);
     }
     SendLed((SHUTDOWN >> 8), (SHUTDOWN | 0x01));
}
void WriteNum(char *z, char *y, char *x)
{
     for(int i = 0; i < 8; i++)
     {
          PORTB &= ~(CS);

          SPI_WriteByte(i + 1);
          SPI_WriteByte(pgm_read_byte(&(z[i])));

          SPI_WriteByte(i + 1);
          SPI_WriteByte(pgm_read_byte(&(y[i])));

          SPI_WriteByte(i + 1);
          SPI_WriteByte(pgm_read_byte(&(x[i])));

          PORTB |= CS;
     }
}
void SetIntensity(uint8_t a)  // 0 down to 15
{
     SendLed((INTENSITY >> 8), (SHUTDOWN | a));
}

#include "OLED.h"

uint8_t oled_pointer;

void SetPointer(uint8_t a)
{
     oled_pointer = a;
}
uint8_t GetPointer()
{
     return oled_pointer;
}

void InitOLED()
{
```

```
     i2cstart(SSD1306_ADDR);
     i2cwrite(0x00);

     i2cwrite(SSD1306_DISPLAYOFF);                    // 0xAE
     i2cwrite(SSD1306_SETDISPLAYCLOCKDIV);            // 0xD5
     i2cwrite(0x80);                                       // the
suggested ratio 0x80

     i2cwrite(SSD1306_SETMULTIPLEX);                  // 0xA8
     i2cwrite(SSD1306_LCDHEIGHT - 1);

     i2cwrite(SSD1306_SETDISPLAYOFFSET);              // 0xD3
     i2cwrite(0x0);                                  // no offset
     i2cwrite(SSD1306_SETSTARTLINE | 0x0);            // line #0
     i2cwrite(SSD1306_CHARGEPUMP);                    // 0x8D
     if (0x2 == SSD1306_EXTERNALVCC)
     {
          i2cwrite(0x10);
     }
     else
     {
          i2cwrite(0x14);
     }
     i2cwrite(0x20);                                       // 0x20
SSD1306_MEMORYMODE
     i2cwrite(0x00);                                 //   0x0   act
like ks0108
     i2cwrite(SSD1306_SEGREMAP | 0x1);
     i2cwrite(SSD1306_COMSCANDEC);

     i2cwrite(SSD1306_SETCOMPINS);                    // 0xDA
     i2cwrite(0x02);
     i2cwrite(SSD1306_SETCONTRAST);                   // 0x81
     i2cwrite(0xFF);

     i2cwrite(SSD1306_SETPRECHARGE);                  // 0xd9
     if (0x2 == SSD1306_EXTERNALVCC)
     {
          i2cwrite(0x22);
     }
     else
     {
          i2cwrite(0xF1);
     }
     i2cwrite(SSD1306_SETVCOMDETECT);                 // 0xDB
     i2cwrite(0x40);
     i2cwrite(SSD1306_DISPLAYALLON_RESUME);           // 0xA4
     i2cwrite(SSD1306_NORMALDISPLAY);                 // 0xA6

     i2cwrite(SSD1306_DEACTIVATE_SCROLL);
```

```
     i2cwrite(SSD1306_DISPLAYON);//--turn on oled panel
     i2cstop();
     oled_pointer = 0x00;
}

void  SetPointOLED(uint8_t  Start_Collumn,  uint8_t  End_Collumn,
uint8_t Start_Page, uint8_t End_Page)
{
     i2cstart(SSD1306_ADDR);
     i2cwrite(CODE_COMMAND);
     i2cwrite(SSD1306_COLUMNADDR);
     i2cwrite(Start_Collumn);
     i2cwrite(End_Collumn);
     i2cstop();
     i2cstart(SSD1306_ADDR);
     i2cwrite(CODE_COMMAND);
     i2cwrite(SSD1306_PAGEADDR);
     i2cwrite(Start_Page);
     i2cwrite(End_Page);
     i2cstop();
}

void OLED_Command(uint8_t data)
{
     i2cstart(SSD1306_ADDR);
     i2cwrite(CODE_COMMAND);      // Co = 0, D/C = 0
     i2cwrite(data);
     i2cstop();
}

void ClearOLED()
{
     SetPointOLED(0x00, 0x7F, 0x04, 0x07);

     i2cstart(SSD1306_ADDR);
     i2cwrite(CODE_DATA);

     for(int kk = 0; kk < 4; kk++)
     {
          for(int k = 0; k < 128; k++)
          {
               i2cwrite(0x00);
          }
     }
     i2cstop();
     oled_pointer = 0x00;
}

void SelectDisplay(int i)
{
     switch(i)
     {
```

```
            case 3:
            {
                PORTB |= (PortB0 | PortB1 | PortB2);
                break;
            }
            case 0:
            {
                PORTB |= (PortB0);
                PORTB &= ~(PortB1 | PortB2);
                break;
            }
            case 1:
            {
                PORTB |= (PortB1);
                PORTB &= ~(PortB0 | PortB2);
                break;
            }
            case 2:
            {
                PORTB |= (PortB2);
                PORTB &= ~(PortB0 | PortB1);
                break;
            }
        }
}

void Set_OLED_Image(IMAGE_OLED a, unsigned char *b)
{
    //OLED_Command(SSD1306_DISPLAYOFF);
    SetPointOLED(oled_pointer, oled_pointer + a.long_image - 1,
0x04, (0x04 + a.height_image));
    i2cstart(SSD1306_ADDR);
    i2cwrite(CODE_DATA);
    for(long int kk = 0; kk < a.array_size; kk++)
    {
        i2cwrite(pgm_read_byte(&(b[kk])));
    }
    i2cstop();
}

void Set_OLED_Num(unsigned char *a)
{
    SetPointOLED(oled_pointer,  (oled_pointer  +  0x19),  0x04,
0x07);
    i2cstart(SSD1306_ADDR);
    i2cwrite(CODE_DATA);
    for(int kk = 0; kk < 104; kk++)
    {
        i2cwrite(pgm_read_byte(&(a[kk]))); //LSB  вверху,  MSB
снизу
    }
    i2cstop();
```

```
}

uint8_t* GetNum(int i)
{
     switch(i)
     {
          case 0:
          {
               return zero_logo;
               break;
          }
          case 1:
          {
               return one_logo;
               break;
          }
          case 2:
          {
               return two_logo;
               break;
          }
          case 3:
          {
               return thee_logo;
               break;
          }
          case 4:
          {
               return four_logo;
               break;
          }
          case 5:
          {
               return five_logo;
               break;
          }
          case 6:
          {
               return six_logo;
               break;
          }
          case 7:
          {
               return seven_logo;
               break;
          }
          case 8:
          {
               return eitht_logo;
               break;
          }
          case 9:
```

```
          {
               return nine_logo;
               break;
          }
     }
}

#include "ds1307.h"
uint8_t buffer0, buffer1, buffer2;

void DS1307_Init(uint8_t rs)
{
     buffer0 = 0xFF;
     buffer1 = 0xFF;
     buffer2 = 0xFF;
     rs &= 0x03;
     i2cstart(DS1307_ADDR);
     i2cwrite(DS1307_CONTROL_REGISTER);
     i2cwrite(0x13);
     i2cstop();
     i2cstart(DS1307_ADDR);
     i2cwrite(0x00);
     i2cwrite(0x00);
     i2cstop();
}

// Функция читает байт из внутреннего регистра slave-устройства.
uint8_t DS1307_ReadRegister(uint8_t deviceRegister)
{
     uint8_t boo = 0;
     i2cstart(DS1307_ADDR);
     i2cwrite(deviceRegister);
     i2cstop();
     i2cstart(DS1307_ADDR | 0x01);
     boo = i2cread(0x00);
     i2cstop();
     return boo;
}

void DS1307_SetTime(uint8_t hour, uint8_t minutes)
{
     i2cstart(DS1307_ADDR);
     i2cwrite(DS1307_MINUTES_REGISTER);
     i2cwrite(minutes);
     i2cwrite(hour);
     i2cstop();
}


void GetTime(void)
{
```

```
     uint8_t second = 0x00, minute = 0x00, hour = 0x00, a = 0x00,
b = 0x00;

     second = DS1307_ReadRegister(DS1307_SECONDS_REGISTER);

     a = (second>>4);
     b = second;
     a&= 7;
     b&= 15;
     if(b != buffer0)
     {
          SelectDisplay(0);
          if(b == 0x00 || b == 0x02|| b == 0x04 || b == 0x06||b
== 0x08)
          {
               SetPointer(0x3C);
               Set_OLED_Image(Ncolon_struct, Ncolon_logo);
          }
          else
          {
               SetPointer(0x3C);
               Set_OLED_Image(colon_struct, colon_logo);
          }
          buffer0 = b;
          //SelectDisplay(0);
          //SetPointer(0x5B);
          //Set_OLED_Num(GetNum(b));
          //SetPointer(0x41);
          //Set_OLED_Num(GetNum(a));

          minute = DS1307_ReadRegister(DS1307_MINUTES_REGISTER);

          a = (minute>>4);
          b = minute;
          a&= 7;
          b&= 15;

          //if(b != buffer1)
          //{
               //SelectDisplay(0);
               SetPointer(0x5B);
               Set_OLED_Num(GetNum(b));
               SetPointer(0x41);
               Set_OLED_Num(GetNum(a));
               buffer1 = b;

               hour = DS1307_ReadRegister(DS1307_HOURS_REGISTER);
               a = (hour>>4);
               b = hour;
               a&= 3;
               b&= 15;
               //if(b != buffer2)
```

```
                //{
                    //SetPointer(0x22);
                    Set_OLED_Num(GetNum(b));
                    SetPointer(0x08);
                    Set_OLED_Num(GetNum(a));
                    buffer2 = b;
                //}
            //}
        }
        else
        {
            return;
        }
}

//запись в бсд
uint8_t In_BCD(uint8_t n)
{
    asm volatile(
        "ld   r26,y+"  "\n\t"
        "clr  r30"          "\n\t"
    "bin2bcd0:"             "\n\t"
        "subi r26,10"  "\n\t"
        "brmi bin2bcd1" "\n\t"
        "subi r30,-16" "\n\t"
        "rjmp bin2bcd0" "\n\t"
    "bin2bcd1:"             "\n\t"
        "subi r26,-10" "\n\t"
        "add  r30,r26" "\n\t"
        "ret" "\n\t"
    ::);
}

//чтение из бсд
uint8_t Out_BCD(uint8_t n)
{
    asm volatile(
    "ld   r30,y"        "\n\t"
    "swap r30"              "\n\t"
    "andi r30,0xf"      "\n\t"
    "mov  r26,r30"      "\n\t"
    "lsl  r26"              "\n\t"
    "lsl  r26"              "\n\t"
    "add  r30,r26"      "\n\t"
    "lsl  r30"              "\n\t"
    "ld   r26,y+"       "\n\t"
    "andi r26,0xf"      "\n\t"
    "add  r30,r26"      "\n\t"
    "ret" "\n\t"
    ::);
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Спецификация

ПРИЛОЖЕНИЕ В

(обязательное)

Перечень элементов

ПРИЛОЖЕНИЕ Г

(обязательное)

Ведомость документов