Оглавление

[Введение 3](#_Toc59569460)

[1. Описание технического задания 4](#_Toc59569461)

[2. Обоснование и описание схемы электрической принципиальной 4](#_Toc59569462)

[2.1 Выбор микроконтроллера 4](#_Toc59569463)

[2.2 Выбор источника тактового сигнала 6](#_Toc59569464)

[2.3 Выбор приёмо – передатчика 6](#_Toc59569465)

[2.4 Выбор драйвера двигателя 7](#_Toc59569466)

[2.5 Выбор источника опорного напряжения 8](#_Toc59569467)

[2.6 Выбор гальванической развязки 9](#_Toc59569468)

[2.7 Выбор преобразователей питающего напряжения 10](#_Toc59569469)

[2.8 Выбор разъёмов 12](#_Toc59569470)

[2.9 Выбор конденсаторов 12](#_Toc59569471)

[2.10 Выбор резисторов 13](#_Toc59569472)

[2.11 Выбор индуктивности 13](#_Toc59569473)

[3. Описание и обоснование конструкции 13](#_Toc59569474)

[4. Разработка программного обеспечения модуля управления двигателями приводов подводного манипулятора. 15](#_Toc59569475)

[4.1 Протокол обмена 15](#_Toc59569476)

[4.2 Основные блоки ПО 17](#_Toc59569477)

[4.2.1 Подпрограмма инициализации 17](#_Toc59569478)

[4.2.2 Основной цикл программы START 18](#_Toc59569479)

[4.2.3 Подпрограмма обработки прерывания по окончанию преобразования АЦП (IN\_ADC) 18](#_Toc59569480)

[4.2.4 Подпрограмма обработки данных с АЦП (end\_ADC) 19](#_Toc59569481)

[4.2.5 Подпрограммы обработки внешних прерывай INT1 и INT0(drive1\_fault и drive2\_fault) 19](#_Toc59569482)

[4.2.6 Подпрограмма подготовки данных к отправке (fault\_prepare\_1, fault\_prepare\_2 и current\_state) 20](#_Toc59569483)

[4.2.7 Программа обработки прерывания по опустошению регистра данных UDR (B\_TRANS) 20](#_Toc59569484)

[4.2.8 Программа обработки прерывания по окончанию передачи байта по USART ( TRANdate) 21](#_Toc59569485)

[4.2.9 Подпрограмма обработки прерывания таймера/счётчика Т1 (Time\_OUT) 21](#_Toc59569486)

[4.2.10 Подпрограмма обработки прерывания по приёму байта по USART (REC\_date) 22](#_Toc59569487)

[4.2.11 Подпрограмма обработки команд (read\_info) 23](#_Toc59569488)

[4.3 Листинг программы 23](#_Toc59569489)

# Введение

За последние несколько десятилетий значительно расширился спектр задач в которых применяются различные манипуляторы, и подводная робототехника не стала исключением. Сейчас манипуляторы присутствуют практически на всех подводных аппаратах: от небольших автономных аппаратов осмотрового класса до тяжёлых автономных аппаратов, спроектированных для проведения технических работ на больших глубинах, и обитаемых аппаратов.

Целью данной курсовой работы является проектирование модуля управления двигателями приводов манипулятора подводного аппарата.

# Описание технического задания

Целью данной работы является разработка модуля управления двумя двигателями постоянного тока мощностью 100 Вт приводов манипулятора подводного аппарата. Поскольку данное устройство разрабатывается отдельно от всего робототехнического комплекса, оно должно быть самостоятельным и соединяться с главным микроконтроллером подводной робототехнической системой по интерфейсу RS-485.

В данной работе были разработаны схема электрическая принципиальная, конструкция печатной платы, сборочный чертёж модуля и программное обеспечение для микроконтроллера, используемого в данной схеме.

# Обоснование и описание схемы электрической принципиальной

## Выбор микроконтроллера

Техническим заданием рекомендовано использовать в данной работе микроконтроллеры семейства Atmel AVR.

Для выполнения поставленной задачи необходимы следующий модули в составе микроконтроллера:

1. USART для организации приёмо-передачи по интерфейсу RS-485
2. АЦП для обеспечения обратной связи по положению выходного вала двигателя
3. Таймеры для контроля времени получения посылки

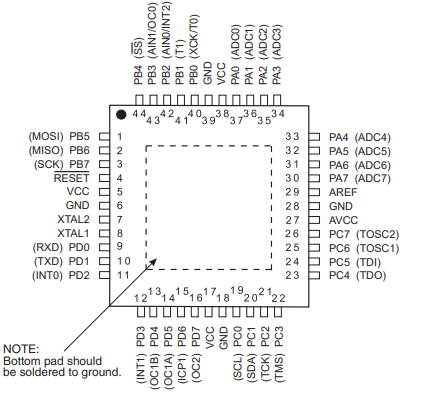
Исходя из этих требований был выбран микроконтроллер ATMEGA16 – 16AU в корпусе TQFP-44 с габаритными размерами 10х10. Исполнение в данном корпусе было выбрано из-за его малых габаритов.

Рисунок 2.1 ATMEGA16 – 16AU в корпусе TQFP-44

Характеристики выбранного микроконтролера:

1. Корпус TQFP-44
2. Ядро AVR
3. Максимальная частота 16 МГц
4. Объём памяти программ
5. Объём EEPROM памяти
6. Объём оперативной памяти
7. Интерфейсы I2C, SPI, USART
8. Десятиразрядный АЦП
9. Напряжение питания от 4.5 до 5.5 В
10. Габаритные размеры 10х10
11. Два восьмиразрядных таймера-счётчика и один шестнадцатиразрядный

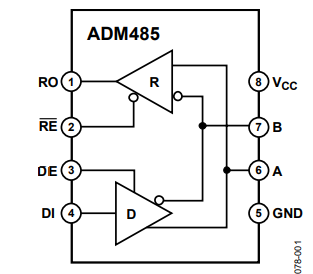
## 2.2 Выбор источника тактового сигнала

В качестве основы для источника тактового сигнала в данной схеме выбран кварцевый резонатор NX5032GA (позиционное обозначение Z) с рабочей частотой 11.0592 МГц. При использовании резонатора такой частоты будет отсутствовать погрешность при делении частоты для асинхронного приемо-передатчика, что будет означать, что можно передавать данные любой длины, не опасаясь за их потерю, вызванную погрешностью частоты тактирования. Выберем скорость работу USART равную 9600 бит/с.

В схему подключения данного элемента входят 2 керамических конденсатора ёмкостью 8пкФ (позиционное обозначение С14 и С16).

## 2.3 Выбор приёмо – передатчика

Для реализации интерфейса RS – 485 была выбрана микросхема ADM485 (позиционное обозначение). ADM485 – это дифференциальный приёмо – передатчик, предназначенный для высокоскоростного двустороннего обмена данными по интерфейсам RS – 485 и RS – 422. В корпусе данной микросхемы присутствует и устройство приёмника, и устройство передатчика, которые работают независимо друг от друга. Структурная схема ADM485 представлена на рисунке 2.2

Рисунок 2.2 Структурная схема ADM585

ADM485 позволяет передавать данные со скоростью до 5МБ/с.

## 2.4 Выбор драйвера двигателя

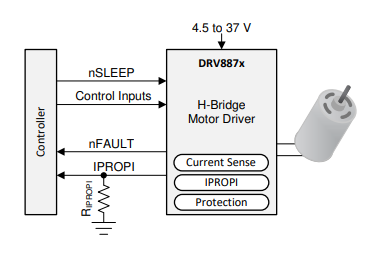
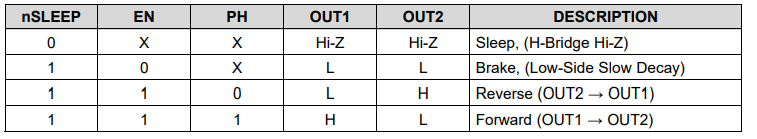
 В качестве драйвера шагового двигателя в данной схеме выбран драйвер для двигателей постоянного тока DRV8874. Микросхема позволяет управлять одним двигателем постоянного тока. Максимальный выходной ток составляет 6 А при максимальном напряжении 37 В. Драйвер имеет простой интерфейс управления и производится в корпусе HTSSOP-16.

Рисунок 2.3 Структурная схема драйвера

Ниже представлена таблица истинности для данной микросхемы.

Таблица 1. Таблица истинности DRV8874

К достоинствам этого драйвера можно отнести и то, что у него есть обратная связь по состоянию драйвера. При перегреве микросхемы или подаче слишком большого напряжения на неё на выходе nFault появится высокий уровень сигнала. К данному драйверу необходимо подключить 4 конденсатора: один танталовый ан 1мкФ (позиционное обозначение на схеме C17, C26) 0805 AL 1uF ±20% 50V 298D105X0050P2T, один керамический на 22пкФ (позиционное обозначение на схеме C20, C31) X7R 0805 22pF ±10% 50V C0805C220K5RAC7800, и два керамических конденсатора на 0.1мкФ (позиционное обозначение на схеме C18, C19, C27, C30) X7R 0.1uF ±20% 50V 08055C104MAT2A.

## 2.5 Выбор источника опорного напряжения

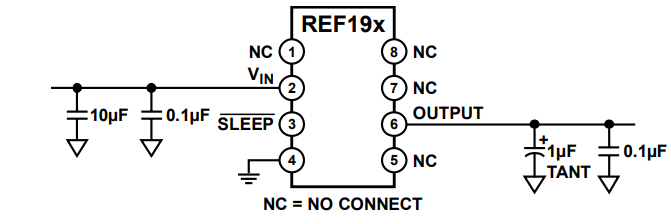
Выбранный микроконтроллер уже имеет внутренний источник опорного напряжения, но к нему нельзя подключить потенциометры. Поэтому была выбрана микросхема REF198 в корпусе SOIC – 8 (позиционное обозначение DD11). Выходное напряжение микросхемы составляет 4.096В, что позволяет избежать погрешности деления напряжения при использовании АЦП. Точность выходного напряжения 2мВ, максимальное входное напряжение 18В. 

Рисунок 2.4 Рекомендуемая схема подключения REF198

В схему подключения микросхемы также входят 4 конденсатора: керамический на 10мкФ (позиционное обозначение на схеме С36) X5R 0805 10uF ±20% 10V LMK212BJ105MD-T и керамический на 0.1 мкФ на входное напряжение (позиционное обозначение на схеме С37) X5R 0805 0.1uF ±10% 6.3V 08056D104KAT2A, танталовый на 1мкФ (позиционное обозначение на схеме С20 и С31) 0805 AL 1uF ±20% 10V TPSR105K010R9000 и керамический на 0.1мкФ (позиционное обозначение на схеме С25 и С39) X5R 0805 0.1uF ±10% 6.3V 08056D104KAT2A на выходное напряжение.

## 2.6 Выбор гальванической развязки

В качестве гальванической развязки между цепью RS-485 и микроконтроллером была выбрана микросхема ADuM1301 в корпусе SOIC-16 (позиционное обозначение DD7). Она позволяет гальванически развязать цепь RS-485 от микроконтроллера и питания платы, что позволяет обезопасить схему от протекания больших токов, так как цепь RS-485 находится вне прочного корпуса и может подвергнуться короткому замыканию при нарушении целостности изоляции.

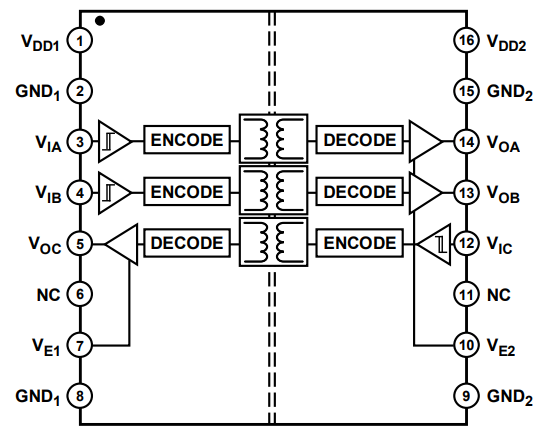


Рисунок 2.5 Структурная схема ADuM1301

Это двунаправленная развязка, что позволяет использовать её при организации асинхронной приёмо – передаче.

Для развязки между цепью двигателей и микроконтроллером была выбрана микросхема ADuM1401 в корпусе SOIC – 16, по аналогичным соображениям.

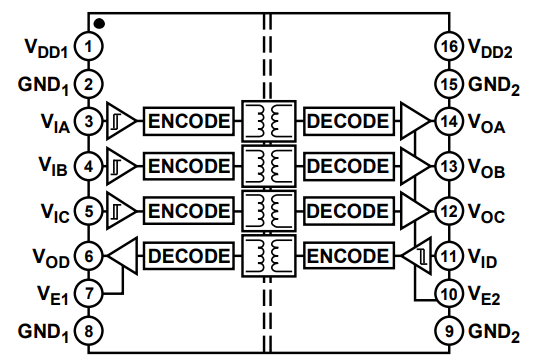


Рисунок 2.6 Структурная схема ADuM1401

ADuM1401 также является двунаправленной развязкой, что позволяет нам организовать обратную связь по состоянию двигателя.

## 2.7 Выбор преобразователей питающего напряжения

Из технического задания следует, что входное напряжение питания платы +27В. Так как для корректной работы микросхем на плате необходимо напряжение питания ниже, то необходимо использование DC – DC преобразователей с гальванической развязкой для понижения напряжения питания.

Понижающий DC – DC преобразователь с гальванической развязкой необходим для того, чтобы понизить входное напряжение до необходимого уровня и отвязать одну цепь с высоким напряжением от цепи м низким напряжением.

Всего в цепи присутствуют 3 цепи питания:

1. Цепь питания +5В для микроконтроллера ATMEGA16-16AU, источника опорного напряжения REF198 и 3 гальванических развязок: ADuM 1301 и ADuM 1401. Микроконтроллер потребляет 200мА, источник опорного напряжения 45мА, а каждая гальваническая развязка по 11мА. Для этой цепи выберем понижающий DC – DC преобразователь на 2 Вт c гальванической развязкой TDR-2-2411SM.
2. Цепь питания +5В для приёмо-передатчика ADM485 и гальванической развязки ADuM1301. Приёмо – передатчик потребляет 50мА, а гальваническая развязка потребляет 11 мА. Для этой цепи выберем понижающий DC – DC преобразователь на 1 Вт c гальванической развязкой TMR-1-2411SM.
3. Цепь питания +5В для питания двух драйверов двигателей DRV8874 и двух гальванических развязок ADuM1401. Драйверы потребляет 75мА, а гальванические развязки по 11мА. Для этой цепи выберем понижающий DC – DC преобразователь на 1 Вт c гальванической развязкой TMR-1-2411SM.

Для подключения DC – DC преобразователей выберем следующие элементы: электролитические конденсаторы (позиционные обозначения С3, С7, С11) ALUM 100uF ±20% 50V EEE-FTH101XAL, керамические конденсаторы X5R 0805 10uF ±20% 50V CGRM21BR61H106KE43L (позиционное обозначение на схеме С1, С5, С9), X5R 0805 10uF ±20% 10V LMK212BJ105MD-T (позиционное обозначение на схеме С4, С8, С12), X5R 0805 0.1uF ±10% 6.3V 08056D104KAT2A (позиционное обозначение на схеме С2, С6, С10).

## 2.8 Выбор разъёмов

Для выполнения данного проекта нам необходимо несколько видов разъёмов.

Разъём PLS-2 (позиционное обозначение ХР) служит для подключения к плате источника питания на +27В. Разъём имеет 2 контакта, номинальное напряжение 250В, номинальный ток 3А, шаг контактов 2.54 мм.

Для подключения к главному микроконтроллеру служит разъём PLS-2 (позиционное обозначение ХР6).

Разъём PLS-6 (позиционное обозначение ХР2) служит для подключения двух потенциометров. Разъём имеет 6 контактов, номинальное напряжение 250В, номинальный ток 3А и шаг контактов 2.54 мм.

Разъём IDC-6 (позиционное обозначение ХР5) необходим для подключения программатора AVR-ISP500. Разъём имеет 6 контактов, номинальное напряжение 250В, максимальный ток 3А и шаг контактов 2.54мм.

Для подключения двигателей были выбраны 2 разъёма 15EDGVC-3.5-02 (позиционное обозначение ХР3 и ХР4). Разъём имеет 2 контакта, номинальное напряжение 250В, номинальный ток 7А и шаг контактов 3.5мм. Эти разъёмы могут быть как прямыми, так и угловыми. Для данной платы был выбран прямой разъём

## 2.9 Выбор конденсаторов

Для выполнения поставленной задачи необходимо несколько номиналов конденсаторов 0.1мкФ и 100мкФ. Были выбраны резисторы ALUM 100uF ±20% 50V UEEE-FTH101XAP, X5R 0805 0.1uF ±10% 6.3V 08056D104KAT2A и X5R 0805 10uF ±20% 50V CGRM21BR61H106KE43L.

## 2.10 Выбор резисторов

Для выполнения поставленной задачи необходимы резисторы номиналом 120 Ом. Были выбраны резисторы 0805 120 Ohms ±5% 0.125W RMCF0805JT120R.

## 2.11 Выбор индуктивности

Для выполнения поставленной задачи необходима индуктивность номиналом 10 мкГн.

Была выбрана индуктивность 0603AF-103XJEW COILCRAFT 10µH Unshielded Inductor 180mA 4.5Ohm Max 0603.

# 3. Описание и обоснование конструкции

По техническому заданию необходимо спроектировать плату управления двигателями приводов подводного робота. За основу была взята прямоугольная плата габаритами 80х75. По краям платы расположены отверстия диаметром 3.3мм под винт М3, которые расположены на расстоянии 5 мм от края платы.

Прежде чем начинать трассировку платы, необходимо рационально скомпоновать элементы. Среди критериев рациональной компоновки можно отметить:

* Группировка отдельно аналоговых элементов от цифровых;
* Минимизация длины печатных проводников;
* Разбиение схемы на отдельные функциональные блоки, при этом элементы, входящие в блоки, на плате располагаются рядом друг с другом;
* Учёт электрофизических особенностей электронных устройств (сюда можно включить расположение фильтрующих конденсаторов и индуктивностей и т.д.).

Плата была разбита на 4 функциональных модуля:

1. Модуль питания, в котором располагаются 3 DC – DC преобразователя и разъём для подключения к источнику питания;
2. Модуль, в котором распложён микроконтроллер, источник опорного напряжения, разъём для прошивки микроконтроллера и разъём для подключения потенциометров;
3. Модуль приёмо –передатчика, в котором располагается разъём для подключения к центральному микроконтроллеру и сам приёмо – передатчик;
4. Модуль, в котором находятся драйвера двигателей и разъёмы для подключения двигателей.

На границах 2, 3 и 4 модуля расположены гальванические развязки.

Резисторы и конденсаторы преимущественно расположены в линейку, почти вплотную друг другу, так как данные элементы паяются с торцов. Конденсаторы располагаются как можно ближе к микросхемам, к цепям которых они подключены.

Дорожки цепи питания +27В проведены с толщиной 1.5мм, переходные отверстия этой же цепи сделаны с диаметром 1мм. Для цепей питания +5В использованы дорожки с толщиной от 1 до 0.6 мм. Цепи же сигнальных проводников проведены с шириной от 0.6 до 0.3 мм.

На нижнем слое платы находятся 4 полигона, каждый из которых соответствует «общему» своего функционального модуля.

# Разработка программного обеспечения модуля управления двигателями приводов подводного манипулятора.

Программное обеспечение разрабатывается в специальной среде AVR Studio версии 4.1 на языке программирования AVR Assembler. Программное обеспечение необходимо для микроконтроллера, чтобы он успешно реализовал свои функции. Основные блоки программного обеспечения показаны на структурной схеме программы.

## Протокол обмена

В подавляющем большинстве протоколов обмена между устройствами применяются контрольные суммы для контроля сохранности данных для передачи. Так же протокол, использованный в этом проекте – адресный, что означает, что первый байт сообщения будет содержать в себе адрес устройства. В случае если адрес устройства не совпадает с адресом в сообщении – сообщение игнорируется. Члены сети данного протокола имеют разный вес, в данном случае центральный микроконтроллер – master (ведущий) и микроконтроллер на плате – slave (ведомый). Ведущий инициирует обмен и решает, кто может занимать шину в текущий момент времени. В нашем случае наша плата – ведомая.

Таким образом сообщение выглядит так:

Где – первые байты сообщения, полезная информация;

A - адрес, C – контрольная сумма (по одному байту)

Суммирование выполняется по всем байтам полезной информации без учёта переполнения командой ADD.

Ведущий контроллер передает 3 байта следующей информации:

0-й байт – адресный байт;

1-й байт – байт команды;

2-й байт – байт контрольной суммы.

В байте команды содержится номер команды:

* 0х01 – drive1\_forward;
* 0x02 – drive1\_reverse;
* 0x03 – drive2\_ forward;
* 0x04 – drive2\_ reverse;
* 0x05 – all\_stop;
* 0x06 – drive1\_stop;
* 0x07 – drive2\_stop;
* 0x08 – current\_state.

Ответ (квитанция) контроллера ведущему состоит из пяти байт:

0-й байт – адресный байт;

1-й байт – положения вала первого двигателя;

2-й байт – положения вала второго двигателя;

3-й байт – байт состояния двигателя;

4-й байт – байт контрольной суммы.

В байте состояния двигателя находится значение 0x0F если перегрелся первый драйвер, и 0x70 если перегрелся второй драйвер.

## Основные блоки ПО

## Подпрограмма инициализации

Начинается подпрограмма инициализации с инициализации стека. Записываем младший байт константы RAMEND в регистр SPL, а старший в регистр SPH. Это установит указатель стека в начало ОЗУ.

Далее идёт инициализация портов ввода/вывода. PORTD будет использован для приёмо – передачи через USART. PD0(RXD) инициализируется как вход, PD1(TXD) и PD4 инициализируются как выходы. На PD4 устанавливается 0, что сигнализирует о работе микроконтроллера в режиме приёма. PD2(INT0) и PD3(INT1) инициализируем как входы.

PORTC будет использован для подачи команд управления на драйверы двигателей. PC1 … PC3 и PC4 … PC7 будут использованы как выходы. Подаём на все выходы 0, чтобы перевести двигатели в спящий режим.

Далее идёт подпрограмма инициализации USART. Частота внешнего источника тактового сигнала 11,0592 МГц, скорость обмена данными 9600 бод, бит UX20 установлен в 0, поэтому загружаем значение 77 в регистр UBRR. Устанавливаем формат посылки 8 бит.

Далее идёт подпрограмма инициализации АЦП. Выбираем вход АЦП PA0, загружая значение 0 в регистр DDRA. Устанавливая биты REFS1, REFS0 в 0, указываем на использование внешнего источника опорного напряжения. Устанавливаем частоту работы АЦП равную 125кГц.

Далее идёт подпрограмма инициализации таймера/счётчика T1. Таймер/счётчик необходим для проверки корректности прихода посылки по USART. Таймер счётчик тактируется от внешнего источника тактовых импульсов. Выставляем в 0 биты COM1A1 и COM1A0, чтобы отключить таймер счётчик от входа ОС1А. Выставляем время работы таймера равное 6мс, выставляя значение Е808 в регистрах OC1AH, OC1AL.

Далее идёт подпрограмма инициализации внешних прерываний INT0 и INT1. Разрешаем эти прерывания и устанавливаем, что они будут проходить по спаду сигнала.

Далее следуют подпрограммы очистки SRAM. В программе будут задействованы 2 буфера SRAM. Первый - для принимаемой посылки, второй для ответной посылки. Далее выставляется единица в бит I регистра SREG. Данный бит разрешает прерывания в программе.

## 4.2.2 Основной цикл программы START

Основной цикл начинается с включения АЦП, вызывается подпрограмма Start\_ADC, в ней устанавливаем 1 в бит ADEN, включая АЦП и в бит ADIE, разрешая прерывание по окончанию преобразования АЦП.

Далее программа ожидает посылки. Если посылка пришла корректно, то устанавливается флаг F\_receive. Если флаг не установлен, переходим к подпрограмме stop\_all, которая останавливает двигатели и переводит их в режим низкого энергопотребления. Если же флаг установлен, то мы переходим в подпрограмму обработки команды read\_info, а из неё переходим в нужную подпрограмму выполнения команды. Далее идёт подготовка посылки к отправке. Как только посылка будет готова к отправке будет выставлен флаг F, поэтому дальше идёт его проверка. Если флаг не выставлен, то возвращаемся в цикл. Если флаг выставлен – переходим к отправке. Если флаг окончания установлен, то переходим к ожиданию новой посылки. Если не установлен, то переходим к подпрограмме stop\_all, которая останавливает двигатели и переводит их в режим низкого энергопотребления.

## 4.2.3 Подпрограмма обработки прерывания по окончанию преобразования АЦП (IN\_ADC)

В программе используются 2 канала АЦП для преобразования аналогового сигнала с 2 потенциометров, которые измеряют угловое положение выходных валов двигателей. В начале подпрограммы сохраняем РОНы temp\_L, temp\_H и регистр SREG в стек. Далее данные из регистра ADCH суммируются со значениями в регистрах ADC\_H, ADC\_L. Далее инкрементируем счётчик количества преобразований cou\_ADC и сравниваем его с количеством преобразований, если это преобразование четвёртое, то происходит переход в подпрограмму окончания цикла преобразований end\_ADC, в этой подпрограмме мы обнуляем счётчик cou\_ADC и запуск подпрограммы обработки данных с АЦП. Если значение счётчика ещё не равно 4, то начинаем новое преобразование. Каналы меняются после того, как АЦП совершит 4 преобразования для одного сигнала.

## 4.2.4 Подпрограмма обработки данных с АЦП (end\_ADC)

В начале подпрограммы проверяем номер канала, с которого приходили показания АЦП. Если номер канала number = 0 (1), то переходим в подпрограмму first, которая соответствует потенциометру, установленному на выходном валу первого двигателя. Далее получаем среднее значение 4 преобразований делением на 4. После этого происходит проверка среднего на крайние положения. Если было достигнуто крайнее левое или крайнее правое положение, то останавливаем первый двигатель, переходя в под программу drive1\_stop (drive2\_stop).

Если номер канала number = 1, то переходим в подпрограмму second, которая соответствует потенциометру, установленному на выходном валу второго двигателя. В подпрограмме second происходят действия аналогичные тем, что происходят в подпрограмме first.

Далее следует очистка регистров суммы ADC\_L и ADC\_H, смена канала чтения данных и выход из подпрограммы.

## 4.2.5 Подпрограммы обработки внешних прерывай INT1 и INT0(drive1\_fault и drive2\_fault)

Эти две подпрограммы аналогичны друг другу, за исключением того, что вызываются сигналами от разных драйверов.

В начале подпрограммы сохраняем РОНы temp\_L, temp\_H и регистр SREG в стек. Далее останавливаем нужный двигатель подавая на выходы нужного двигателя 0 (PC1 …PC3 для первого двигателя и PC5 … PC7 для второго). Далее вызывается подпрограмма подготовки ответа (fault\_prepare\_1 или fault\_prepare\_2) для загрузки данных посылки в буфер varBuf\_TXD. Подробнее про работу этой программы будет написано ниже. После возвращения из подпрограммы подготовки ответа, происходит установка флага отправки F\_answer. Далее разрешатся прерывание UDRIE и происходит выход из подпрограммы обработки прерываний.

## 4.2.6 Подпрограмма подготовки данных к отправке (fault\_prepare\_1, fault\_prepare\_2 и current\_state)

В данных подпрограммах происходит подготовка данных посылки на центральный микроконтроллер в буфере varBuf\_TXD.

В начале программ происходит перемещение указателя Z на начало буфера varBuf\_TXD. Далее очищаем регистр для хранения контрольной суммы c\_sum\_tran. После этого начинается последовательная запись каждого байта информации в буфер с перемещением указателя Y на последующий байт. Разница между тремя подпрограммами в записи 3 байта информации в буфер. Подпрограмма current\_state записывает в него 0х00, а подпрограммы fault\_prepare\_1 и fault\_prepare\_2 0х0d и 0x7c соответственно. После записи всех данных посылки в буфер происходит выход из этой подпрограммы.

## 4.2.7 Программа обработки прерывания по опустошению регистра данных UDR (B\_TRANS)

В данной подпрограмме обрабатываются 2 прерывания:

* Прерывание по UDRIE (прерывание по событию регистр данных USART пуст);
* Пребывание по TXCIE (прерывание по завершению передачи байта).

В начале подпрограммы сохраняем РОНы temp\_L, temp\_H и регистр SREG в стек. Первый байт данных будет передан по прерыванию UDRIE Далее идёт разрешение прерыванию по TXCIE, которое разрешается следующим шагом программы. После этого устанавливаем флаг F\_receive в 0 и происходит выход из подпрограммы обработки прерывания.

## 4.2.8 Программа обработки прерывания по окончанию передачи байта по USART ( TRANdate)

В начале подпрограммы сохраняем РОНы temp\_L, temp\_H и регистр SREG в стек. Далее инкрементируем счётчик отправленных байт cou\_Tran и проверка cou\_Tran = val\_TR, где val\_TR – количество байт в передаваемой посылке. Если значение счётчика переданных байт меньше количества байт в передаваемой посылке, то загружаем новый байт информации из буфера var\_bufTXD в регистр UDR и сдвигаем указатель Y на следующий байт. После этого происходит выход из подпрограммы

Если же условие cou\_Tran = val\_TR выполняется, то очищаем счётчик cou\_Tran, и переходим в подпрограмму окончания передачи end\_tr. В ней переключаем USART на приём, устанавливаем указатель Y на начало буфера приёма данных varBuf\_RXD и устанавливаем флаг окончания передачи F\_trans. Следующим действием возвращаемся из подпрограммы обработки прерывания.

## 4.2.9 Подпрограмма обработки прерывания таймера/счётчика Т1 (Time\_OUT)

Данный таймер/счётчик отвечает за корректность приёма посылки по USART. Если приём посылки длится дольше 6 мсек, то происходит переход по прерыванию OCIE1A в подпрограмму обработки прерывания таймера/счётчика Т1 TimaOUT.

В начале подпрограммы сохраняем РОНы temp\_L, temp\_H и регистр SREG в стек. Далее происходит остановка таймера/счётчика Т1, запрет прерывания OCIE1A. После этого происходит установка указателя Y в начало буфера приёма var\_bufRXD, очистка РОНов c\_sumREC и Cou\_REC, и выход из подпрограммы обработки прерываеия.

## 4.2.10 Подпрограмма обработки прерывания по приёму байта по USART (REC\_date)

В начале подпрограммы сохраняем РОНы temp\_L, temp\_H и регистр SREG в стек. Переход по прерыванию RXCIE происходит тогда, когда в буфере появляются непрочитанные байты посылки. Далее загружаем данные из регистров UDR, UCSRA в templ\_L, templ\_H соответственно. После происходит проверка данных регистра UCSRA на наличие ошибок с помощью проверки флагов FE, DOR, PE. Если ошибки есть, то происходит выход из подпрограммы обработки прерывания. Если ошибок нет, то происходит переход в подпрограмму USART2NoError. В ней инкрементируем счётчик полученных байт и сохраняем данные из регистра UDR в регистр приёма. Если это первый адресный байт посылки, то совершается переход в подпрограмму rec\_1\_b, в которой происходит проверка совпадения значения адресного байта посылки и константы right\_adres, в которой хранится правильный адрес. Если пришёл неправильный адрес, то происходит переход в подпрограмму error\_addres, в которой происходит очистка счётчика cou\_Rec, установка указателя Y на начало буфера varBuf\_RXD и выход из подпрограммы обработки прерывания. Если пришёл верный адрес, то устанавливаем флаг MPCM, что говорит о начале приёма байт посылки по USART. Далее суммируем адресный байт со значением контрольной суммы c\_sumREc и запускаем таймер Т1. Далее идёт выход из подпрограммы обработки прерывания и ожидание получения следующего байта посылки, посылка не придёт за 6 мс, то будет вызвано прерывание таймера Т1. При приходе следующего байта посылки, после входа в подпрограмму обработки прерывания, проверки на ошибки и проверки на 1 байт происходит проверка на последний байт. Если пришедший байт не последний, то мы инкрементируем счётчик принятых байт cou\_Rec и складываем значение принятого байта с значением c\_sumRec. Если же пришедший байт последний, то вызывается подпрограмма rec\_end. В ней происходит инверсия принятого байта и проверка контрольной суммы. Если контрольная сумма посчитана правильно, то выставляется флаг правильного приёма данных, после останавливается таймер Т1, указатель Y устанавливается на начало varBuf\_RXD, флаг MPCM устанавливается в 1, сброс счётчика cou\_Rec и контрольной суммы c\_sumREC и выход из подпрограммы обработки прерывания.

## 4.2.11 Подпрограмма обработки команд (read\_info)

В принимаемой посылке, в зависимости значения байта команды, могут содержаться 6 команд:

* 0х01 – drive1\_forward;
* 0x02 – drive1\_reverse;
* 0x03 – drive2\_ forward;
* 0x04 – drive2\_ reverse;
* 0x05 – all\_stop;
* 0x06 – drive1\_stop;
* 0x07 – drive2\_stop;
* 0x08 – current\_state.

В начале подпрограммы мы выгружаем байт с информацией из varDuf\_RXD. Дальше, если это байт команды, для каждой команды вызывается соответствующая под программа, которая выставляет необходимые значения на порты PC1 …PC3 для первого двигателя и PC5 … PC7 для второго.

## 4.3 Листинг программы

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Designer Shestakov A. A.

;\* Version: 1.3

;\* Date 26.11.2020

;\* Device ATmega16

;\* Clock frequency:Частота кв.резонатора 11.0592 mHz

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.include "m16def.inc"

; присоединение файла описаний;

.list ;включение листинга

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Register Variables

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.def temp\_L =R16

.def temp\_H =R17

;

.def res\_a\_op =R8;

.def Time =R25;счетчик переполн.Т0(1024\*255/8000000\*Xotc=1cek(Xotc=30)

;

.def Cou\_Rec =R4;счетчик принятых байт

.def Cou\_Tran =R5;счетчик переданных байт

.def c\_sumREC =R6;контр сумма прин. байт

.def c\_sum\_tran =R7;контр сумма переданных байт

;

.def cou\_ADC =R25

.def ADC\_h =R3

.def ADC\_l =R4

.def val\_n\_adc =R9

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.def Byte\_fl =R2; байт флагов

;-------------------

.equ F\_receive =2;флаг принятого запроса

.equ F\_trans =3;флаг завершения передачи

.equ F\_end\_ADC =1;флаг завершения (Val\_N\_ADC) преобразований АЦП

.equ F\_driver = 0; номер двигателя

.equ Val\_N\_ADC =4;4;колич-во преобразований АЦП

.equ zero\_position =0 ; крайние положения

.equ full\_position =0xFF ; крайние положения

.equ right\_adres = 0x01 ; проверка адреса мултикомплекс. обменя

;Количество байт обмена с ПЭ�'М

;----------------------------------------

.equ VAL\_TR =5

.equ VAL\_REC =3

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Variable

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.DSEG

;

varBuf\_Rxd: .BYTE 8;буфер приема(3 байта)

;

varBuf\_Txd: .BYTE 8;буфер передачи (3 байта)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.cseg

.org $0000

rjmp Init

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.org INT0addr;=$002 ;External Interrupt0 Vector Address

rjmp drive2\_fault

.org INT1addr;=$004 ;External Interrupt1 Vector Address

rjmp drive1\_fault

.org OC2addr; =$006 ;Output Compare2 Interrupt Vector Address

reti

.org OVF2addr;=$008 ;Overflow2 Interrupt Vector Address

reti

.org ICP1addr;=$00A ;Input Capture1 Interrupt Vector Address

; reti

.org OC1Aaddr;=$00C ;Output Compare1A Interrupt Vector Address

rjmp Time\_OUT

.org OC1Baddr;=$00E ;Output Compare1B Interrupt Vector Address

reti

.org OVF1addr;=$010 ;Overflow1 Interrupt Vector Address

reti

.org OVF0addr;=$012 ;Overflow0 Interrupt Vector Address

rjmp time\_d\_k ;

.org SPIaddr; =$014 ;SPI Interrupt Vector Address

reti

.org URXCaddr;=$016 ;UART Receive Complete Interrupt Vector Address

rjmp REC\_date

.org UDREaddr;=$018 ;UART Data Register Empty Interrupt Vector Address

rjmp B\_TRANS

.org UTXCaddr; =$01A ;UART Transmit Complete Interrupt Vector Address

rjmp TRANdate

.org ADCCaddr; =$01C ;ADC Interrupt Vector Address

rjmp IN\_ADC

.org ERDYaddr; =$01E ;EEPROM Interrupt Vector Address

reti

.org ACIaddr; =$020 ;Analog Comparator Interrupt Vector Address

reti

.org TWIaddr; =$022 ;Irq. vector address for Two-Wire Interface

reti

.org INT2addr; =$024 ;External Interrupt2 Vector Address

reti

.org OC0addr; =$026 ;Output Compare0 Interrupt Vector Address

reti

.org SPMRaddr; =$028 ;Store Program Memory Ready Interrupt Vector Address

reti

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Start Of Main Program

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Init:

ldi temp\_L,LOW(RAMEND);выбор вершины стека

out SPL, temp\_L;Указатель стека

ldi temp\_H,HIGH(RAMEND)

out SPH, temp\_H

;�нициализация портов

; ldi temp\_L,0b11111111;(PB1-PB7)-выходы,

; out DDRB,temp\_L

;

ldi temp\_L,0c11101110;(PC1-PC3, PC5-PC7)-выходы, (PC0, PC4)-входы

out DDRB,temp\_L

ldi temp\_L,0c00000000;переводим двигатели в спящий режим

out PORTB,temp\_L;

;

ldi temp\_L,0b00010010;PD1, PD4 - выход, PD0 вход

out DDRD,temp\_L

;INIT USART

ldi temp\_L,71;(Частота кв. 11.0592 м�"ц,скорость обмена 9600 бит|c),U2X0=0,

ldi temp\_H,00

out UBRRL,temp\_L;

out UBRRH,temp\_H

ldi temp\_L,(1<<RXEN)|(1<<RXCIE);UCSZ2=0,UCSZ1=1,UCSZ0=1 - 8 bit

out UCSRB,temp\_L

ldi temp\_L,(1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1);- 1 stop bit,

out UCSRC,temp\_L;

;Analog-to-digital

ldi temp\_L,0x00

out DDRA,temp\_L; входы АЦП

;

ldi temp\_L,(0<<REFS1)|(0<<REFS0)|(1 << MUX0); �спользуем внешний �ОН на 4.096�'

out ADMUX,temp\_L

;

ldi temp\_L,(1<<ADPS2)|(1<<ADPS1); Устанавливаем частоту работы АЦП 125 к�"ц

out ADCSR,temp\_L

;

clr byte\_fl

clr cou\_ADC

;

clr ADC\_h

clr ADC\_l

;�нициализация таймера TCNT1

ldi temp\_L,00;compare A,(COM1A1,COM1A0=00) OC1A disconnect

out TCCR1A,temp\_L;

ldi temp\_L,(1<<WGM12);WGM13=0,WGM12=1,WGM11=0,WGM10=0,режим CTC

; No clock source,CS42,CS41,CS40=000

out TCCR1B,temp\_L;(No prescaling CS10=1

ldi temp\_H,0xD8 ; time\_out 6mcek - 0,006\*9216000=55296(D800)

ldi temp\_L,0x00 ;

out OCR1AH,temp\_H

out OCR1AL,temp\_L

; Enable external interrupts

ldi temp\_L, (1 << INT0)|(1 << INT1)

out GICR, temp\_L

; Init INT0

ldi temp\_L, (0 << ISC00)|(1 << ISC01); прерывание будет происходить по спаду

out MCUCR, temp\_L

; Init INT1

ldi temp\_L, (0 << ISC10)|(1 << ISC11); прерывание будет происходить по спаду

out MCUCR, temp\_L

; Обнуление SRAM

ld\_b\_r: st Y+,temp\_L;начальная установка буфер приема

dec temp\_H

cpi temp\_H,0x00

brne ld\_b\_r

;

ldi YL,low(varBuf\_Txd); Load Y register low буфер передачи

ldi YH,high(varBuf\_Txd) ; Load Y register high буфер передачи

ldi temp\_L,0x00

ldi temp\_H,VAL\_TR

ld\_b\_t: st Y+,temp\_L;начальная установка буфер передачи

dec temp\_H

cpi temp\_H,0x00

brne ld\_b\_t

;

ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема

ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема

;

sei ;разрешаем прерывания

clr Byte\_fl

clr Cou\_Rec

clr Cou\_Tran

clr c\_sumREC

clr ADC\_h;

clr ADC\_l

ldi eight,Val\_N\_ADC

clr position\_1

clr position\_2

;==================================================

;начало цикла программы

;==================================================

Start:

; TODO write main logic

Start: rcall start\_ADC

wait: sbrs Byte\_fl,F\_receive;проверка флага принятого запроса

rjmp wait

rcall read\_info

sbrs Byte\_fl,F\_answer

rjmp wait

clt

bld Byte\_fl,F\_answer

Wait\_tr: sbrs Byte\_fl,F\_trans

rjmp Wait\_tr

clt

bld Byte\_fl,F\_trans

rjmp wait

;==================================================

; конец цикла программы

;==================================================

; =================================================

; �-апуск преобразования АЦП

;==================================================

start\_ADC: ldi temp\_L,(1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1);ч-та преобр.64(125к�"ц)

out ADCSR,temp\_L

ldi temp\_L,(1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADSC);

out ADCSR,temp\_L

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; pre\_date:;Подпрограмма подготовки данных к передаче

; ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; ldi YL,low(varBuf\_Txd); Load Y register low буфер передачи

; ldi YH,high(varBuf\_Txd) ; Load Y register high буфер передачи

; clr c\_sum\_tran

; add c\_sum\_tran,temp\_H

; st Y+,temp\_H

; add c\_sum\_tran,temp\_L

; st Y+,temp\_L

; com c\_sum\_tran

; st Y+,c\_sum\_tran

; ret

; ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; подпрограмма обработки данных, принятых на МК и выполнение действий

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

read\_info:

ldd temp\_L,Y+1

cpi temp\_L,0x01

breq drive1\_forward

cpi temp\_L,0x02

breq drive2\_forward

cpi temp\_L,0x03

breq drive1\_reverse

cpi temp\_L,0x04

breq drive2\_reverse

cpi temp\_L,0x05

breq all\_stop

cpi temp\_L,0x06

breq current\_state

ret

; ============================================

drive1\_forwar:; Подпрограмма вращения двигателя 1 в прямом направлении

; ============================================

ldi temp\_L, 0c00001110

out PORTC, temp\_L

rjmp ex\_out

; ============================================

drive1\_reverse:;Подпрограмма вращения двигателя 2 в прямом направлении

ldi temp\_L, 0c1110000

out PORTC, temp\_L

rjmp ex\_out

; ============================================

; ============================================

drive2\_forward:;Подпрограмма вращения двигателя 1 в обратном направлении

; ============================================

ldi temp\_L, 0c1010000

out PORTC, temp\_L

rjmp ex\_out

; ============================================

drive2\_reverse:;Подпрограмма вращения двигателя 1 в обратном направлении

; ============================================

ldi temp\_L, 0c0001010

out PORTC, temp\_L

rjmp ex\_out

; ===========================================

drivers\_stop:

; ===========================================

ldi temp\_L, 0c0000000

out PORTC, temp\_L

rjmp ex\_out

; ===========================================

current\_state:

; ===========================================

ldi YL,low(varBuf\_Txd);буфер передаваемых данных (5 байт):

ldi YH,high(varBuf\_Txd)

clr c\_sum\_tran

ldi temp\_L,0x12

st Y+,temp\_L ; first bite - address

add c\_sum\_tran,temp\_L

mov temp\_L,position\_1

st Y+,temp\_L ; second bite - position first gain

add c\_sum\_tran,temp\_L

mov temp\_L,position\_2

st Y+,temp\_L ; third bite - position second gain

add c\_sum\_tran,temp\_L

ldi temp\_L,0x00

st Y+,temp\_L ; fourth bite - nfault

add c\_sum\_tran,temp\_L

com c\_sum\_tran

mov temp\_L,c\_sum\_tran

st Y+,temp\_L ; fiveth bite - c\_sum

exit\_out:

clt

bld Byte\_fl,F\_receive

ret

; ============================================

; TODO write this shit

; ============================================

drive2\_fault:;прерывание INT0

push temp\_L

push temp\_H

in temp\_L,SREG

push temp\_L

ldi temp\_L, 0c0000000

out PORTC, temp\_L

rjmp fault\_prepare\_2

ldi temp\_L,(1<<TXEN)|(1<<UDRIE);UCSZ2=0,UCSZ1=1,UCSZ0=1 - 8 bit

out UCSRB,temp\_L;переход к передаче через вызов прерывания UDRE

set

bld Byte\_fl,F\_answer

reti

; ============================================

drive1\_fault:;прерывание INT1

push temp\_L

push temp\_H

in temp\_L,SREG

push temp\_L

ldi temp\_L, 0c0000000

out PORTC, temp\_L

rjmp fault\_prepare\_1

ldi temp\_L,(1<<TXEN)|(1<<UDRIE);UCSZ2=0,UCSZ1=1,UCSZ0=1 - 8 bit

out UCSRB,temp\_L;переход к передаче через вызов прерывания UDRE

set

bld Byte\_fl,F\_answer

reti

; ============================================

fault\_prepare\_1:

ldi YL,low(varBuf\_Txd);буфер передаваемых данных (5 байт):

ldi YH,high(varBuf\_Txd)

clr c\_sum\_tran

ldi temp\_L,0x0d; TODO this some shit, that i need understand

st Y+,temp\_L ; first bite - address

add c\_sum\_tran,temp\_L

mov temp\_L,position\_1

st Y+,temp\_L ; second bite - position first gain

add c\_sum\_tran,temp\_L

mov temp\_L,position\_2

st Y+,temp\_L ; third bite - position second gain

add c\_sum\_tran,temp\_L

ldi temp\_L,F\_driver

st Y+,temp\_L ; fourth bite - nfault

add c\_sum\_tran,temp\_L

com c\_sum\_tran

mov temp\_L,c\_sum\_tran

st Y+,temp\_L ; fiveth bite - c\_sum

ret

fault\_prepare\_2:

ldi YL,low(varBuf\_Txd);буфер передаваемых данных (5 байт):

ldi YH,high(varBuf\_Txd)

clr c\_sum\_tran

ldi temp\_L,0x72; TODO this some shit, that i need understand

st Y+,temp\_L ; first bite - address

add c\_sum\_tran,temp\_L

mov temp\_L,position\_1

st Y+,temp\_L ; second bite - position first gain

add c\_sum\_tran,temp\_L

mov temp\_L,position\_2

st Y+,temp\_L ; third bite - position second gain

add c\_sum\_tran,temp\_L

ldi temp\_L,F\_driver

st Y+,temp\_L ; fourth bite - nfault

add c\_sum\_tran,temp\_L

com c\_sum\_tran

mov temp\_L,c\_sum\_tran

st Y+,temp\_L ; fiveth bite - c\_sum

ret

;Программа обработки прерывания АЦП

; ============================================

IN\_ADC: push temp\_L

push temp\_H

in temp\_L,SREG

push temp\_L

;

rd\_ADC: in temp\_L,ADCL

in temp\_H,ADCH

add ADC\_l,temp\_L

adc ADC\_h,temp\_H

inc cou\_ADC

cpi cou\_ADC,Val\_N\_ADC;колич-во преобразований АЦП

breq end\_ADC

ldi temp\_L,(1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADSC);;start convers.

out ADCSR,temp\_L

ex\_INADC: pop temp\_L

out SREG,temp\_L

pop temp\_H

pop temp\_L

reti

;-------------------------

end\_ADC: clr cou\_ADC

set

bld byte\_fl,F\_end\_ADC

rjmp ex\_INADC

number\_gain:

cpi number,0 ; 0 - first 1 - second

brne second

;---------------------------------------

first: inc number

minus\_more\_1:

sub Adc\_h,eight

sbc Adc\_l,temp\_H

brcs check\_pos\_1

inc position\_1 ; тут потом будет среднее значение, когда все обнулится( как Путин)

rjmp minus\_more\_1

check\_pos\_1:

cpi position\_1,zero\_position

breq stop\_drive1

cpi position\_1,full\_position

breq stop\_drive1

rjmp change\_channel

;---------------------------------------

second: dec number

minus\_more\_2:

sub Adc\_h,eight

sbc Adc\_l,temp\_H

brcs check\_pos\_2

inc position\_2 ; тут потом будет среднее значение, когда все обнулится( как Путин)

rjmp minus\_more\_2

check\_pos\_2:

cpi position\_2,zero\_position

breq stop\_drive2

cpi position\_2,full\_position

breq stop\_drive2

rjmp change\_channel

;---------------------------------------

stop\_drive1:

in temp\_L,PORTB

andi temp\_L,0b00000100

out PORTB,temp\_L

rjmp change\_channel

stop\_drive2:

in temp\_L,PORTB

andi temp\_L,0b00100000

out PORTB,temp\_L

rjmp change\_channel

;---------------------------------------

change\_channel:

clt

sbrs byte\_fl,F\_number

set

bld byte\_fl,F\_number

ch\_mux\_ADC:

sbrs byte\_fl,F\_number

rjmp first\_in

ldi temp\_L,(1<<MUX0)|(0<<MUX1) ; второй движок

out ADMUX,temp\_L

ex\_c\_mux:

clr ADC\_h

clr ADC\_l

ldi temp\_L,(1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADSC);;start convers.

out ADCSR,temp\_L

rjmp ex\_INADC

;-------------------------------

first\_in: ldi temp\_L,(0<<MUX0)|(0<<MUX1) ; первый движок

out ADMUX,temp\_L

rjmp ex\_c\_mux

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Подпрограмма обработки прерывания USART RX Complete

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

REC\_date: push temp\_L

push temp\_H

in temp\_L,SREG

push temp\_L

;

in temp\_H,UCSRA

rd\_UDR: in temp\_L,UDR

rjmp rt\_rec

;

pop\_rec:

pop temp\_L

out SREG,temp\_L

pop temp\_H

pop temp\_L

reti

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

rt\_rec:

andi temp\_H,(1<<FE)|(1<<DOR)|(1<<PE)

breq USART2NoError

rjmp pop\_rec

USART2NoError:

st Y+,temp\_L

inc Cou\_Rec

mov temp\_H,Cou\_Rec

cpi temp\_H,0x01

breq rec\_1\_b

cpi temp\_H,VAL\_REC

breq rec\_end

add c\_sumREC,temp\_L

rjmp pop\_rec

rec\_1\_b: cpi temp\_L,right\_adres

brne error\_addres

clr temp\_H

out UCSRA,temp\_H

add c\_sumREC,temp\_L

in temp\_L,TIMSK

set

bld temp\_L,OCIE1A; OCIE1A разрешить прерывание

out TIMSK,temp\_L

ldi temp\_L,(1<<WGM12)|(1<<CS10); WGM3=0,WGM2=1,WGM1=0,WGM0=0,режим CTC

; clkI/O/1 (No prescaling),CS2,CS1,CS0=001

out TCCR1B,temp\_L; (No prescaling CS0=1 )

rjmp pop\_rec

rec\_end:

com temp\_L; инверсия принятого байта

cp temp\_L,c\_sumREC; проверка контр суммы

brne ex\_rec

set

bld Byte\_fl,F\_receive; прием верных данных

;

ex\_rec: in temp\_L,TIMSK

clt

bld temp\_L,OCIE1A; OCIE1A запретить прерывание

out TIMSK,temp\_L

ldi temp\_L,(1<<WGM12); WGM3=0,WGM2=1,WGM1=0,WGM0=0,режим CTC

; clkI/O/1 (stop TCNT1),CS2,CS1,CS0=000

out TCCR1B,temp\_L

ldi temp\_L,0x00

ldi temp\_H,0x00

ldi temp\_L,(1<<MPCM)

out UCSRA,temp\_L

ldi temp\_L,0x00

out TCNT1H,temp\_H

out TCNT1L,temp\_L

clr Cou\_Rec

clr c\_sumREC

ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема

ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема

rjmp pop\_rec

error\_addres:

clr Cou\_Rec

ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема

ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема

rjmp pop\_rec

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Subroutine interrupt USART Data register Empty

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

B\_TRANS: push temp\_L

in temp\_L,SREG

push temp\_L

;

ldi temp\_L,(1<<TXEN)|(1<<TXCIE);разрешить прерывание TXC

out UCSRB,temp\_L

ldi YL,low(varBuf\_Txd); Load Y register low буфер передачи

ldi YH,high(varBuf\_Txd); Load Y register high буфер передачи

ld temp\_L,Y+

clt

bld Byte\_fl,F\_receive;

out UDR,temp\_L

;

pop temp\_L

out SREG,temp\_L

pop temp\_L

reti

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Subroutine interrupt USART, Tx Complete

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TRANdate: push temp\_L

in temp\_L,SREG

push temp\_L

push temp\_H

rjmp r\_trans

;

pop\_tran:

pop temp\_H

pop temp\_L

out SREG,temp\_L

pop temp\_L

;

reti

;

r\_trans: inc Cou\_Tran

mov temp\_L,Cou\_Tran

cpi temp\_L,VAL\_TR

breq end\_tr

ld temp\_L,Y+

out UDR,temp\_L

rjmp pop\_tran

end\_tr: clr Cou\_Tran

ldi temp\_L,(1<<RXEN)|(1<<RXCIE);

out UCSRB,temp\_L

ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема

ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема

cbi PORTB,CID

set

bld Byte\_fl,F\_trans

rjmp pop\_tran

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Subroutine interrupt OC1A

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Time\_OUT: push temp\_L

push temp\_H

in temp\_L,SREG

push temp\_L

;

ldi temp\_L,(1<<WGM12);WGM13=0,WGM12=1,WGM11=0,WGM10=0,режим CTC

; No clock source,CS42,CS41,CS40=000

out TCCR1B,temp\_L;

in temp\_L,TIMSK

clt

bld temp\_L,OCIE1A;OCIE1A запретить прерывание

out TIMSK,temp\_L

ldi temp\_H,0x00

ldi temp\_L,0x00 ;

out TCNT1H,temp\_H;сброс счетчика

out TCNT1L,temp\_L

clr Cou\_Rec

clr c\_sumREC

ldi YL,low(varBuf\_Rxd) ; Load Y register low буфер приема

ldi YH,high(varBuf\_Rxd) ; Load Y register high буфер приема

;

pop temp\_L

out SREG,temp\_L

pop temp\_H

pop temp\_L

reti