СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc28248391)

[1 ВЫБОР МИКРОКОНТРОЛЛЕРА 6](#_Toc28248392)

[1.1 Основные ядра, используемые в микроконтроллерах 6](#_Toc28248393)

[1.2 Обзор серий микроконтроллеров 8](#_Toc28248394)

[1.3 Выбор модели микроконтроллера 10](#_Toc28248395)

[2 ПРИНЦЫП РАБОТЫ УСТРОЙСТВА 12](#_Toc28248396)

[2.1 Структурная схема электрическая 12](#_Toc28248397)

[2.2 Функциональная схема электрическая 15](#_Toc28248398)

[2.3 Принципиальная схема электрическая 18](#_Toc28248399)

[3 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ 21](#_Toc28248400)

[3.1 Структура программного кода 21](#_Toc28248401)

[3.2 Описание режимов программного кода 25](#_Toc28248402)

[ЗАКЛЮЧЕНИЕ 27](#_Toc28248403)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 28](#_Toc28248404)

**ВВЕДЕНИЕ**

В современном мире электроника и различная вычислительная техника играет важную роль в различных областях человеческой деятельности. Это связано с тем, что прогресс не стоит на месте, все повсеместно модернизируется, улучшается, автоматизируется. Человек двигается и с каждым разом перед ним стоят более серьезные вопросы и задачи, которые без применения различных электронных устройств либо попросту невозможны, либо займут слишком много времени для решения той или иной задачи. Все эти, казалось бы, маленькие факты и нюансы достаточно серьезно влияют на скорость и точность получаемых результатов, фактически любых измерений. Кроме того, в последнее время в научно-технической сфере, все чаще стали применяться различные микроконтроллеры, которые позволяют ускорить и упросить достаточно большой спектр задач.

Поскольку практикум и выполнение лабораторных работ является важной частью при подготовке студентов инженерной специальности. А навыки, получаемые в физической лаборатории: понимание наблюдаемых процессов, пользование измерительными приборами, обработка полученных результатов, позволяют студентам применять их в своей будущей трудовой деятельности. Было принято решение о создании универсального блока управления на микроконтроллере. Данное устройство позволяет более детально изучить различные физические явления, связанные с линейными или угловыми перемещениями. [9]

Целью данной работы является разработка функционального блока для управления и обработки измерений линейных и угловых перемещений физических тел с использованием микроконтроллера.

Основные задачи, необходимые для разработки устройства измерения линейного и углового перемещения на микроконтроллере:

1. Выбор оптимального микроконтроллера, который бы отвечал следующим параметрам:
   1. быстродействие
   2. энергопотребление
   3. габариты
   4. объем памяти
   5. простота обслуживания
   6. стоимость
2. Проектирование структурной электрической схемы.
3. Проектирование функциональной электрической схемы.
4. Проектирование принципиальной электрической схемы.
5. Разработка и написание программного обеспечения.
6. Отладка программного кода.
7. Сборка и регулировка разработанного устройства.

# 1 Устройство измерения линейного и углового перемещения на микроконтроллере

Для достижения цели данной дипломной работы необходимо реализовать измерительное устройство со следующими техническими характеристиками:

- диапазон измерения линейного перемещения 0 – 100 см;

- диапазон измерения углового перемещения 0 – 180°;

- дискретность измерения линейного перемещения 1 мм;

- дискретность измерения углового перемещения 0,1°;

- реализовать дистанционное управление (ДУ) внешними устройствами (многофункциональные измерители временных интервалов, а также системами управления механизмом перемещения);

- в качестве датчика применить оптоэлектронный энкодер;

- блок управления и измерения выполнить на микроконтроллере;

- динамическая индикация;

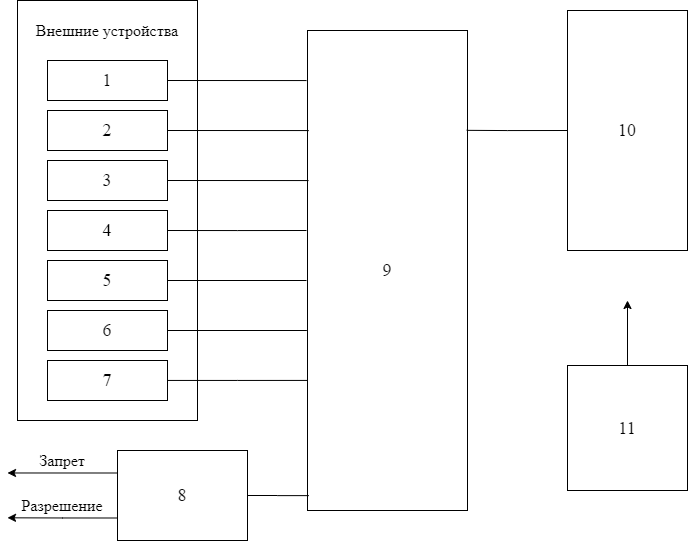
- напряжение питание 5В;

- составить структурную, функциональную и принципиальную схемы;

Схемы электрические структурная, функциональная, принципиальная должны учитывать заданные технические характеристики. И позволить изготовить устройство измерения линейного и углового перемещения на микроконтроллере.

## 1.1 Структурная схема электрическая

Схема электрическая структурная разработанного устройства представлена на рисунке 1.



1-датчик ИК, 2 - датчик подсчета периодов, 3 - датчик линейного перемещения, 4 - датчик углового перемещения влево, 5 - датчик углового перемещения вправо, 6 - блок выбора режима работы, 7 – звуковой излучатель, 8 – блок дистанционного управления, 9 - блок измерения и управления, 10 - блок индикации, 11 - блок питания 5 В

Рисунок 1 - Структурная схема электрическая

В данной структурной схеме представлены основные функциональные компоненты разработанного устройства. Кроме того, в ней представлены взаимосвязи между основными компонентами данной схема. Стоит отметить, что каждый компонент представленный в данной схеме реализует полностью законченное действие.

Поскольку блок измерения и управления 9 разрабатывается, как многофункциональное устройство, в данном блоке реализуется 5 режимом работы. Далее приведены основные режимы, реализованные на программном уровне:

1. Режим измерения линейного перемещения
2. Режим измерения углового перемещения
3. Режим подсчета периодов колебания
4. Режим таймера
5. Режим секундомера

При включении блока управления в сеть, устройство совершает аппаратный сброс. Далее с помощью Блока выбора режима 6 представленного в структурной схеме, происходит установка режима и его выбор.

Описание и рассмотрение режимов работы разработанного и изготовленного устройства описано ниже.

В состав схемы режима измерения линейного перемещения входят следующие компоненты: датчик ИК 1, датчик линейного перемещения 3, блок выбора режима работы 6, внешние сигналы управления 8, блок измерения и управления 9, блок индикации 10.

Первым этапом выполнения данного режима является установка расстояния при помощи блока выбора режима 6, которое должно пройти тело. Задаваемое значение отображается на блоке индикации 10. При поступлении разрешающего сигнала на блок дистанционного управления 8, после чего ожидается получение управляющего сигнала с ИК датчика 1 на блок управления и измерения 9. Данный управляющий сигнал является флагом для перехода блока 9 во второй этап выполнения управляющей программы.

Вторым этапом выполнения данного режима является получение и подсчет импульсов, получаемых от датчика линейного перемещения 3. После чего происходить обработка данных измерений на программном уровне. По окончанию линейного перемещения тела, результат измерений выводиться на блок индикации 10, после чего устройство переходит в исходное положения и может быть использовано в любом из пяти возможных режимов работы.

В состав схемы режима измерения углового перемещения входят следующие компоненты: датчик углового перемещения влево 4, углового перемещения вправо 5, блок дистанционного управления 8, блок измерения и управления 9, блок индикации 10.

При старте данного режима ожидается получение разрешающего сигнала с блок дистанционного управления. После чего ожидается получение сигналов с датчиков углового перемещения 4 и 5 на соответствующие входы блока управления и измерения 9. После чего происходит программная обработка полученных результатов. После выполнения программной части результат вычислений выводиться на блок индикации 10. После окончания данного режима, разработанное устройство переходит в исходное положения и может быть использовано в любом из возможных режимов работы.

Третий режим позволяет производить измерения периодов механических колебаний различных видом маятников (математический, маятник Обербека и др.). В состав схемы данного режима входят следующие компоненты: датчик подсчета периодов 2, блок выбора режима работы 6, блок измерения и управления 9, блок индикации 10.

Начальным этапом данного режима, является установка необходимого количества колебаний для измерений с помощью блока выбора режима работы 6. После чего ожидается получение разрешающего сигнала на блок дистанционного управления 8. Данный сигнал является флагом для блока управления и измерения 9, который начинает подсчет импульсов, получаемых с датчика подсчета периодов 2. Сигнал вышеупомянутого датчика поступает на блок 9 при каждом перекрытии датчика 2 стержнем маятника. После чего происходить программная обработка получаемых данных. По завершению данного режима, подается звуковой сигнал с использованием звукового излучателя 7 и вывод информации на блок индикации 10. По окончанию данного режима, устройство переходит в исходное положения и может быть использовано в любом из пяти возможных режимов работы.

**1.2 Выбор элементарной базы**

**1.2.1 Выбор микроконтроллера**

Микроконтроллер – это специальная микросхема, предназначенная для управления различными электронными устройствами. Данное устройство объединяет в себе процессор, память, ПЗУ и периферию внутри одного корпуса, внешне похожего на обычную микросхему. [5]

Выбор микроконтроллера следует начать с описания основных семейств МК. Основные рассматриваемыми вариантами являются семейства ARM, AVR и PIC.

ARM — семейство лицензируемых 32-битных и 64-битных микропроцессорных ядер разработки компании ARM Limited. Представляет собой микроконтроллеры придерживающихся идеологии RISC для системных команд. Данное семейство микроконтроллеров является одним из лидеров на рынке микроконтроллеров за счет своей простоты и дешевизны. По приведенной статистике, на процессоры ARM приходилось порядка 90 % всех встроенных 32-разрядных процессоров. Поскольку данные процессоры имеют низкое энергопотребление, в связи с этим они находят широкое применение во встраиваемых системах и преобладают на рынке мобильных устройств, для которых данный фактор немаловажен [11].

AVR - семейство восьми битных микроконтроллеров. Данные микроконтроллеры являются представителями гарвардской архитектуры, что подразумевает под собой разделение памяти программ и памяти данных. Также стоит отметить, что данное семейство имеет систему команд, близкую к идеологии RISC. Особенностью такого подхода, является упрощение системы команд, как результат конечный пользователь получает выигрыш в производительности и быстродействии. Процессор AVR имеет 32 8-битных регистра общего назначения, объединённых в регистровый файл. Кроме всего прочего, стоит отметить, что система команд микроконтроллеров AVR весьма развита и насчитывает в различных моделях от 90 до 133 различных инструкций. Большинство команд выполняется за 1 такт. Что крайне положительно сказывается на производительности. [10]

PIC - семейство восьми битных или же шестнадцати битных микроконтроллеров. Данному семейству соответствует RISC архитектура, обеспечивающая выпол­нение большинства команд процессора за один машинный цикл. Гарвардская архитектура, обес­печивающая одновременный доступ к памяти данных и программ.  КМОП технология, обеспечивающая: • полностью статический режим работы, при котором остановка тактового генератора не приводит к потере логических состояний внутренних узлов, широкий диапазон напряжений питания (2…6 В) и температур (–40…+70 °С) а также малое энергопотребление.

Следующим этапом является сравнение наиболее популярных 8 – ми разрядных МК представленных на рынке. Для этого были выбраны следующие представители. Семейство AVR представлено микроконтроллером серии Atmega8, а именно моделью ATmega328P-AU. Семейство ARM будет представлено так же 8 – ми разрядным микроконтроллером STM8L151C8T6. Представителем семейства PIC является PIC16F627A.

Таблица 1 - сравнительная характеристика наиболее популярных моделей семейства: AVR, ARM, PIC.

|  |  |  |  |
| --- | --- | --- | --- |
| Параметры | Модель микроконтроллера | | |
| ATmega328P-AU | STM8L151C8T6 | PIC16F627A |
| Семейство | AVR | ARM | PIC |
| Серия | ATmega | STM8 | PIC16 |
| Разрядность | 8 бит | 8 бита | 8 бита |
| Архитектура | RISC | RISC | RISC |
| Тактовая частота, МГц | 20 | 16 | 20 |
| Объем памяти программ, кбайт | 32 | 64 | 3,5 |
| Тип памяти программ | Flash | flash | flash |
| Объем EEPROM, кбайт | 1 | 2 | 0.128 |
| Объем RAM, кбайт | 2 | 4 | 0,224 |
| I2C интерфейс | 1 | 1 | - |
| Напряжение питания | 1.8…5.5 в | 1.8…3.6 в | 3…5.5 в |
| Список команд | 90 | 32 | 35 |
| Стоимость, BYN | 4.40 | 4.80 | 3.40 |

Из таблицы 1 можно увидеть, что все микроконтроллеры, приведенные в таблице, являются представителя RISC архитектуры. Кроме того, стоит отметить, что все 3 модели имеют одинаковый тип памяти программ, но в тоже время модель из серии STM8, имеет в два раза больший объем данной памяти, в сравнении с моделью из серии Atmega и гораздо больший объем в сравнении с моделью из серии PIC16. Кроме того в пользу выбора модели из серии STM8, мог бы говорить тот факт, что у данной модели в два раза больше энергонезависимой памяти (EEPROM). Аналогичная ситуация обстоит и с оперативной памятью (RAM). Но стоит заметить, что модель из серии STM8 проигрывает двум оставшимся моделям в тактовой частоте, которая влияет на производительность микроконтроллера. Также немаловажным параметром является количество команд для управления микроконтроллером и в данном случае, бесспорным лидером является представитель серии Atmega. Так же немаловажным фактом при выборе микроконтроллера было наличие I2C интерфейса. В модели серии PIC16 данный интерфейс отсутствует, что является серьезным недостатком данного микроконтроллера.

Сравнив 3 наиболее популярные модели, было решено выбирать модель микроконтроллера серии Atmega или же серии STM8. Поскольку модель микроконтроллера серии PIC16 пусть и дешевле, но проигрывает оставшимся двум вариантом почти по всем характеристикам и параметрам.

После более детально рассмотрения двух вышеупомянутых моделей, было принято решение отдать предпочтение модели серии Atmega. Данное решение обосновывается тем фактом, что данная модель выигрывает по быстродействию, а также имеет больший список различной технической литературы, что упрощает программную разработку. Так же немаловажным фактом является простота использования, а как результат, простата дальнейшей поддержки устройства в рабочем состоянии. Стоит отметить, что абсолютна вся документация для модели серии Atmega распространяется по свободной лицензии.

**1.2.2 Выбор операционного усилителя датчиков линейного и углового перемещения**

В качестве операционного усилителя датчиков линейного и углового перемещения, было принято использовать готовое решение на основе модуля KY-037. В состав которого входят два компаратора LM393YD.

**1.2.3 Выбор устройства для блока выбора режимов**

В качестве устройства для блока выбора режима было принято решение использовать электромеханический инкрементный энкодер KY-040 имеющий 24 импульса на оборот. Плюсом данного устройства является его простата, поскольку в данном типе энкодера применятся скользящие контакты. Также плюсом является наличие кнопки. Минусом данного устройства является дребезг контактов, но это решается программной обработкой.

**1.2.4 Выбор устройства блока индикации**

В качестве индикации рассматривались два варианта: lcd дисплей 1602А или два семи сегментных модуля tm1637. Поскольку для решения основной технической задачи касающейся индикации (динамическая индикация) подходят оба варианта, все же было решено отдать предпочтения lcd дисплею 1602А. Это вызвано тем фактом, что, выбрав данное устройство, мы можем выводить абсолютно любую текстовую информацию. Данный факт позволяет разработать приятный и отзывчивый интерфейс для пользователя.

Важным фактом является использование вышеупомянутого дисплея с подключением через I2C интерфейс. Данное решение применяется для уменьшения задействованных портов ввода/вывода микроконтроллера. Так при обычном подключении необходимо задействовать 6 цифровых портов, в тоже время при использовании модуля переходника на основе микросхемы PCF8574AT, которая предназначена для расширения количества линий ввода/вывода, задействованы лишь 2 цифровых порта. Микросхема подключается по I2C интерфейсу и имеет порт из 8 линий ввода/вывода.

**1.2.5 Выбор звукового излучателя**

В качестве звукового излучателя был выбран простейший пьезоизлучатель KPM-G1205В. Данный излучатель удобен как своей простотой, так и простотой подключения его к МК. А именно один из его контактов подключается к цифровому порту блока управления и измерения, имеющему в наличии аппаратный широтно-импульсный модулятор.

**1.2.5 Выбор блока питания**

В качестве блока питания применяем внешний, готовый источник питания номиналом 9-12В и током не менее 1А.

## 1.3 Функциональная схема электрическая

На рисунке 2 приведена функциональная схема разработанного устройства. Схема содержит в себе функциональную схему блока внешних устройств, а также блока управления и измерения.

**1.3.1 Функциональная схема блока выбора режима**

Выбор режимов работы устройства происходит с помощью использования инкрементного электромеханического энкодера. Функциональная схема данного модуля построена на основе двух скользящих контактов CLK и DT, а также кнопки SW подключенной по схеме с подтягивающим резистором. При вращении энкодера с выходов CLK и DT поступают прямоугольные импульсы, на соответствующие входы блока управления и измерений, сдвинутые друг относительно друга на фазовый угол 90 °. При вращении в одну сторону задний фронт импульса на выходе CLK соответствует паузе между импульсами на выходе DT. После чего происходит программная обработка получаемых сигналов с помощью микроконтроллера.

**1.3.2 Функциональная схема блока индикации**

Поскольку для динамической индикации используется lcd дисплей 1602А подключенный по интерфейсу I2C с модулем PCF8574AT применяющемся для расширения линий ввода/вывода. Микросхема подключается по I2C интерфейсу и имеет порт из 8 линий ввода/вывода, принцип функционирования простой, при записи байта данных в микросхему, линии порта принимают уровни, соответствующие значениям битов полученного байта. Операция чтения возвращает байт данных, биты которого указывают состояние линий порта. Таким образом, микросхема позволяет расширить количество линий ввода/вывода, используя два управляющих провода.

**1.3.3 Функциональная схема датчиков линейного и углового перемещения.**

Функциональные и принципиальные схемы электрические внешних устройств построены по однотипным типовым схемам на операционном усилителе. Отличие заключается лишь в различных коэффициентах усиления, которые устанавливаются с помощью регулировочного резистора.

Принцип работы первых трех режимов, а именно режим линейного перемещения, режим углового перемещения и режим подсчета периодов рассматриваются по примеру функциональной схемы датчика линейного перемещения. Поскольку остальные режимы работы разработанного устройства реализованы на программном уровне.

Функциональная схема датчика линейного перемещения, построена на оптопаре состоящей из пары фотодиодов FD1 AL106 (излучающий) и FD2 AL107 (принимающий). Сигнал от FD1 прошедший механический энкодер, представляющий собой диск с прямоугольными отверстиями и напечатанный на 3D принтере, через FD2 поступает на однокаскадный усилитель, собранный на ВТ1 и компаратор представляющий собой операционный усилитель AD1. С операционного усилителя сигналы ТТЛ уровня поступают на советующий вход блока управления и измерения.

**1.3.4 Функциональная схема звуковой индикации**

Сигнализирование о завершение того-либо иного режима происходит с помощью пьезоизлучателя. При поступлении сигнала от блока управления и измерения сигнала на соответствующий цифровой порт вышеупомянутого блока, происходить широтно-импульсная модуляция благодаря наличии аппаратного широтно-импульсного модулятора на соответствующем порту. После чего за счёт ШИМ полученные импульсы поступают на пьезоизлучатель, который в свою очередь побуждает колебательные движения биморфной конструкции пьзоизлучателя, преобразовывая их в звук.

**1.3.6 Функциональная схема блока управления**

Схема содержит устройства связи системы с внешними устройствами, а также устройство индикации. Внешнее управление становится возможным и при помощи системного разъема, но по другим правилам в отличие от порта ввода/вывода, что расширяет возможности контроллера, с его помощью может осуществляется связь с шинами базовой ЭВМ. Также этот разъем может быть использован при отладке работы микропроцессорной системы в целом.

Так же, стоит отметь наличие в системе трех шин, а именно шины управления, адреса и данных. Они представляют из себя проводные линии(проводники) соединяющие между собой различные составные части всей системы. Шины имеют различные разрядности. Также стоит отметить, что разрядность шины может зависеть от производительности системы. Большое количество информации, вызывает потребность в большей шине данных.

Далее будет рассмотрено назначение каждой из шин, применяемых в данном микроконтроллере.

Шина данных – шина основное назначение которой является передача различного рода данных. Данные могут быть получены как из памяти, так и от устройства ввода/вывода и переданы процессору. Поскольку обратное также верно, можно говорить о том, что данная шина является двунаправленной. Следует отметить, что используемый микроконтроллер имеет 8-ми разрядную шину данных, что говорит о том, что за одну передачу/прием максимум может быть передано 8 бит информации.

Шина адреса — главное назначение данного элемента является указание адреса ячеек памяти или портов ввода/вывода. Разрядность же имеет прямую зависимость от необходимого размера памяти.

Шина управления – основной целью данного компонента является передача информации устройству ввода/вывода или памяти о готовности микропроцессора выполнить пересылку данных.

Как результат, процесс работы имеет следующую структуру. По управляющей шине передаются сигналы, которые позволяют устройству ввода/вывода или же памяти обращаться к процессору с запросами. Разрядность данной шины большей степени зависит от типа используемого процессора и количества его управляющих или используемых сигналов.

Из приведенной выше схемы видно, что инициатором всех действий, выполняемых системой является микропроцессор. Данный микропроцессор вырабатывает сигналы управления и выдает их на шину управления. Далее по данной шине сигналы подают на устройства памяти, а также на устройства ввода/вывода. Стоит отметить, что именно процессором задается и адрес выбранной ячейки памяти. После чего выбранный адрес подает на шину адреса и на адресные входы микросхем памяти, в свою очередь по шине данных информация идет в процессор или из него.

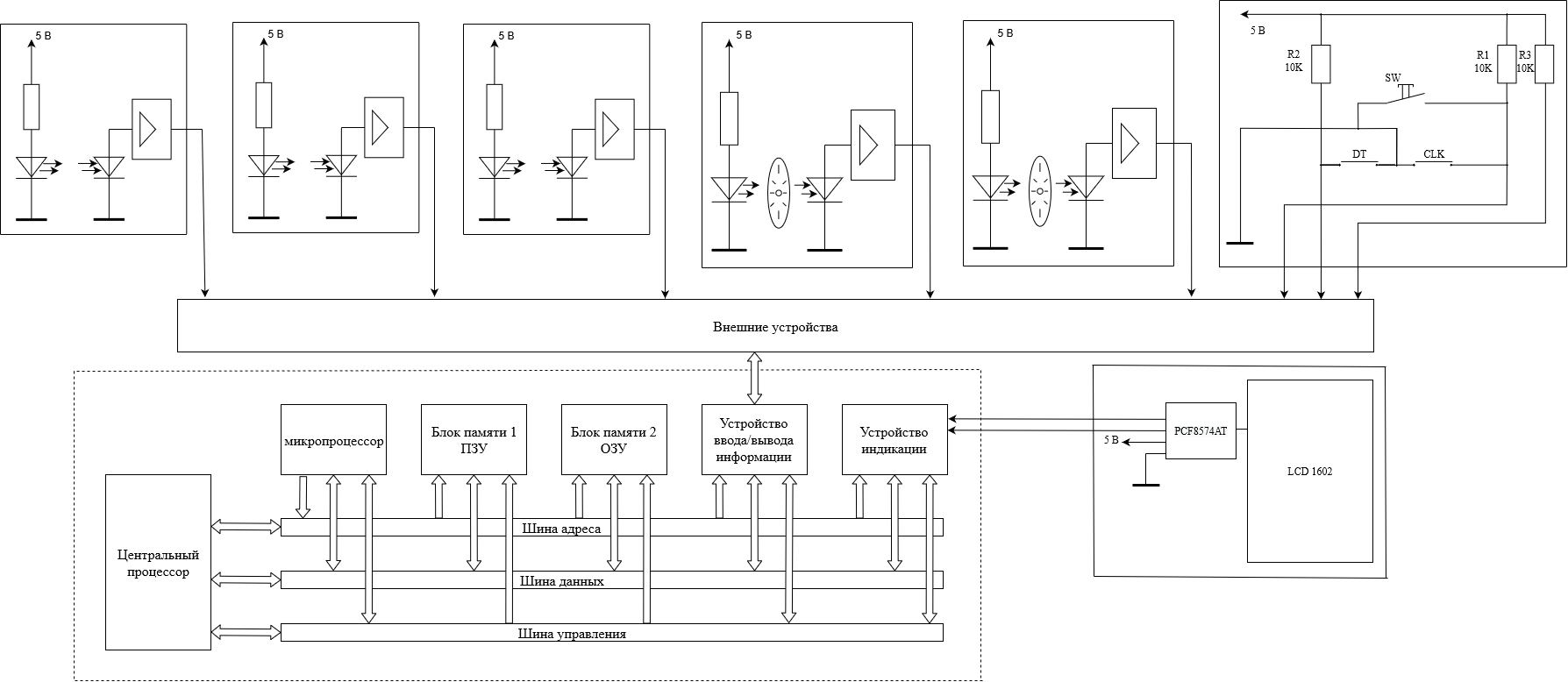


Рисунок 2 – функциональная схема разработанного устройства

## 2.3 Принципиальная схема электрическая

Рассмотрение принципиальной схемы устройства На рисунках 3а, 3б, 3в представлена принципиальная схема электрическая разработанного устройства.

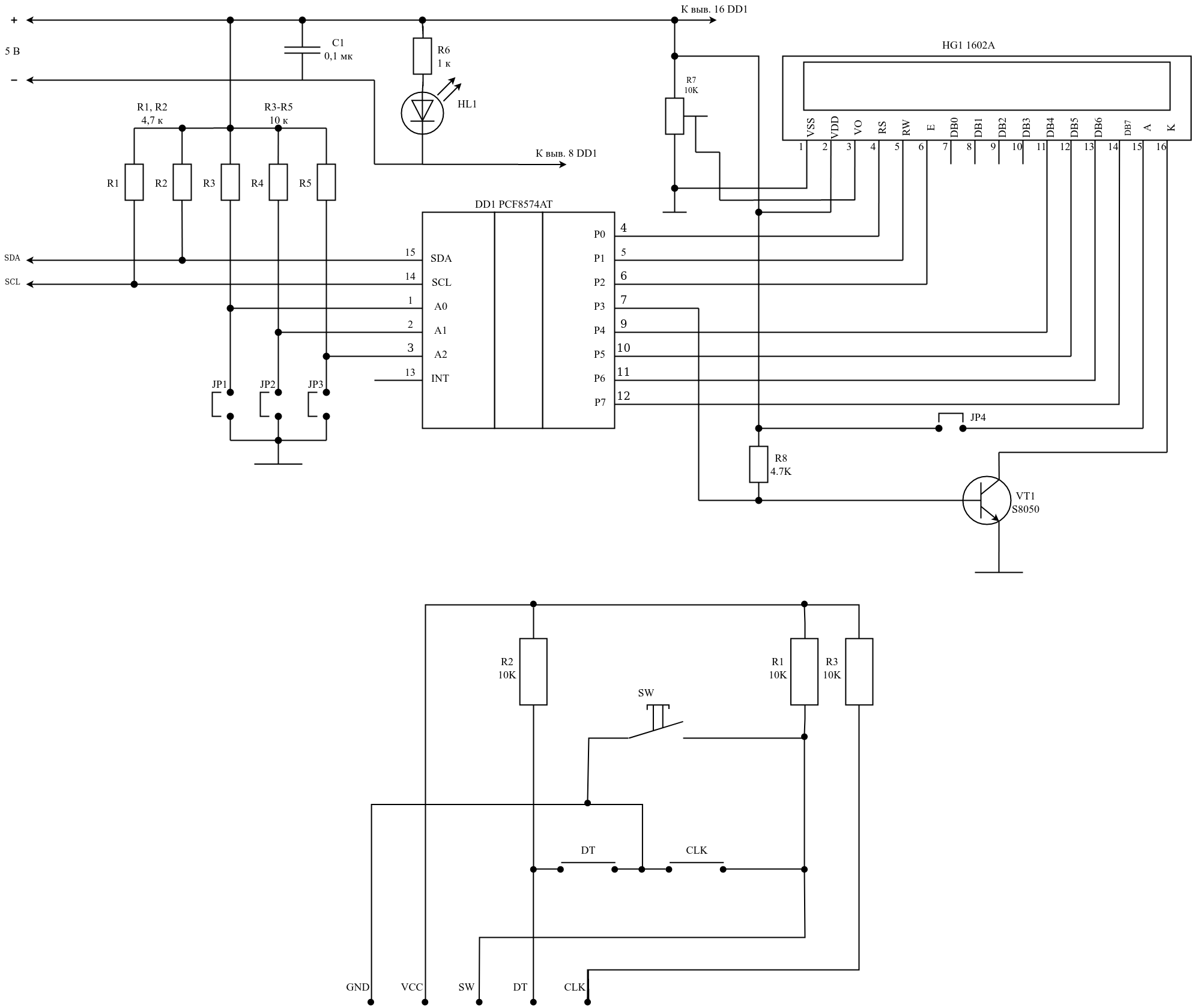


Рисунок 3а — принципиальная схема электрическая блока индикации, а также блока выбора режима.

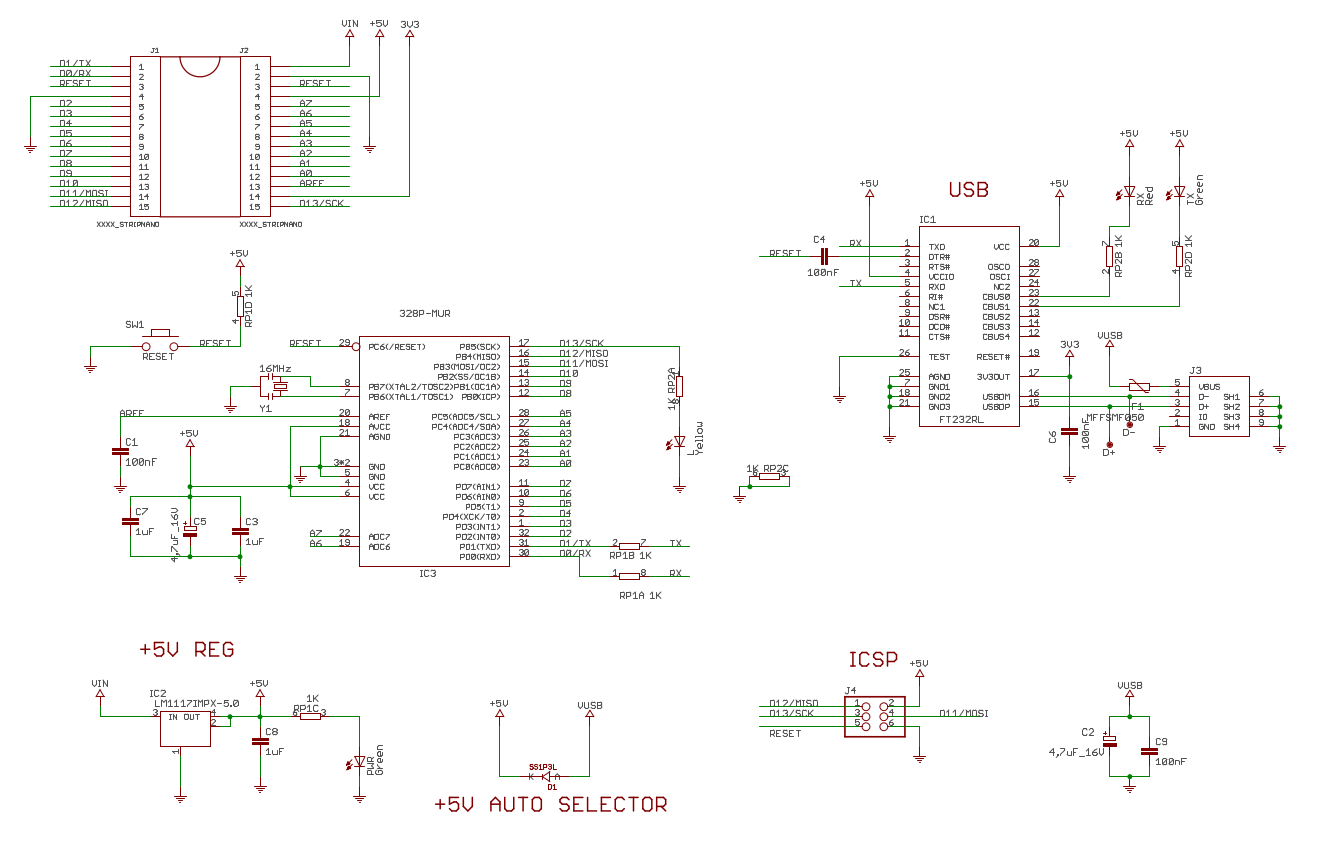


Рисунок 3б — принципиальная схема электрическая блока управления и измерения.

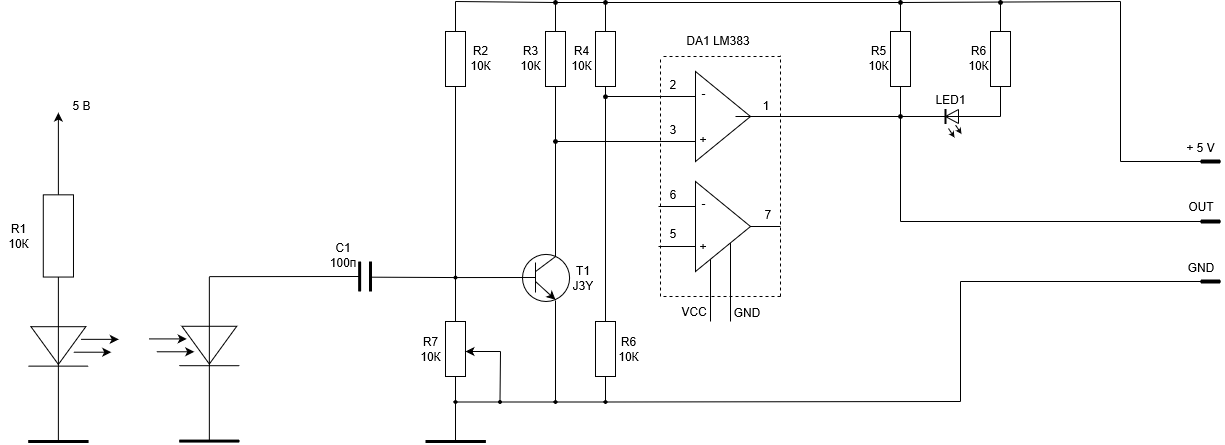


Рисунок 3в — принципиальная схема электрическая усилителя датчиков внешних устройств.

Поскольку объем данной дипломной работы ограничен, а схема структурная электрическая и схема функциональная электрическая рассмотрены достаточно подробно. Принципиальная схема в данной работе не рассматривается.

# 3 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

## 3.1 Структура программного кода

Поскольку для реализации задачи поставленной в данной дипломной работе был выбран микроконтроллер Atmega328P, то следует сказать, что написание программного кода для данного микроконтроллера осуществляется с помощью языка С++.

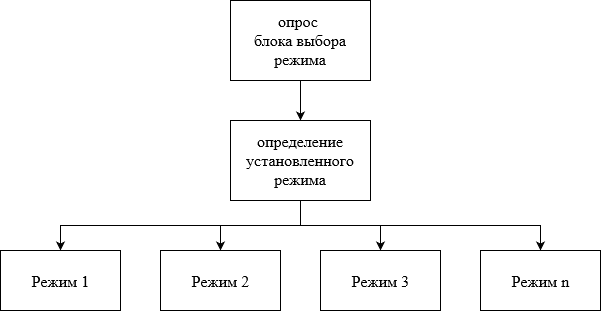


Рисунок 4 – структура программного кода

Как видно из рисунка 4 разработанный блок является многофункциональным. Это достигается за счет возможности выбора режима и написания большого количества программных модулей, реализующих нужный функционал для определенных измерений.

Логика описанного ниже программного кода представлена в виде блок-схемы. Данная блок-схема представлена в приложении A.

Программирование микроконтроллера начинается с создания двух функций с именами *setup()* и *loop()*. Функция *setup()* выполняется лишь один раз и реализует первоначальная настройка основных компонентов используемых в дальнейшем в программном коде. По завершению, функция *setup()* передает управление микроконтроллером функции *loop()*, которая реализует основную логику выполняемую данным микроконтроллером [3].

void setup() {

/\*настройка блока индикации \*/

lcd.*init();*

lcd*.backlight();*

/\*настройка энкодера\*/

*pinMode(pin\_DT, INPUT);*

*pinMode(pin\_CLK, INPUT);*

/\*настройка кнопки энкодера\*/

*pinMode(pin\_Btn, INPUT\_PULLUP);*

/\*настройка звукового излучателя \*/

*pinMode(pin\_Sound\_Signal, OUTPUT);*

/\*настройка датчиков линейного перемещения\*/

*pinMode(pin\_Optocoupler, INPUT);*

*pinMode(pin\_Impulse\_Counter, INPUT);*

}

Поскольку в данной работе для подключения внешних устройств используется двунаправленная шина данных, в начале программного кода необходима установка режима работы используемых выводов. Для этих целей используется функция *pinMode(pinNumber, pinType)*.

В качестве параметра pinNumber используется номер вывода. Параметром pinType является константа, указывающая на режим работы. Параметр pinType может принимать следующие значения:

*INPUT* – режим в котором шина микроконтроллера поддерживается в высокоимпедансном состоянии, для работы с внешними источниками сигналов.

*OUTPUT* - режим в котором шина микроконтроллера поддерживается в низкоимпедансном состоянии, при котором на внешнее устройство выдается максимально возможный ток.

*INPUT\_PULLUP* – режим аналогичен режиму INPUT, разница заключается в том, что получаемый сигнал инвертируется.

void *loop()* {

/\*функция выбора режима работы\*/

*choiseOfMode();*

}

Следующим этапом является вызов функции *choiseOfMode()*.Данная функцияотвечает за вызов одного из 5 возможных режимов. Код данной функции представлен ниже.

void *choiseOfMode()* {

/\*Флаг для перехода в выбранный режим по нажатию кнопки энкодера\*/

isEncoderButtonPressed = false;

switch (*getEncoderState*()){

case eNone:

return;

case eButton:

isEncoderButtonPressed = true;

break;

case eLeft:

counter--;

*printOption(counter);*

break;

case eRight:

 counter++;

*printOption(counter);*

break;

}

/\*Переход в выбранный режим если кнопка энкодера нажата\*/

if (isEncoderButtonPressed){

*goToOption(counter);*

}

}

Для использования данной функции необходимы следующие функции: *getEncoderState()*, *printOption()*, *goToOption()*.

Первой рассматриваемой функцией является *getEncoderState()*. Поскольку данная функция позволяет определить положение энкодера. Данная функция опрашивает энкодер каждые 5 миллисекунд возвращая одно из четырех значений:

eNone – энкодер не изменил своего положения.

eLeft – энкодер совершил поворот влево.

eRight – энкодер совершил поворот вправо.

eButton – кнопка энкодера нажата.

Второй рассматриваемой функцией является *printOption()*. Принимающая в качестве параметра целое значение указывающее на выбранный режим. Данная функция выводит название режима, в соответствии выбранному режиму, в виде текстовой информации на блок индикации.

Оставшейся функцией является *goToOption()*. Данная функция, как и уже описанная, принимает значение выбранного режима, после чего контроль переходит к выбранному режиму.

## 3.2 Описание режимов программного кода

Поскольку режимы работы основных режимов идентичны, в данной главе будет рассмотрен программный код для режима измерения линейного перемещения.

После выбора вышеупомянутого режима работы функция *goToOption()* предает управлениемикроконтроллером первому модулю.

void *firstMove()* {

/\*задание расстояния для линейного перемещения\*/

*encoderDistanceSetting();*

/\*измерение пройденного расстояния\*/

*counterDown();*

}

Сначала функция *firstMove()* передает управление функции *encoderDistanceSetting()*. В свою очередь данная функция реализует задание линейного расстояния необходимого для прохождения физического тела, на основе значений получаемых функцией *GetEncoderState()*.

void *encoderDistanceSetting()* {

/\*определение положения энкодера\*/

switch ( *GetEncoderState()* ) {

case eNone: return;

case eLeft: { // Энкодер вращается влево

if (counter == 0){ //блокировка на задание отрицательных значений

return;

}

counter = counter - stapForEncoder;

break;

}

case eRight: { // Энкодер вращается вправо

counter = counter + stapForEncoder;

break;

}

case eButton:{

buttonStateEncoder = true ;

return;

}

}

После того, как было задано расстояние, которое должно пройти тело. Микроконтроллер переходит в режим подсчета импульсов. Этот режим реализуется с помощью функции void counterDown()

void counterDown(){

while (counter != 0 ){

stateMinus = digitalRead(pinMinus);

if (stateMinus != HIGH){ //ожидаем импульса от датчика линейного перемещения

counter--;

break;

}

}

}

После того, как тело завершило свое передвижение, устройство переходит первоначальное состояние. После чего, снова возможен выбор режима с помощью функции *choiseOfMode().*

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной дипломной работы, были проанализированы основные варианты микроконтроллеров, которые могли бы быть использованы в разработке конечного устройства. Также следует отметить, что был проделан большой объем работ как в проектировании структурной и функциональной схемах, так и в разработке принципиальной схемы, которая бы полностью отвечала задачам, поставленным в данной дипломной работе.

Кроме того, было разработано программное обеспечение, достоинством которого является использование так называемых модулей. Что позволяет с легкость добавлять новый функционал.

Как результат, разработанное устройство является многофункциональным блоком управления измерений линейных, а также угловых перемещений.

Поскольку разработанное устройство является универсальным, благодаря нему можно в значительной степени упростить, ускорить, а кроме того увеличить точность большого количества исследовательских, а также различных лабораторных работ при выполнении которых, требуется измерение линейного или же углового перемещения.

Кроме того, стоит отметь, что данный блок также может быть использован и в паре с каким-либо секундомером/таймером, что позволит еще больше расширить диапазон выполняемых лабораторных работ. Примером таких работ может быть машина Атвуда или же работы, связанные с различного рода маятников. [4]

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Гребнев В.В. Микроконтроллеры семейства AVR фирмы Atmel: ИП РадиоСофт, 2002. - 170c.
2. Голубцов М.С. Микроконтроллеры ARM. От простого к сложному: СОЛОН-Пресс, 2003. – 288c.
3. Оптика и атомная физика: Лабораторный практикум по физике/ Под рел.Р.И.Солоухина. Новосибирск: Наука. Сиб. ото-пне, 1983. 384 с.
4. Физический практикум/ Под ред.В.И.Ивероновой1 В 2 ч. М.: Физматгиз, 1967 - 1968. Ч.1-2.
5. ЭВМ в курсе общей физики.Изд-во Моск.ун-та, 1982. 230 с.
6. Методы физических измерений: Лабораторный практикум по физике/Под ред.Р.И.Солоухяна.
7. <https://www.alldatasheet.com/datasheet-pdf/pdf/506518/STMICROELECTRONICS/STM8L151C8T6.html> Дата доступа 11.12.2019
8. <https://web.mit.edu/6.115/www/document/16f628.pdf> Дата доступа 11.12.2019
9. <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf> Дата доступа 13.12.2019
10. https://ru.wikipedia.org/wiki/AVR Дата доступа 9.12.2019
11. https://ru.wikipedia.org/wiki/ARM Дата доступа 25.11.2019

ПРИЛОЖЕНИЕ А

#include <LCD\_1602\_RUS.h>

LCD\_1602\_RUS lcd(0x27, 16, 2);

#define pin\_CLK 4

#define pin\_DT 2

#define pin\_Btn 9

const int pin\_Optocoupler = 5;

const int pin\_Impulse\_Counter = 6;

const int pin\_Sound\_Signal = 3;

bool isNeedPreView = true;

bool isEncoderButtonPressed = false;

unsigned long CurrentTime, LastTime;

enum eEncoderState {

eNone,

eLeft,

eRight,

eButton

};

uint8\_t EncoderA, EncoderB, EncoderAPrev;

int8\_t counter = 0;

bool ButtonPrev;

int leftBallValues[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

int rightBallValues[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

long leftBallTimeValues[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

long rightBallTimeValues[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

eEncoderState GetEncoderState() {

eEncoderState Result = eNone;

CurrentTime = millis();

if (CurrentTime - LastTime >= 5) {

LastTime = CurrentTime;

if (digitalRead(pin\_Btn) == LOW) {

if (ButtonPrev) {

Result = eButton; // Нажата кнопка

ButtonPrev = 0;

}

}

else {

ButtonPrev = 1;

EncoderA = digitalRead(pin\_DT);

EncoderB = digitalRead(pin\_CLK);

if ((!EncoderA) && (EncoderAPrev)) {

if (EncoderB)

Result = eRight;

else

Result = eLeft;

}

EncoderAPrev = EncoderA;

}

}

return Result;

}

void setup() {

Wire.begin();

lcd.init();

lcd.backlight();

pinMode(pin\_DT, INPUT);

pinMode(pin\_CLK, INPUT);

pinMode(pin\_Btn, INPUT\_PULLUP);

pinMode(pin\_Sound\_Signal, OUTPUT);

Serial.begin(9600);

counter = 0;

pinMode(pin\_Optocoupler, INPUT);

pinMode(pin\_Impulse\_Counter, INPUT);

}

void loop(){

if (isNeedPreView) {

prtintTitle();

printOption();

isNeedPreView = false;

}

if (isEncoderButtonPressed) {

goToOption();

isNeedPreView = true;

}

switch (GetEncoderState()){

case eNone:

return;

case eButton:

isEncoderButtonPressed = true;

break;

case eLeft:

counter--;

prtintTitle();

printOption();

isEncoderButtonPressed = false;

break;

case eRight:

counter++;

prtintTitle();

printOption();

isEncoderButtonPressed = false;

break;

}

delay(100);

}

void encoder(){

switch (GetEncoderState()) {

case eNone:

isEncoderButtonPressed = false;

return;

case eButton:

isEncoderButtonPressed = true;

break;

case eLeft:

counter++;

isEncoderButtonPressed = false;

break;

case eRight:

counter--;

isEncoderButtonPressed = false;

break;

}

}

void prtintTitle(){

if (counter > 4){

counter = 0;

}

if (counter < 0){

counter = 4;

}

Serial.println(counter);

lcd.setCursor(0, 0);

lcd.clear();

lcd.print(L"PEЖИМ");

}

void printOption(){

lcd.setCursor(0, 1);

if (counter > 4){

counter = 0;

}

if (counter < 0){

counter = 4;

}

switch (counter){

case 0:

lcd.print(L"1. ЛИНЕЙНОЕ");

break;

case 1:

lcd.print(L"2. ПЕРИОД");

break;

case 2:

lcd.print(L"3. УГЛОВОЕ");

break;

}

}

void goToOption(){

switch (counter){

case 1:

linearMovement();

counter = 0;

exitFromOption();

break;

case 2:

angleMovement();

counter = 4;

exitFromOption();

break;

case 3:

oscillation();

counter = 3;

exitFromOption();

break;

}

}

void exitFromOption(){

encoder();

delay(500);

}

void linearMovement(){

const byte stepForLinearMovement = 5; //cm

long timeForStartCounting = 0;

long distanceTravelTime = 0;

int needDistance = 0;

int pinDistanceControlStart = 2 ;

int pinDistanceControlStop = 3 ;

isNeedPreView = true;

counter = 0;

int last = counter;

delay(50);

do {

if (last != counter || isNeedPreView) {

counter = counter <= 0 ? 0 : counter;

last = counter;

needDistance = last \* stepForLinearMovement;

isNeedPreView = false;

printLinearMovementDistance(needDistance);

}

if (isAnalogButtonPressed(pinDistanceControlStart) && needDistance > 0) {

while (true) {

if (isPinHigh(pin\_Optocoupler)) {

distanceTravelTime = startLinearDistanceCalculation(needDistance);

printResultDistanceTravelTime(distanceTravelTime, needDistance);

delay(2000);

break;

}

}

}

encoder();

} while (!isEncoderButtonPressed);

}

void printResultDistanceTravelTime(long timeToPrint, int distanceToPrint){

lcd.setCursor(0, 0);

lcd.clear();

lcd.print(L"ДИСТАНЦИЯ ");

lcd.print(String(distanceToPrint));

lcd.print(L" см");

lcd.setCursor(0, 1);

lcd.print(" ");

long second = timeToPrint / 1000000;

long millisecond = timeToPrint - second \* 1000000;

millisecond = millisecond / 100;

lcd.print(String(second));

lcd.print(".");

lcd.print(String(millisecond));

lcd.print(L" сек");

}

void printLinearMovementDistance(int distanceToPrint){

if (distanceToPrint < 0) {

distanceToPrint = 0;

}

lcd.setCursor(0, 0);

lcd.clear();

lcd.print(L"РАССТОЯНИЕ");

lcd.setCursor(0, 1);

lcd.print(" ");

lcd.print(String(distanceToPrint));

lcd.print(L" см");

}

long startLinearDistanceCalculation(int needLinerDistance) {

long timeForStartCounting = 0;

long distanceTravelTime = 0;

while (needLinerDistance > 0) {

timeForStartCounting = micros();

if (isPinHigh(pin\_Impulse\_Counter)) {

needLinerDistance-- ;

distanceTravelTime = micros() - timeForStartCounting;

printResultDistanceTravelTime(distanceTravelTime, needDistance);

}

}

return distanceTravelTime;

}

void angleMovement() {

isNeedPreView = true;

isEncoderButtonPressed = false;

int lastLeftBallValues = leftBallValues[0];

int lastRightBallValues = rightBallValues[0];

do {

if (isNeedPreView) {

printAngleMovement();

isNeedPreView = false;

}

encoderLeftAngle();

encoderRightAngle();

encoder();

if (lastLeftBallValues != leftBallValues[0] || lastRightBallValues != rightBallValues[0]) {

printInfo(leftBallValues[0], rightBallValues[0]);

lastLeftBallValues = leftBallValues[0];

lastRightBallValues = rightBallValues[0];

}

delay(100);

} while (!isEncoderButtonPressed);

isNeedPreView = true;

}

void encoderLeftAngle(){

int leftPlusPin = 3;

int leftMinusPin = 2;

int step = 1;

if(isAnalogButtonPressed(leftPlusPin)) {

changeLeftBallArray(1 \* step);

}

if(isAnalogButtonPressed(leftMinusPin)) {

changeLeftBallArray(-1 \* step);

}

}

void changeLeftBallArray(int newLeftValue){

leftBallValues[0] = leftBallValues[0] + newLeftValue;

if(leftBallValues[0] < 0) {

leftBallValues[0] = 0;

}

leftBallValues[1] = leftBallValues[0];

leftBallValues[2] = leftBallValues[1];

leftBallValues[3] = leftBallValues[2];

leftBallValues[4] = leftBallValues[3];

leftBallValues[5] = leftBallValues[4];

leftBallValues[6] = leftBallValues[5];

leftBallValues[7] = leftBallValues[6];

leftBallValues[8] = leftBallValues[7];

leftBallValues[9] = leftBallValues[8];

}

void encoderRightAngle(){

int rightPlusPin = 1;

int rightMinusPin = 0;

int step = 1;

if(isAnalogButtonPressed(rightPlusPin)) {

changeRightBallArray(1 \* step);

}

if(isAnalogButtonPressed(rightMinusPin)) {

changeRightBallArray(-1 \* step);

}

}

void changeRightBallArray(int newRightValue){

rightBallValues[0] = rightBallValues[0] + newRightValue;

if(rightBallValues[0] < 0){

rightBallValues[0] = 0;

}

rightBallValues[1] = rightBallValues[0];

rightBallValues[2] = rightBallValues[1];

rightBallValues[3] = rightBallValues[2];

rightBallValues[4] = rightBallValues[3];

rightBallValues[5] = rightBallValues[4];

rightBallValues[6] = rightBallValues[5];

rightBallValues[7] = rightBallValues[6];

rightBallValues[8] = rightBallValues[7];

rightBallValues[9] = rightBallValues[8];

}

void printAngleMovement(){

lcd.setCursor(0, 0);

lcd.clear();

lcd.print(L"ОТКЛОНИТЕ ШАРЫ");

}

void printInfo(long left, long right){

lcd.clear();

lcd.setCursor(0, 0);

lcd.print(L"ЛЕВЫЙ УГОЛ ");

lcd.print(String(left));

lcd.setCursor(0, 1);

lcd.print(L"ПРАВЫЙ УГОЛ ");

lcd.print(String(right));

}

void oscillation() {

long timeForStartCounting = 0;

long countOfOscillations = 0;

int startPin = 3;

int pinImpulse = 2;

isNeedPreView = true;

counter = 0;

int last = counter;

delay(50);

do {

if (last != counter || isNeedPreView) {

counter = counter <= 0 ? 0 : counter;

last = counter;

isNeedPreView = false;

printOscillation(0, counter);

}

if (isAnalogButtonPressed(startPin) && counter > 0) {

timeForStartCounting = micros();

counter \*= 2;

while (counter != 0) {

if (isAnalogButtonPressed(pinImpulse)) {

counter--;

}

if(counter % 2 == 0) {

printOscillation(micros() - timeForStartCounting, counter/2);

} else{

printOscillation(micros() - timeForStartCounting, (counter + 1)/2);

}

delay(100);

}

isNeedPreView = true;

delay(2500);

}

encoder();

} while (!isEncoderButtonPressed);

}

void printOscillation(long time, int numberOfOscillation){

lcd.setCursor(0, 0);

lcd.clear();

lcd.print(L"КОЛЕБАНИЯ ");

lcd.print(" ");

lcd.print(String(numberOfOscillation));

lcd.setCursor(0, 1);

long second = time / 1000000;

long millisecond = time - second \* 1000000;

millisecond = millisecond / 100;

lcd.print(String(second));

lcd.print(".");

lcd.print(String(millisecond));

lcd.print(L" сек.");

}

bool isAnalogButtonPressed(byte analogPinNumber) {

return analogRead(analogPinNumber) < 300;

}

bool isPinHigh(byte pinNumber) {

return digitalRead(pinNumber) == HIGH;

}

void soundSignal(int numbersOfSound ,int timeBetweenSonds) {

const int valueOfSignal = 100;

for (int j = 0; j < numbersOfSound; j++){

analogWrite(pin\_Sound\_Signal, 100);

delay(timeBetweenSonds);

analogWrite(pin\_Sound\_Signal, 0);

delay(timeBetweenSonds);

}

}