МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Южно-Уральский государственный университет» (национальный исследовательский университет)

Высшая школа электроники и компьютерных наук

Кафедра «Информационно-измерительная техника»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.т.н., профессор

_____А.Л. Шестаков

_____2017 г.

Разработка программного обеспечения для динамического измерительного стенда

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ЮУрГУ – 12.03.01.2017.382.ПЗ ВКР

Руководитель, инженер-исследователь

_____И.С. Никитин

_____2017 г.

Автор

студент группы КЭ–436

_____Е.А. Трусова

_____2017 г.

Нормоконтролер, доцент

_____А.С. Волосников

_____2017 г.

Челябинск 2017

АННОТАЦИЯ

Трусова Е.А. Разработка программного обеспечения для динамического измерительного стенда. – Челябинск: ЮУрГУ, КЭ–436, 2017, 76 с. 27 илл., 1 табл., библиогр. список – 13 наим., 1 прил.

Данная выпускная квалификационная работа направлена на разработку программного обеспечения нижнего уровня для блока управления клапанами устройства формирования пульсирующего давления.

Для того чтобы разработать программу было необходимо провести аналитический обзор существующих архитектур микроконтроллера, выявить достоинства и недостатки каждой архитектуры и выбрать подходящую.

Приведена структурная схема устройства формирования пульсирующего давления, описана работа всех элементов данного динамического стенда.

Приведены блок-схемы алгоритма программного обеспечения и разработан код программы.

Проведен эксперимент с применением разработанной программы, рассмотрены все возможности данной программы.

Данное программное обеспечение может быть использовано на кафедре университета, для тестирования различных датчиков давления.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 6
1 ПРОВЕДЕНИЕ АНАЛИТИЧЕСКОГО ОБЗОРА СУЩЕСТВУЮЩИХ АРХИТЕКТУР МИКРОКОНТРОЛЛЕРОВ. 7
1.1 Основные типы микроконтроллеров и их архитектура 7
1.2 Достоинства и недостатки архитектур AVR, ARM и MSP430 13
1.3 Выбор подходящей архитектуры микроконтроллеров 17
2 СТРУКТУРНАЯ СХЕМА И ОПИСАНИЕ ПРИНЦИПА РАБОТЫ ДИНАМИЧЕСКОГО ИЗМЕРИТЕЛЬНОГО СТЕНДА 20
2.1 Назначение динамического измерительного стенда 20
2.2 Описание принципа работы динамического измерительного стенда 21
2.2.1 Генератор опорного сигнала 21
2.2.2 Блок клапанов и блок управления клапанами 23
2.2.3 Рабочий объем и испытуемые датчики 28
3 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И РУКОВОДСТВА ПРОГРАММИСТА 30
3.1 Алгоритмы и методы их разработки 30
3.2 Разработка алгоритма программы 31
3.2.1 Функция считывания давления и расчета погрешности 34
3.2.2 Функция для приема данных 42
4 ПРОВЕДЕНИЕ ИСПЫТАНИЙ С ПРИМЕНЕНИЕМ ПРОГРАММЫ ВЕРХНЕГО УРОВНЯ 53
ЗАКЛЮЧЕНИЕ 62
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 63
ПРИЛОЖЕНИЯ
ПРИЛОЖЕНИЕ А 64

ВВЕДЕНИЕ

Динамические измерения, т.е. измерения с использованием средств измерений в динамическом режиме, получают все большее распространение в технике и научных исследованиях. Эти измерения связаны в первую очередь с изучением закономерностей протекания физических процессов в исследуемых объектах. Поэтому роль динамических измерений особенно велика в отраслях техники и производства, для которых характерно создание новых технологических процессов и испытание новых машин, приборов и аппаратов.

Существуют различные стенды для измерения динамических процессов. Одним из таких стендов является – динамический измерительный стенд формирования пульсирующего давления. Он предназначен для того, чтобы тестировать различные датчики давления, создавая в ресивере заданное давление порционной подачей воздуха через пневматические клапана. Для управления клапанами стоит блок, который представляет собой отладочную плату с микроконтроллером, для которого и необходимо разработать программное обеспечение нижнего уровня, а также разработать алгоритмы программирования.

Целью выпускной квалификационной работы является:

Разработка программного обеспечения для управления клапанами устройства формирования пульсирующего давления.

Для достижения данной цели, были поставлены следующие **задачи**:

- Проведение аналитического обзора существующих архитектур микроконтроллеров.
- Описание принципа работы динамического измерительного стенда.
- Разработка программного обеспечения и руководства программиста.
- Проведение испытаний с применением разработанной программы.

1 ПРОВЕДЕНИЕ АНАЛИТИЧЕСКОГО ОБЗОРА СУЩЕСТВУЮЩИХ АРХИТЕКТУР МИКРОКОНТРОЛЛЕРОВ.

В данной главе необходимо провести обзор основных типов микроконтроллеров и их архитектур. Описать особенности основных архитектур, таких как AVR, ARM и MSP430. Проанализировать и сделать вывод, какой микроконтроллер использовать для разработки программного обеспечения.

1.1 Основные типы микроконтроллеров и их архитектура

В настоящее время практически ни одно современное устройство не обходится без микроконтроллеров (МК). Микроконтроллер – это специальная микросхема, предназначенная для управления различными электронными устройствами. Если назвать все типы современных микроконтроллеров, то мы поразимся, какое же их огромное количество. Однако все эти устройства можно разделить на следующие основные типы:

- Встраиваемые (embedded) 8-разрядные МК.
- 16- и 32-разрядные МК.
- Цифровые сигнальные процессоры.

Обобщенную структуру микроконтроллеров любого типа изобразим на рисунке 1.1.

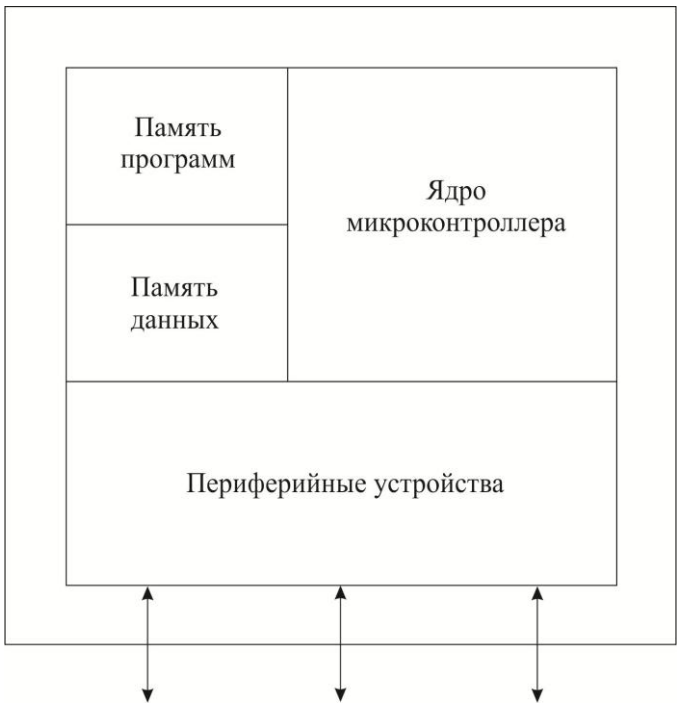


Рисунок 1.1 – Обобщенная структурная схема микроконтроллера

На базе микроконтроллерного ядра, различные фирмы-производители микросхем разрабатывают изделия, различные по номенклатуре модулей памяти и периферийных устройств, но совместимые между собой по системе команд и циклам обмена данными. Такое множество МК называется семейством микроконтроллеров.

Периферийные устройства предназначены для обеспечения связи микроконтроллеров с внешними объектами.

Выпускается очень широкая номенклатура встраиваемых микроконтроллеров. В них все необходимые ресурсы (память, устройства ввода-вывода и т.д.) располагаются на одном кристалле с процессорным ядром. С внешним миром МК общается через порты ввода вывода. Как только мы подаем напряжение и тактовые импульсы на соответствующие входы микроконтроллера, он «оживает» и можно начинать работу с данным устройством. К входам мы можем подсоединить различные датчики, их также называют рецепторами, а к выходам, например, динамики, светодиоды, двигатели и прочее (именуются эффектами).

Микроконтроллеры включают в себя достаточно много вспомогательных устройств, с помощью которых они могут включаться в реальную систему, используя минимальное количество дополнительных компонентов. Например, такие компоненты как:

- Схема начального запуска процессора (Reset).
- Генератор тактовых импульсов.
- Центральный процессор.
- Память программ и программный интерфейс.
- Средства ввода/вывода данных.
- Таймеры, фиксирующие число командных циклов.

Дополнительными возможностями более сложных встраиваемых микроконтроллеров можно назвать:

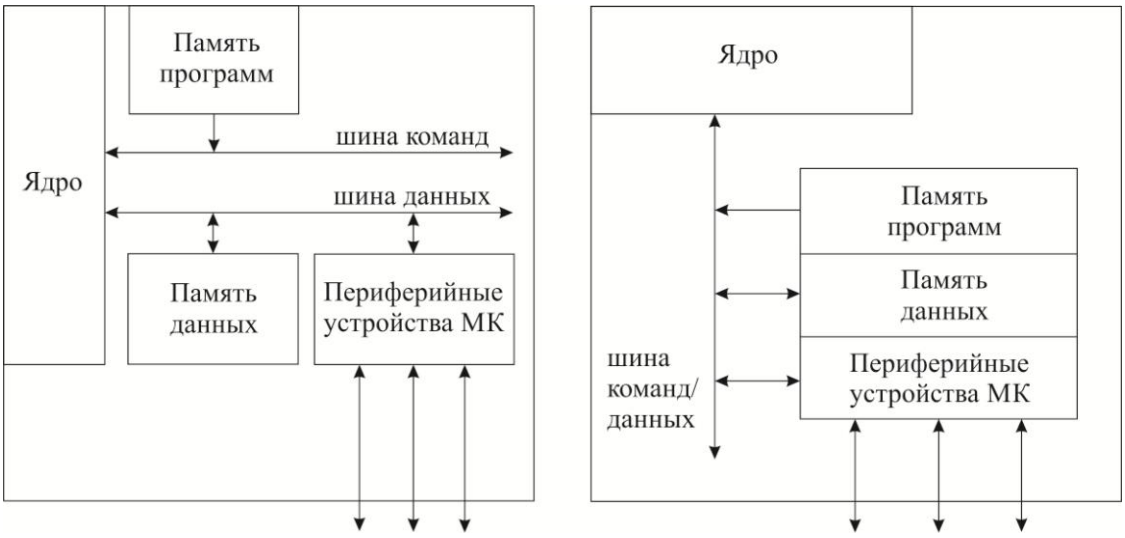
- Внутренние средства программирования памяти программ (ROM).
- Обработка прерываний от различных источников.
- Аналоговый ввод/вывод.
- Последовательный ввод/вывод (синхронный и асинхронный).
- Параллельный ввод/вывод.
- Подключение внешней памяти.

Все эти возможности упрощают процесс разработки систем на основе микроконтроллеров. Некоторые МК используют только внешнюю память, которая включает в себя память программ (ПЗУ) и небольшой объем оперативной памяти (ОЗУ). Они применяются в тех системах, где количество портов ввода/вывода небольшое, а объем памяти требуется значительный.

Цифровые сигнальные процессоры относятся к достаточно новой категории процессоров. Под их назначением понимается получение текущих данных от аналоговых систем, обработка этих данных, и формирование соответствующего отклика в реальном масштабе времени. Такие процессоры в основном используются в составе устройств управления внешним оборудованием.

Существует две группы системы команд микроконтроллеров: CISC (Complex Instruction Set Computer) и RISC (Reduce Instruction Set Computer). Разделение на эти группы происходит в зависимости от числа используемых кодов операций. CISC – сложная система команд, а RISC – сокращенная. Основная идея сокращенной архитектуры – это тщательная подборка определенных комбинаций кодов операций, которые выполнялись бы за один такт тактового генератора. Таким образом, обеспечивается резкое упрощение аппаратной реализации центрального процессора и повышение его производительности. Таким образом, одной команде CISC соответствует несколько команд RISC.

В микроконтроллерах, выполненных на основе сокращенной (RISC) архитектуры, программы и данные располагаются в независимых блоках памяти с различными методами доступа (рисунок 1.2, а). В микроконтроллерах, выполненных на основе CISC архитектуры, программы и данные могут располагаться в общем блоке памяти, и для обращения к ним используется единый метод доступа (рисунок 1.2, б).



а) б)

Рисунок 1.2 – RISC и CISC архитектуры

Каждый микроконтроллер обладает определенным набором команд (также системой команд), который характеризуется списком команд и их форматом. Этот список команд представляет собой набор операций, выполнение которых предусмотрено на процессоре данного микроконтроллера. В списке команд любого микроконтроллера можно выделить четыре основных группы операций[1]:

- Операции передачи данных (между ячейками памяти или другими элементами микроконтроллера).
- Арифметические операции (сложение, вычитание, умножение, деление).
- Логические операции («И», «ИЛИ», «исключающее ИЛИ», инверсия).
- Операции передачи управления (переход по заданному адресу, переход по условию равенства или неравенства операндов, переход на подпрограмму и возврат из нее).

В настоящее время грань между RISC и CISC архитектурой стирается. Например, микроконтроллеры семейства AVR производимые фирмой Atmel имеют такую систему команд, которая включает в себя 120 инструкций, это соответствует архитектуре CISC. В то же время большая часть инструкций выполняется за один такт тактового генератора, что является признаком архитектуры RISC. Сегодня принято считать, что признаком RISC архитектуры является выполнение команд за один такт тактового генератора. Таким образом, в настоящее время число команд уже не имеет никакого значения.

Что касается памяти микроконтроллеров, то можно выделить три основных вида:

- Память программ.
- Память данных.
- Регистры МК.

Память программ представляет собой постоянную память, которая предназначена для хранения программного кода и констант. Эта память не изменяет содержимого в процессе выполнения программы.

Память данных предназначена для хранения переменных в ходе выполнения программы.

Что касается регистров микроконтроллера, то этот вид памяти включает внутренние регистры процессора и внешние регистры, служащие для управления устройствами ввода/вывода данных.

Для того чтобы хранить программы, используется энергонезависимая память (ROM) – это такая память, которая сохраняет свое содержимое даже после отключения питания МК. Наличие такой памяти необходимо, в виду того, что микроконтроллеры не содержат каких-либо устройств массовой памяти, с которых выполняется загрузка программ в компьютерах. Программы все время хранятся в микроконтроллерах.

В процессе выполнения, программы считываются из энергонезависимой памяти, а блок управления отвечает за обеспечение декодирования программ и выполнение необходимых операций. Содержащиеся в памяти программ, данные, не могут изменяться (перепрограммироваться) в то время когда выполняется программа. Поэтому функциональное назначение микроконтроллера не может измениться, пока содержимое его памяти программ не будет стерто (если это возможно) и перепрограммировано (заполнено новыми командами).

Оперативно запоминающее устройство (ОЗУ, или на англ. RAM) − используется для хранения данных. Читать и записывать данные в ОЗУ можно неограниченное количество раз, в то же время, когда отключается питание МК, вся информация стирается.

Область применения микроконтроллеров огромна. Они применяются в основном во встроенных системах, в игрушках, в различных станках, в домашней технике и автоматике – там, где нужна не мощность процессора, а баланс между ценой и достаточным функционалом.

Именно поэтому самые старые типы микроконтроллеров применяются и в настоящее время. Они используются в различных системах полива газонов или в системах автоматического открывания дверей, даже интегрируются в систему

«умный дом». При этом существуют и более мощные МК, способные выполнять сотни миллионов операций в секунду. Но у них и задачи соответствующие. Таким образом, разработчик сначала оценивает задачу, а уже потом выбирает подходящее под данную задачу «железо».

На сегодняшний день существует огромное многообразие микроконтроллеров, выпускаемых различными компаниями. Наибольшей популярностью у разработчиков пользуются 8-битные микроконтроллеры PIC фирмы Microchip Technology и AVR фирмы Atmel, 16-битные MSP430 фирмы Texas Instruments, а также 32-битные микроконтроллеры, архитектуры ARM, которую разрабатывает фирма ARM Limited и продаёт лицензии другим фирмам для их производства.

1.2 Достоинства и недостатки архитектур AVR, ARM и MSP430

Можно сказать, что микроконтроллер – это маленький компьютер, который имеет свой центральный процессор, память, а также различную периферию (порты ввода/вывода, таймеры, контроллеры прерываний, генераторы разных импульсов и даже аналоговые преобразователи). Всего этого не перечислить. Как нельзя перечислить все применения микроконтроллеров.

Но, если все упростить, то основной функцией микроконтроллера является, так сказать, «дрыганье ножками». То есть у него есть несколько выводов и на этих выводах он может выставить либо 1 (высокий уровень напряжения), либо 0 (низкий уровень напряжения). Почти все современные микроконтроллеры имеют Аналогово Цифровой Преобразователь (АЦП) — это такое устройство, похожее на вольтметр. Он позволяет отслеживать, какой уровень напряжения у него на входе 0 или 1, а также замерить величину этого напряжения от 0 до опорного (опорное напряжение обычно равно напряжению питания) и представить его в виде числа от 0 до 255 (или 1024) в зависимости от разрядности АЦП.

Как я уже ранее говорила, микроконтроллеров существует огромное количество. Практически каждая фирма по производству радиокомпонентов

выпускает свой собственный микроконтроллер. Но и в этом многообразии есть свой порядок. Микроконтроллеры делятся на семейства, и сейчас я опишу такие семейства как ARM и AVR.

Advanced RISC Machine или как мы привыкли ARM − это целая архитектура для семейства 32- и 64-битных процессоров производства компании ARM Limited. В 2007 году практически во всех мобильных телефонах мира (98%) стоял минимум один процессор ARM. Эти процессоры широко распространены на рынке портативной техники и особенно в сфере мобильных устройств, а все почему, да потому что они обладают низким энергопотреблением.

К концу 2012 года новые модели ARM-процессоров Apple и Samsung достигли производительности бюджетных процессоров Intel для ноутбуков. В частности, планшет Samsung Nexus 10 показал рейтинг производительности 2348 баллов, в то время как бюджетный процессор Intel Core Duo в ноутбуке AppleMacAir показывает рейтинг только 1982 балла.

Самой основной особенностью архитектуры ARM является то, что микроконтроллеры этого семейства обладают малым энергопотреблением, что позволяет использовать их в устройствах, работающих на батарейках.

Одним из существенных отличий архитектуры ARM от других архитектур является так называемая предикация − возможность условного исполнения команд. «Условное исполнение» в данном случае значит, что команда будет выполнена или проигнорирована в зависимости от текущего состояния флагов (условий) состояния процессора. Это было достигнуто добавлением в коды особого 4-битового поля (предиката). Одно из его значений зарезервировано на то, что инструкция должна быть выполнена, безусловно, а остальные кодируют то или иное сочетание условий (флагов). С одной стороны это сократило число бит, доступных для кодирования смещения в командах обращения к памяти, но с другой − позволило избавляться от инструкций ветвления при генерации кода для небольших if-блоков.

Также одним из основных семейств является AVR – семейство восьмиразрядных микроконтроллеров фирмы Atmel. Это значит, что большинство операций процессоры производят с восьмиразрядными двоичными числами. Для обработки шестнадцатиразрядных чисел некоторые внутренние регистры могут объединяться попарно. Таким образом, каждая такая пара может работать как один шестнадцатиразрядный регистр. Появилось это семейство в 1996 году. Его удачное сочетание RISC-архитектуры, которое обеспечивает высокую производительность с широким набором команд и энергонезависимой памятью для программ, очень быстро поместило микроконтроллеры семейства AVR на первые позиции. Эти контроллеры имеют RISC архитектуру и выполняют одну команду за один такт. Поэтому быстродействие контроллеров может достигать 1 миллиона операций в секунду при тактовой частоте 1 МГц. Одновременно с появлением этих микроконтроллеров были созданы программные продукты и технические средства, поддерживающие разработку программ для микроконтроллеров. Например, такие как AVR Studio – фирменный пакет фирмы Atmel и отладочные платы в виде стартовых наборов разработчика. Вместе они образуют единую платформу, на которой можно проводить разработку и отладку различных приложений. Микроконтроллеры этого семейства имеют достаточно высокое быстродействие и низкий ток потребления. Микроконтроллеры AVR набиты различной периферией, особенно контроллеры подсемейства ATMega. Периферийные они по отношению к центральному процессорному устройству микроконтроллера. Но находятся они также внутри микросхемы. Одно из преимуществ микроконтроллеров AVR является то, что их очень легко прошивать и для этого не нужно какое-либо сложное оборудование и специальные программаторы. Для этого необходимо только лишь иметь компьютер со специальным портом и несколько проводков. Простота освоения этого микроконтроллера и сделала его таким популярным по всему миру. Однако такие МК имеют и другие достоинства, такие как:

1. Доступность в России. Эти контроллеры очень популярны и поэтому наши торговцы их охотно возят.

2. Низкая цена.

3. Очень много периферии. Конечно для серийного устройства это скорее недостаток. Но все зависит от того, какая задача перед тобой стоит.

4. Единое ядро. МК имеют единую систему команд.

5. Обширная система команд. У AVR примерно 130 команд. Множество команд обеспечивает возможность решения сложных задач с наименьшими промежуточными действиями.

6. Наличие бесплатных кроссплатформенных компиляторов Си.

7. Программы содержатся в электрически перепрограммируемой постоянной памяти программ FLASH ROM. Эта память может быть перепрограммирована до 1000 раз. Это облегчает настройку и отладку системы. Кроме этого, возможность внутрисхемного программирования позволяет не вынимать МК из целевой схемы в процессе программирования, что значительно ускоряет процесс разработки систем на основе этих МК.

Еще одной особенностью семейства микроконтроллеров AVR является то, что система команд всего семейства совместима при переносе программы со слабого МК на более мощный [2].

Все микроконтроллеры AVR имеют гарвардскую RISC-архитектуру, которая предполагает разделение памяти программ и данных. Используемые при этом средства адресации позволяют создавать эффективные программы с высоким быстродействием [3].

Еще одно семейство, которое будем рассматривать – это MSP430, семейство 16-ти разрядных микроконтроллеров фирмы Texas Instruments. Ядро у микроконтроллера MSP430 16-ти битное. Система команд ортогональная, с разнообразными способами адресации. Ортогональность системы команд означает, что в любой команде может использоваться любой способ адресации операнда: константа, прямой из регистра или памяти, косвенный и т.п.

При создании данного семейства, основной упор делали на снижение энергопотребления, но в настоящее время уже достаточно много электронной техники, которые превосходят данное семейство.

Ключевым отличием и «визитной карточкой» семейства MSP430 является возможность тактировать любой модуль периферии асинхронно от ядра. В основном у микроконтроллеров периферия синхронна с ядром (за исключением отдельных специальных узлов). Такая особенность позволяет гибко управлять скоростью (а значит, и потреблением) каждого модуля.

1.3 Выбор подходящей архитектуры микроконтроллеров

Итак, теперь мы знаем, какие преимущества и недостатки существуют у микроконтроллеров с архитектурами AVR , ARM и MSP430. Теперь нужно решить, какую же архитектуру микроконтроллеров использовать в поставленной нами задаче. Как говорилось ранее, сначала разработчик оценивает задачу, а уже потом выбирает «железо», подходящее под данную задачу.

Для того чтобы упростить поиск нужного нам микроконтроллера, проведен анализ характеристик уже готовых отладочных плат, на базе этих микроконтроллеров (таблица 1). Главными характеристиками, которые необходимы для работы, являются – количество аналоговых входов (необходимо 2), количество цифровых ШИМ выходов (необходимо минимум 8), наличие внешнего интерфейса USB, наличие EEPROM памяти.

Таблица 1 – Характеристики отладочных плат

	Texas Instruments (MSP-EXP430FR5969)	Atmel (Arduino Mega 2560)	Миландр (LDM-START-K1986BE92QI)	Analog Devices (ADSP-BF506FE2-KIT Lite)	STMicroelectronics (stm32-p152)
Аналог. вх.	2	16	16	3	24
Цифр. вых.	16	54	43	37	14
USB	да	да	да	да	да
Тактовая частота, МГц	16	16	80	400	24
ОЗУ, кб	0,128	8	32	4	16
EEPROM, кб	нет	4	нет	нет	4
Напряжение питания, В	3,6	6-20	5	5	3,6
Цена, руб	1880	1170	4000	14000	2340

Проанализировав таблицу 1, сравнивая нужные нам характеристики, можно сделать вывод, что единственной подходящей платой является – Arduino Mega 2560 фирмы Atmel, так как у нее достаточно цифровых выходов, к тому же она является самой дешевой из представленных вариантов.

Arduino Mega 2560 – это устройство, на основе микроконтроллера ATmega2560 (архитектура AVR). В его составе: 54 цифровых входа/выхода (из которых 15 могут использоваться в качестве ШИМ-выходов), 16 аналоговых входов, 4 UART, кварцевый резонатор на 16 МГц, разъем USB, разъем питания, разъем ICSP для внутрисхемного программирования и кнопка сброса. Arduino Mega 2560 имеет Flash-память 256 кб из которых 8 кб используются загрузчиком. Энергонезависимая память EEPROM (память, куда мы будем записывать данные с датчиков) составляет 4 кб. Данную плату можно запитать как от USB так и от внешних источников питания, например, аккумулятор/батарея.

Вывод по главе 1:

1. Были проанализированы основные особенности архитектур AVR, ARM и MSP430.

2. Выбрана наиболее подходящая архитектура под поставленную перед нами задачу, а точнее готовая отладочная плата Arduino Mega 2560, на основе микроконтроллера ATmega2560 (архитектура AVR). Данная плата имеет 54 цифровых входа/выхода (15 из которых ШИМ), 16 аналоговых входа, 4 кб EEPROM и 8 кб ОЗУ. Является наименее дорогостоящей, чем отладочные платы других производителей.

3. Далее необходимо рассмотреть принцип работы динамического измерительного стенда, в том числе работу блока клапанов данного устройства.

2 СТРУКТУРНАЯ СХЕМА И ОПИСАНИЕ ПРИНЦИПА РАБОТЫ ДИНАМИЧЕСКОГО ИЗМЕРИТЕЛЬНОГО СТЕНДА

В первой главе был произведен обзор существующих архитектур микроконтроллеров, и выбрана подходящая отладочная плата, с помощью которой будет разрабатываться программное обеспечение нижнего уровня для динамического измерительного стенда.

В данной главе необходимо рассмотреть назначение динамического измерительного стенда и проанализировать принцип его работы.

2.1 Назначение динамического измерительного стенда

Динамический измерительный стенд разрабатывается для тестирования различных датчиков давления, с целью проведения экспериментов в ходе учебного процесса в университете.

Прежде чем описать принцип работы этого стенда, рассмотрим, что же такое динамические измерения, которые будут проводиться на стенде.

По характеру изменения измеряемой величины измерения делятся на статические и динамические. Статические измерения имеют место тогда, когда измеряемая величина практически постоянна. А динамические измерения связаны с такими величинами, которые в процессе измерений претерпевают те или иные изменения.

Измеряемые физические параметры обычно изменяются с течением времени, поэтому для оценки точности измерений необходимо знать, как зависит погрешность измерений от динамических характеристик измеряемой величины, то есть какова динамическая компонента погрешности измерений. Погрешности, которые появляются из-за влияния скоростей изменения измеряемой величины или процесса, называются динамическими погрешностями [10].

2.2 Описание принципа работы динамического измерительного стенда

Для того чтобы описать принцип работы динамического измерительного стенда, необходимо привести функциональную схему этого стенда, которая была исходными данными к выпускной квалификационной работе (рисунок 2.1).

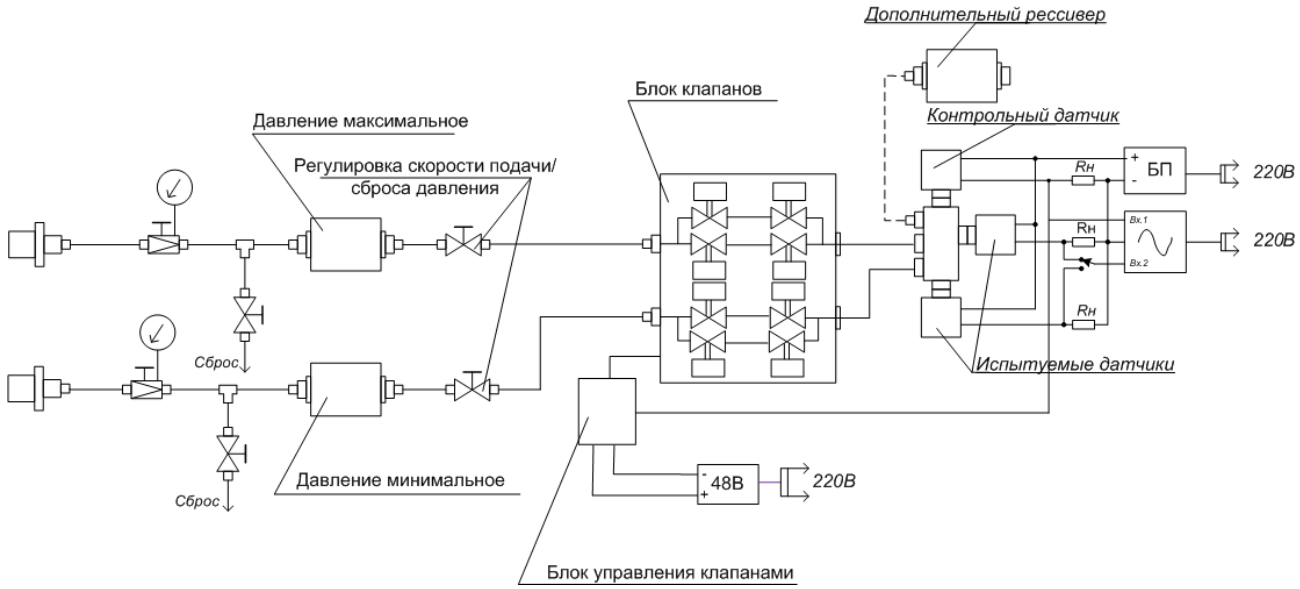


Рисунок 2.1 – Динамический измерительный стенд

Данную схему можно разделить на функциональные элементы:

1. Генератор опорного сигнала.
2. Блок клапанов и блок управления клапанами.
3. Ресивер и испытуемые датчики.

Рассмотрим подробнее выделенные элементы стенда.

2.2.1 Генератор опорного сигнала

Генератор опорного сигнала отдельно показан на рисунке 2.2. Включает в себя источник верхнего давления и источник нижнего давления.

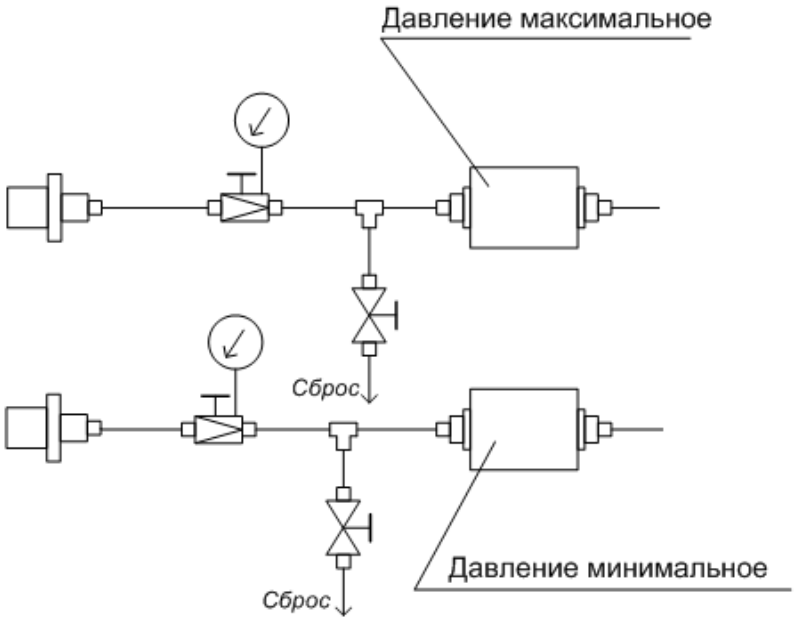


Рисунок 2.2 – Генератор опорного сигнала

Источник верхнего давления через редуктор, работающий от сети, заполняет ресивер воздухом и устанавливает в нем определенное давление. Перед редукторами стоят два клапана, для сброса избыточного давления.

Второй ресивер устанавливает нижнее давление. В частности может быть просто атмосферным давлением, в зависимости от того какую форму выходного сигнала нам необходимо получить.

Вместе они формируют давление синусоидальной формы.

Далее стоят два регулируемые вентиля (рисунок 2.3), у которых очень плавно можно менять расход вручную, за счет игольчатого клапана.

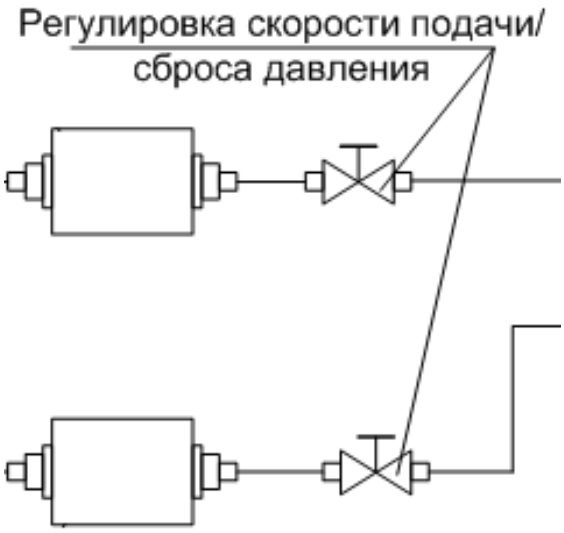


Рисунок 2.3 – Регулируемые вентили

2.2.2 Блок клапанов и блок управления клапанами

Блок клапанов изображен на рисунке 2.4. Клапана расположены таким образом, чтобы можно было расширить возможности их регулирования.

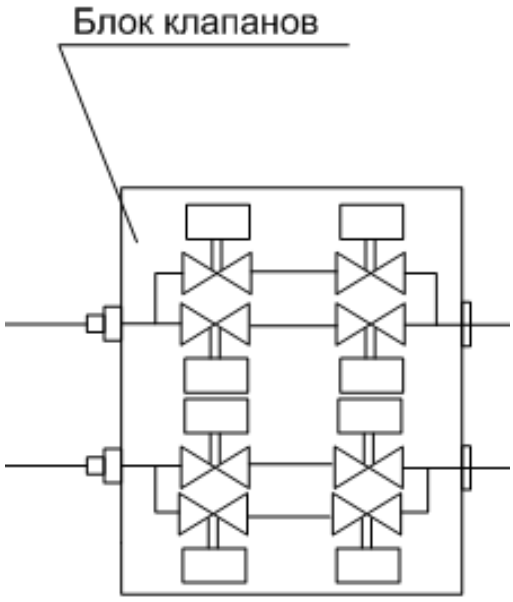


Рисунок 2.4 – Блок клапанов

Клапана, которые в настоящее время можно найти на рынке, могут открываться на время не менее порядка 0,5 мс и частотой до 1 кГц, т.е. быстрее они работать не умеют.

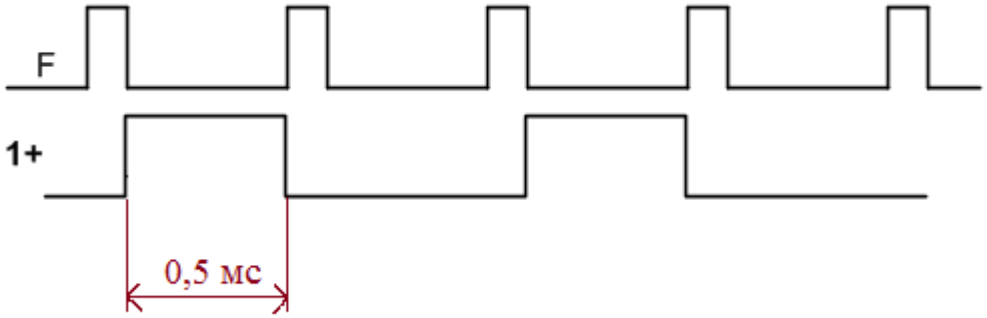


Рисунок 2.5 – Время открытия клапана

Из рисунка 2.5 видно, что клапан открыт 0,5 мс и примерно столько же закрыт. То есть сделать время подачи давления или сброса меньше этих значений мы теоретически не можем. На практике получается, что за счет задержки в клапане, время открытия получается больше чем половина миллисекунды. Небольшие порции воздуха необходимо подавать маленькими частями, иначе получается большое перерегулирование. Поэтому два клапана поставлены последовательно и открываются они с задержкой друг относительно друга (рисунок 2.6).

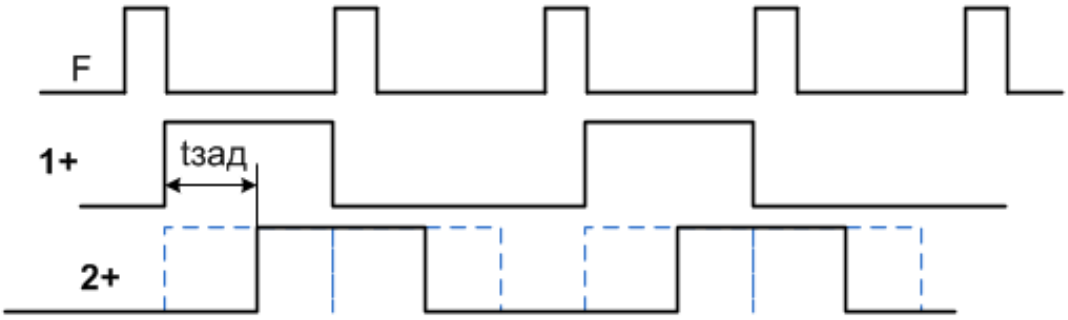


Рисунок 2.6 – Задержка между открытием клапанов

Понятие перерегулирования относится к прямым показателям качества регулирования переходного процесса и означает отклонение регулируемой величины в процессе регулирования от установившегося значения.

Поэтому поставлено два клапана и открываются они с задержкой друг относительно друга. Величина задержки задается в блоке управления клапанами в зависимости от того какую имеем производную функции. Если производная функции меняется быстро, необходимо открыть клапан на большую величину, в связи с этим получается минимальная задержка и клапана открываются одновременно. Если скорость изменения производной маленькая, клапана открываются на небольшую величину, время задержки увеличивается и увеличивается реальное время открытия клапанов. Таким образом, можно регулировать время открытия клапанов от максимального значения пропускной способности клапанов до, практически, нуля.

Для удаления паузы между открытием клапанов поставлена вторая цепочка клапанов (рисунок 2.7).

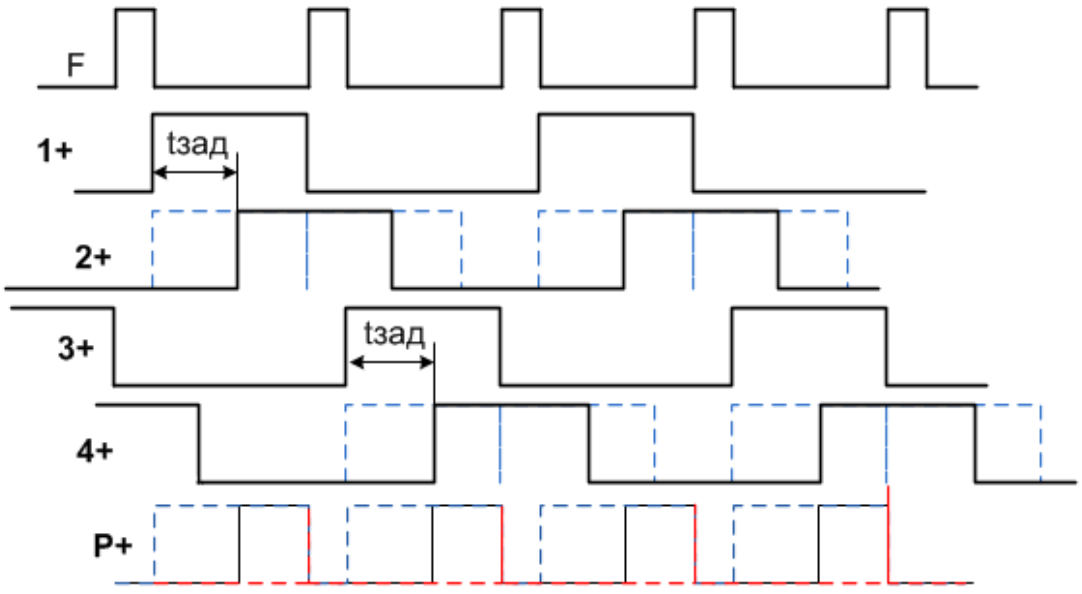


Рисунок 2.7 – Время открытия клапанов

В данном случае за счет такой конструкции блока клапанов, тракт подачи давления практически все время, с небольшими за счет задержек промежутками, открыт.

Аналогичным образом работает тракт на сброс давления.

Для управления клапанами стоит блок, изображенный на рисунке 2.8.

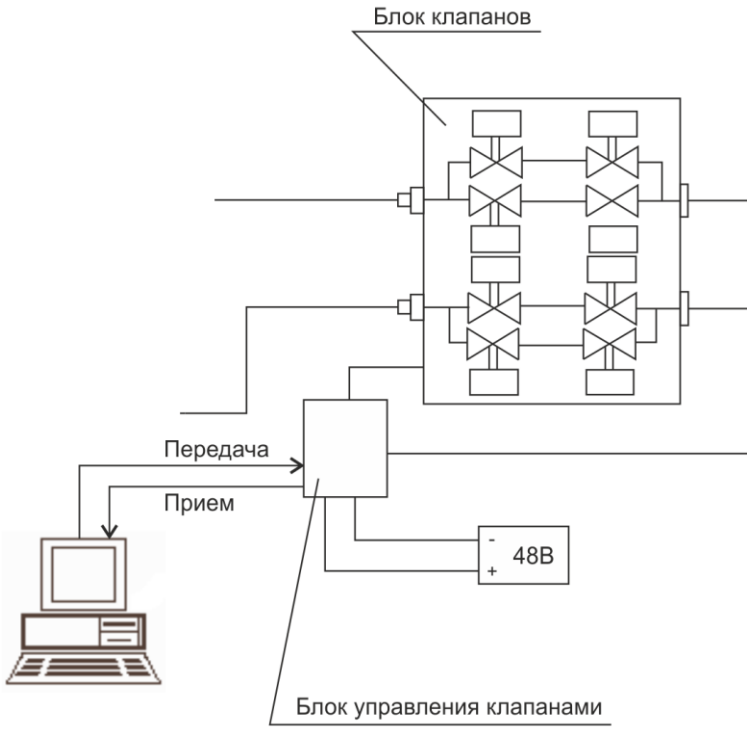


Рисунок 2.8 – Блок управления клапанами

Проводя аналитический обзор существующих архитектур микроконтроллеров, был сделан вывод, что для реализации поставленной задачи нам подходит отладочная плата Arduino Mega 2560 фирмы Atmel. Таким образом, в качестве блока управления клапанами в данном стенде будем использовать выбранную отладочную плату Arduino Mega 2560 с микроконтроллером ATmega 2560. Структурная схема блока управления клапанами и блока клапанов изображена на рисунке 2.9.

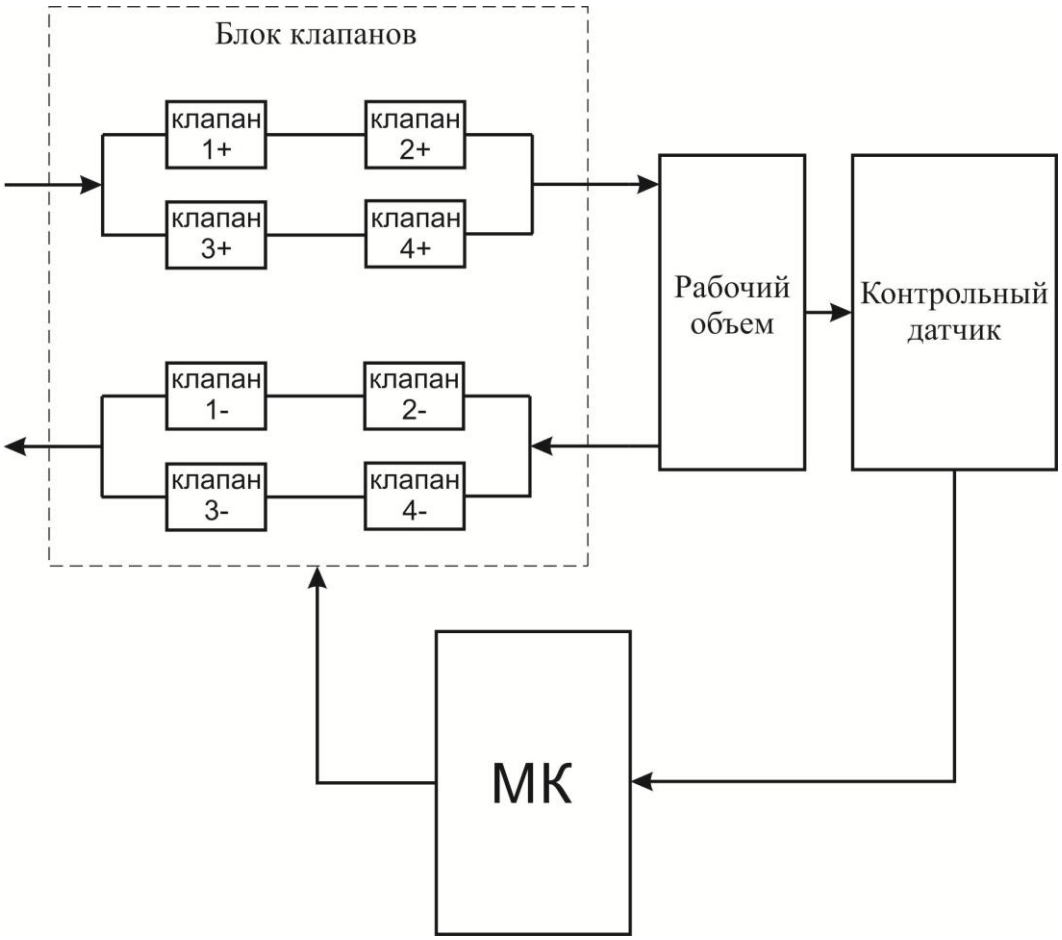


Рисунок 2.9 – Структурная схема блока управления клапанами и блока клапанов

Микроконтроллер считывает данные с измерительных датчиков и оправляет их на персональный компьютер через последовательный порт, затем сравнивает полученные данные с входным сигналом и подает команду на открытие или закрытие клапанов. Так же рассчитывается время, на которое открываются клапана, и задержка между открытием клапанов. Тем самым производиться регулировка уровня давления в ресивере.

Так же функцией микроконтроллера, выступающего в качестве блока управления клапанами, является расчет погрешности сформированного давления и отправка этих данных на персональный компьютер через последовательный порт.

2.2.3 Рабочий объем и испытуемые датчики

Рабочий объем представляет собой ресивер с подключенным к нему дополнительным ресивером, испытуемыми и контрольным датчиками. К объему так же подключен осциллограф для контроля выходного сигнала давления (рисунок 2.10).

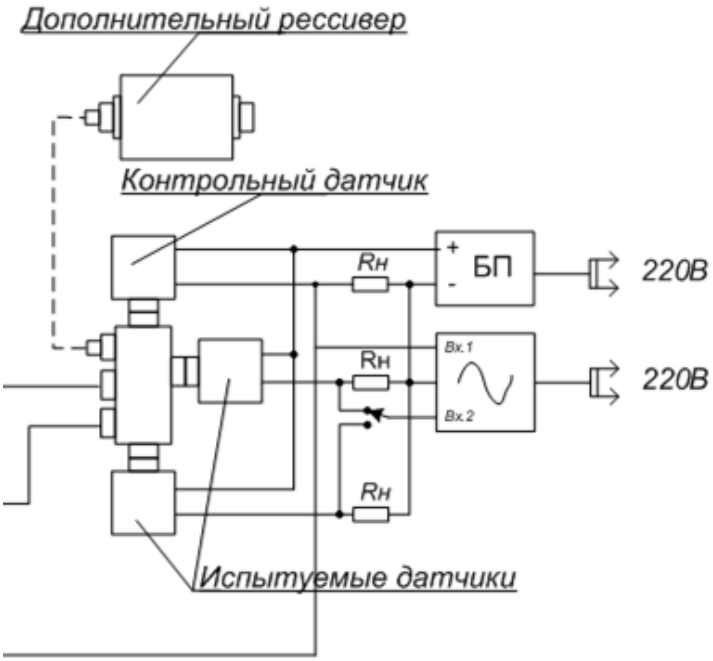


Рисунок 2.10 – Рабочий объем и испытуемые датчики

Вывод по главе 2:

1. В данной главе определено назначение динамического измерительного стенда, которое заключается в тестировании различных датчиков давления, с целью проведения экспериментов в ходе учебного процесса в университете.

2. В данной главе описаны составные части динамического измерительного стенда, а также принцип работы стенда.

3. Далее необходимо разработать алгоритм программы для блока управления клапанами данного стенда, а также написать код программы.

3 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И РУКОВОДСТВА ПРОГРАММИСТА

В данной главе будет рассмотрено, что такое алгоритмы и методы их разработки. Приведен алгоритм разрабатываемого программного обеспечения и описание основных функций алгоритма. А также будет представлено руководство программиста.

3.1 Алгоритмы и методы их разработки

Для начала выясним, что такое алгоритм программирования. Можно представить алгоритм программирования как рецепт, описывающий точные шаги, необходимые компьютеру для решения проблемы или достижения цели. Мы все видели рецепты блюд, они перечисляют необходимые ингредиенты и набор шагов для того, чтобы сделать описанную еду. Такой у них алгоритм. В компьютерном жаргоне слово для рецепта − это процедура, а ингредиенты называются входными данными. Компьютер смотрит на процедуру, следует за ней к письму, и вы видите результаты, которые называются выходами. Алгоритм программирования описывает, как делать что-то, и компьютер будет делать это именно так каждый раз. После того, как алгоритм программы написан, его можно будет преобразовать в язык, понятный компьютеру, то есть код.

Важно отметить, что алгоритм программирования не является компьютерным кодом. Он написан простым русским языком (или тем, на котором говорит программист). У этого алгоритма есть начало, середина и конец. Он включает только то, что нужно для выполнения задачи. В нем нет ничего неясного, часто называемого двусмысленным в компьютерном жаргоне, он написан так, чтобы каждый читающий понимал, о чем идет речь и как решить данную задачу.

Алгоритм − это процедура или формула решения проблемы, основанная на проведении последовательности заданных действий. Компьютерную программу можно рассматривать как сложный алгоритм. В математике и информатике алгоритм обычно означает небольшую процедуру, которая решает повторяющуюся задачу.

Алгоритмы широко используются во всех областях информационных технологий. В программировании алгоритм представляет собой набор четко определенных команд в последовательности для решения программы.

Качества хорошего алгоритма:

1) Входы и выходы должны быть точно определены.

2) Каждый шаг алгоритма должен быть ясным и недвусмысленным.

3) Алгоритм должен быть наиболее эффективным среди множества различных способов решения проблемы.

4) Алгоритм не должен иметь компьютерного кода. Вместо этого алгоритм должен быть написан таким образом, чтобы его можно было использовать в похожих языках программирования.

Перед разработкой программного обеспечения, программисту необходимо нарисовать блок-схему.

Блок-схема представляет собой – графическое или символическое представление алгоритма.

Цель всех блок-схем – это сообщить, как работает или должен работать разработанный алгоритм.

3.2 Разработка алгоритма программы

Первым делом перед разработкой программного обеспечения, необходимо разработать алгоритм работы программы и представить его в виде блок-схемы.

На рисунке 3.13 представлен общий алгоритм работы программы.

Рисунок 3.13 – Общий алгоритм работы программы

Общий алгоритм представляет собой объявление переменных и бесконечный цикл, который осуществляет вызов двух функций: функция считывания давления и расчета погрешности и функция для приема данных.

В блоке 2 осуществляется объявление таких переменных как: максимальное давление, минимальное давление, среднее давление, интервал времени, частота, сдвиг, амплитуда пульсации, допуск, расход воздуха, величина перерегулирования, номер команды, сформированное давление, погрешность формирования давления, номер отсчета, отсчет времени, входная синусоида.

```
float AveragePressure = 200; // Среднее значение давления
float SinIn = 0; // Значение входной синусоиды
float oldSinIn = 0;
float oldoldSinIn = 0;
float SinTest = 0; // Значение образцевой синусоиды
float oldSinTest = 0;
float deltaTime = 0.8; // Интервал времени
float MaxPressure = 400; // Максимальное давление
float MinPressure = 0; // Минимальное давление
float Frequence = 50; // Частота дискретизации
float Shift = -0.5; // Сдвиг
float Um = 0.3; // Амплитуда пульсации
int i = 0;
float Delta=100; // Погрешность
float Pperemen = 0; // Сформированное давление
float oldPperemen = 0;
float limit = 0.01; // Допуск
float Uv = 2.5; // Расход воздуха
float Vp = 0.015; // Величина перерегулирования

float MaxSinIn = 0;
```

```
float ValveStatePlus = 0; // Состояние клапана на подачу (открыт/закрыт)
float OpenTimePlus = 0; // Время на которое открывается клапан
float DelayTimePlus = 0; // Задержка между открытием клапанов
float DelayTimeMinus =0; // Задержка между открытием клапанов
float ValveStateMinus = 0; // Состояние клапана на сброс (открыт/закрыт)
float OpenTimeMinus = 0; // Время на которое открывается клапан
float comNumber = 0; // Номер команды для передачи
int comNum = 0; // Номер команды для приема

float timeArray = 0;
float oldtimeArray = 0;

// Массив для отправки значений сформированного давления
union {float fvalue[2]; byte bvalue[8];} fbvalue;
// Массив для отправки значений погрешности
union {float fpress[2]; byte bpress[8];} fbpress;
// Массив для отправки значений образцового давления
union {float fSinTest; byte bSinTest[4];} fbSinTest;
```

3.2.1 Функция считывания давления и расчета погрешности

В данной функции происходит считывание значения давления из ресивера, расчет погрешности формирования давления и отправка данных на персональный компьютер для дальнейшей обработки и построения графиков.

Алгоритм функции представлен на рисунке 3.14.

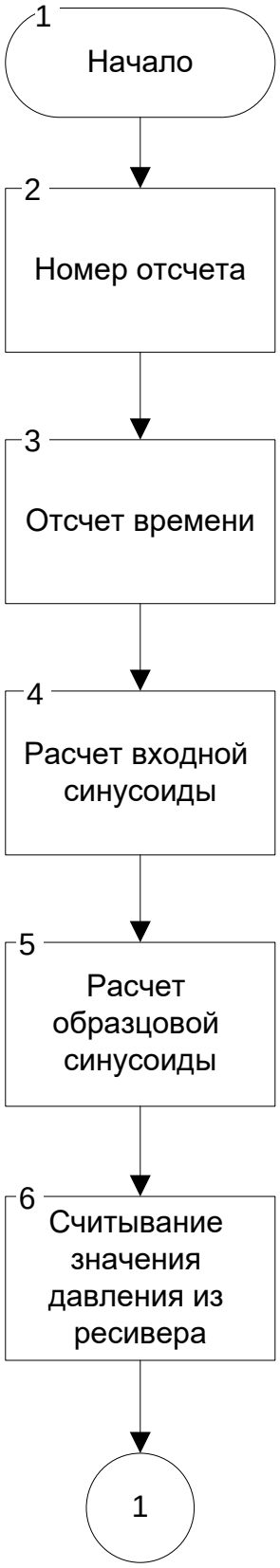


Рисунок 3.14 – Алгоритм работы функции считывания давления и расчета погрешности

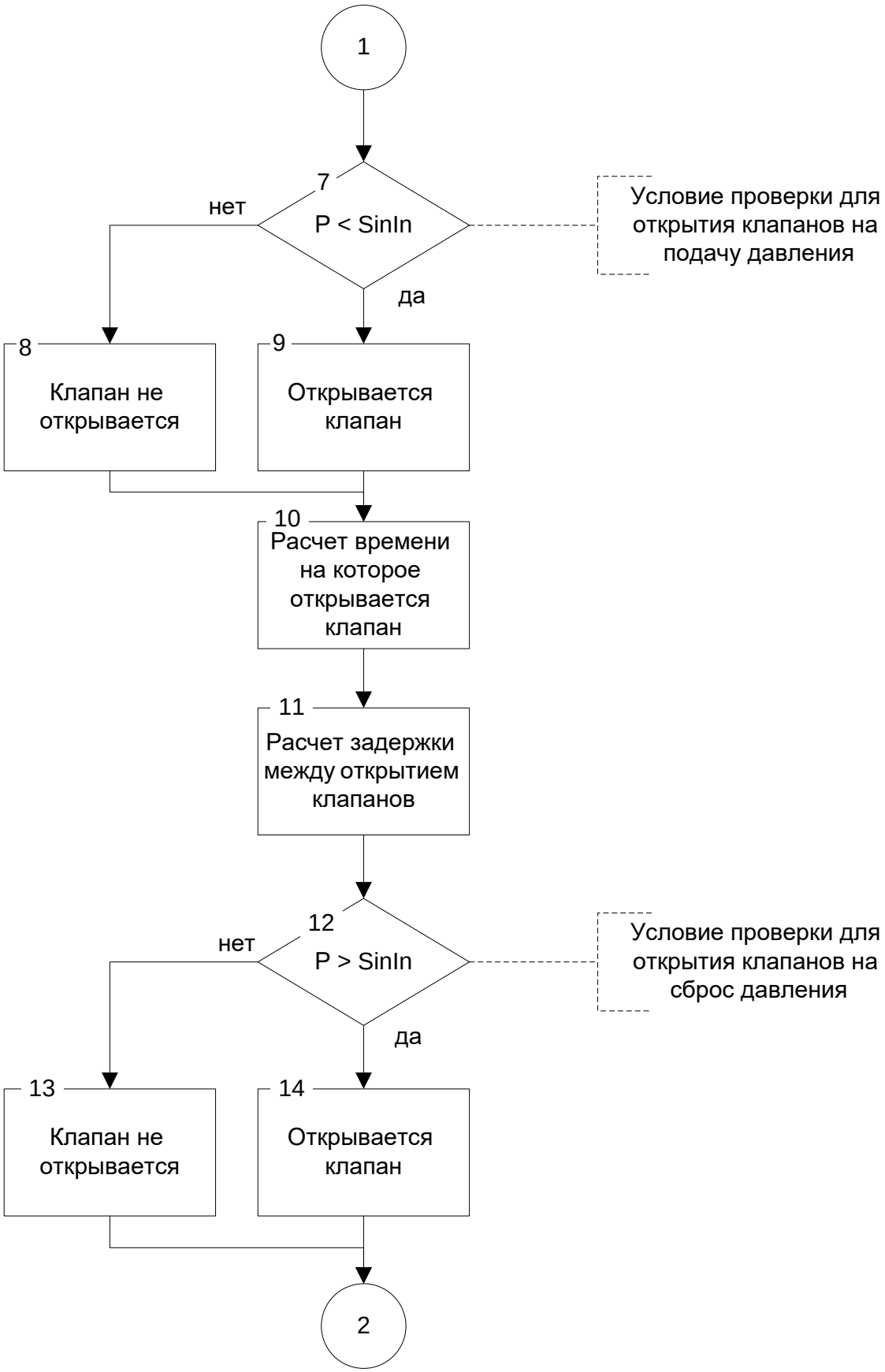


Рисунок 3.14 – Продолжение алгоритма работы функции считывания давления и расчета погрешности

Рисунок 3.14 – Окончание алгоритма работы функции считывания давления и расчета погрешности

Далее рассмотрим более подробно работу функции.

Первым делом вычисляется входная и образцовая синусоиды. Также для их расчета необходим номер отсчета и отсчет времени.

```
// Записываем предыдущее значение отсчета времени
oldtimeArray = timeArray;
// Расчет текущего значения отсчета времени
timeArray = (i - 2)*deltaTime;
// Предыдущие значения входной синусоиды, нужны для нахождения максимальной разницы между значениями синусоиды
oldoldSinIn = oldSinIn;
oldSinIn = SinIn;
// Расчет значения входной синусоиды
SinIn = AveragePressure + Um * AveragePressure * sin(2*PI*Frequence/1000 * timeArray);
// Записываем предыдущее значение образцовой синусоиды
oldSinTest = SinTest;
// Расчет значения образцовой синусоиды
SinTest = AveragePressure + Um * AveragePressure * sin(2*PI*Frequence/1000 * timeArray - Shift/6.28);
```

Далее происходит считывание значения давления из ресивера и его преобразование.

Для чтения с аналогового входа в Arduino есть встроенная функция analogRead(), возвращающая значение от 0 до 1023. Значение 0 относиться к минимальному значению давления заданному с компьютера, а 1023 – к максимальному значению давления также заданному с компьютера [11].

```
// Записываем предыдущее значение давления
oldPperemen = Pperemen;
```

```
// Считывание значения давления с аналогового входа отладочной платы
Pperemen = (analogRead(pin))/2.5575;
```

Затем происходит сравнение считанного и рассчитанного значений давления. Если считанное значение меньше рассчитанного, то необходимо открыть клапана на подачу давления. Если считанное значение больше рассчитанного, то необходимо открыть клапана на сброс давления.

```
// Условие проверки для открытия/закрытия клапанов на подачу
if ((oldPperemen) < (SinIn - AveragePressure * limit))
{
    // Клапана на подачу давления открываются
      ValveStatePlus = 1;
 }
else
{
    // Клапана на подачу давления закрываются
      ValveStatePlus = 0;
}
// Условие для нахождения максимальной разницы между значениями входной синусоиды
if ((oldSinIn - oldoldSinIn) < (SinIn - oldSinIn))
{
      MaxSinIn = SinIn - oldSinIn;
}
// Условие проверки для открытия/закрытия клапанов на сброс
if ((oldPperemen) > (SinIn + AveragePressure * limit))
{
    // Клапана на сброс давления открываются
      ValveStateMinus = 1;
}
```

```
else
{
    // Клапана на сброс давления закрываются
     ValveStateMinus = 0;
}
```

Далее рассчитываются значения времени, на которое открываются клапана и задержка между их открытием, согласно формуле (1) и (2) соответственно .

$$A = \left| \frac{\frac{dU_0}{dt}}{\frac{dU_0}{dt}_{max}} \right|, \quad (1)$$

где A – время, на которое открываются клапана;

$\frac{dU_0}{dt}$ – разница между текущим и предыдущим значением входной синусоиды;

$\frac{dU_0}{dt}_{max}$ – максимальная разница между текущим и предыдущим значением входной синусоиды.

$$t_{зад} = \left(1 - \sqrt{A}\right) * T_{on}, \quad (2)$$

где $t_{зад}$ – задержка между открытием клапанов;

T_{on} – интервал времени, задаваемый с компьютера.

```
// время на которое открываются клапана на подачу давления
OpenTimePlus = ValveStatePlus * abs((SinIn - oldSinIn) / (MaxSinIn));
// задержка между открытием клапанов
DelayTimePlus = (1 - sqrt(OpenTimePlus)) * deltaTime;
// время на которое открываются клапана на сброс давления
OpenTimeMinus = ValveStateMinus * abs((SinIn - oldSinIn) / (MaxSinIn));
```

```
// задержка между открытием клапанов
DelayTimeMinus = (1 - sqrt(OpenTimeMinus)) * deltaTime;
```

Далее рассчитывается погрешность формирования давления и отправляется на персональный компьютер. Также происходит отправка значений сформированного и образцового давления. Отправка происходит по 2 значения в виде: номер команды и значение давления, номер команды и погрешность. Это необходимо для того, чтобы программа высокого уровня смогла определить, какие данные приходят – давление или погрешность.

```
// Расчет значения погрешности
Delta = (Pperemen - SinTest) / Um / AveragePressure * 100;
// Задержка. Нужна для корректной работы последовательного порта
delay(200);
// Номер команды для отправки данных
comNum = 1;
fbSinTest.fSinTest = SinTest;
// Массив байтов для отправки значений
byte Sinvalues[5];
// Нулевой элемент массива это номер команды
Sinvalues[0] = (byte)comNum;
for(int g = 0; g<4; g++)
{
    Sinvalues[g+1] = fbSinTest.bSinTest[g];
}
// отправляем данные в последовательный порт
Serial.write(values, 5);
delay(200);
// номер команды для отправки данных
comNum = 2;
```

```
fbvalue.fvalue = Pperemen;
// Массив для отправки данных
byte Pressvalues[5];
Pressvalues[0] = (byte)comNum;
for(int g = 0; g<4; g++)
{
    Pressvalues[g+1] = fbvalue.bvalue [g];
}
// отправляем данные в последовательный порт
Serial.write(values, 5);
delay(200);
// номер команды для отправки данных
comNum = 3;
fbPress.fPress = Delta;
// Массив для отправки данных
byte Deltavalues[5];
Deltavalues[0] = (byte)comNum;
for(int g = 0; g<4; g++)
{
    Deltavalues[g+1] = fbPress.bPress[g];
}
// отправляем данные в последовательный порт
Serial.write(values, 5);
delay(200);
```

3.2.2 Функция для приема данных

В данной функции происходит считывание данных из последовательного порта и преобразование их из последовательности байтов в числа с плавающей запятой. Алгоритм функции представлен на рисунке 3.15.

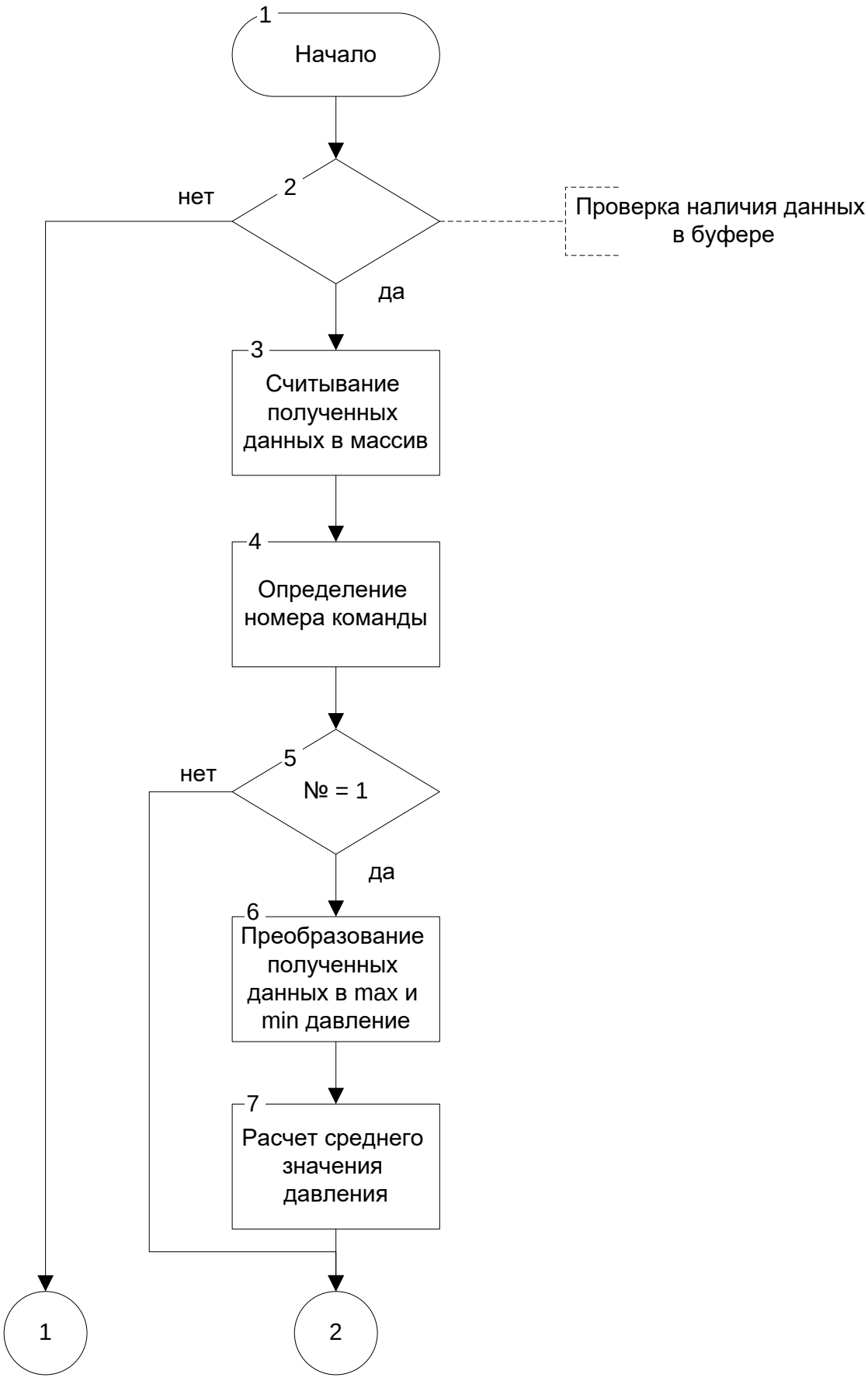


Рисунок 3.15 – Алгоритм функции для приема данных

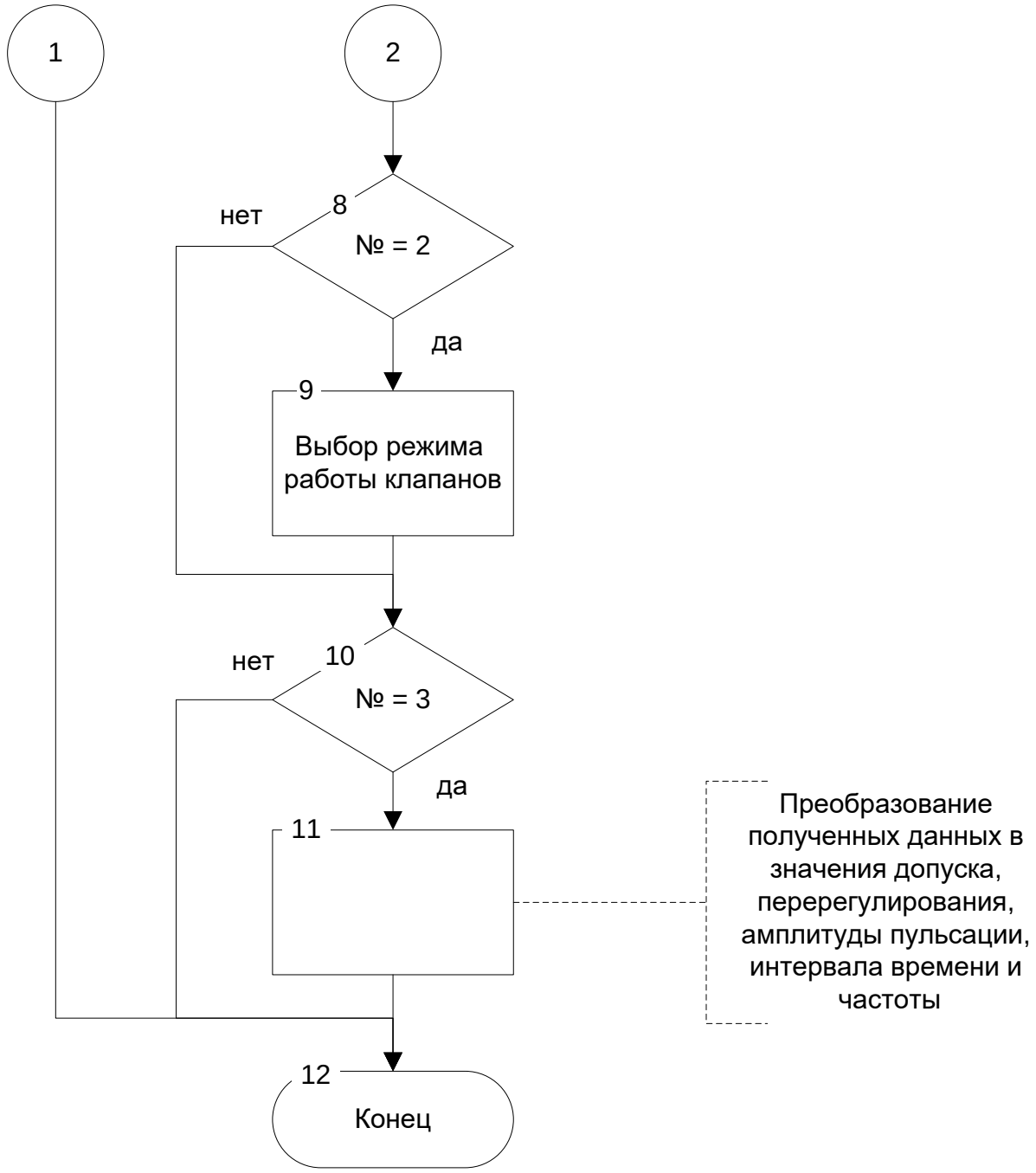


Рисунок 3.15 – Окончание алгоритма функции для приема данных

Первым делом происходит проверка наличия в буфере данных (буфер – промежуточное хранилище данных).

```
// Если в буфере есть данные
if (Serial.available() > 0)
```

Если данные присутствуют, то далее идет считывание их в массив значений.

```
// Стартовый индекс
int serialIndex = 0;
// Пока во входном буфере есть данные
while (Serial.available() > 0)
{
    // считывание текущего байта и запись его в массив
      serialValues[serialIndex] = Serial.read();
    // Увеличение индекса на 1
      serialIndex++;
}
```

Отправка и прием данных осуществляется с помощью протокола обмена разработанного в работе [12].

Протокол обмена представляет собой установленный набор соглашений, посредством которых два компьютера или устройства связи проверяют формат и содержание сообщений, которыми обмениваются [13].

Для приема и передачи данных был разработан следующий формат кадра запроса/ответа: номер команды, поле данных.

Общее число используемых команд будет 4. Одна на отправку данных и три на прием данных.

Первая команда содержит в себе номер команды, максимальное и минимальное значения давления.

```
comNum = (int)(serialValues[0]); // Считываем номер команды
if (comNum == 1) // Если номер команды равен 1
{
```

```
        // Массив необходимый для преобразования байтов в значение с плавающей запятой
        unsigned char MaxP[4];
        MaxP[0] = serialValues[4];
        MaxP[1] = serialValues[5];
        MaxP[2] = serialValues[6];
        MaxP[3] = serialValues[7];
       // Преобразование массива байтов в значение максимального давления
        MaxPressure = (*((float*)(MaxP)));
       // Массив необходимый для преобразования байтов в значение с плавающей запятой
        unsigned char MinP[4];
        MinP[0] = serialValues[8];
        MinP[1] = serialValues[9];
        MinP[2] = serialValues[10];
        MinP[3] = serialValues[11];
       // Преобразование массива байтов в значение минимального давления
       MinPressure = (*((float*)(MinP)));
       // Расчет среднего значения давления
       AveragePressure = (MaxPressure - MinPressure)/2;
    }
```

Вторая команда содержит в себе номер команды и режим управления клапанами.

```
    // Если номер команды равен 2
    if (comNum == 2)
            {
                // Определяем режим работы клапанов
                AutoManMode = (int)(serialValues[1]);
                // Если равно 1 то автоматический режим
```

```
            // Если равно 2 то ручной режим
            // определяем какие клапана будем использовать
            Valve = (int)(serialValues[2]);
            // Если 1 то клапана на подачу
            // Если 2 то клапана на сброс
            // определяем состояние клапанов
            ValveState = (int)(serialValues[3]);
            // Если 1 то клапана открыть
            // Если 2 то клапана закрыть
        }
```

Регулирование клапанов происходит посредством ШИМ.

Код для автоматического/ручного режима управления клапанами выглядит следующим образом:

```
if (AutoManMode == 1) // 1 - автоматический режим
            {
             if (ValveStatePlus == 1)
             {
              analogWrite(InValve1, OpenTimePlus * 255);
              delay(DelayTimePlus); // время задержки между открытием клапанов
              analogWrite(InValve2, OpenTimePlus * 255);
              analogWrite(InValve3, OpenTimePlus * 255);
              delay(DelayTimePlus); // время задержки между открытием клапанов
              analogWrite(InValve4, OpenTimePlus * 255);
             }
             if (ValveStatePlus == 0)
             {
              analogWrite(InValve1, 0);
              analogWrite(InValve2, 0);
              analogWrite(InValve3, 0);
```

```
  analogWrite(InValve4, 0);
 }
 if (ValveStateMinus == 1)
 {
  analogWrite(OutValve1, OpenTimeMinus * 255);
  delay(DelayTimeMinus); // время задержки между открытием клапанов
  analogWrite(OutValve2, OpenTimeMinus * 255);
  analogWrite(OutValve3, OpenTimeMinus * 255);
  delay(DelayTimeMinus); // время задержки между открытием клапанов
  analogWrite(OutValve4, OpenTimeMinus * 255);
 }
 if (ValveStateMinus == 0)
 {
  analogWrite(OutValve1, 0);
  analogWrite(OutValve2, 0);
  analogWrite(OutValve3, 0);
  analogWrite(OutValve4, 0);
 }
}
if (AutoManMode == 2) // 2 - ручной режим
{
 if (Valve == 1) // Если 1 то на подачу
 {
    if (ValveState == 1) // Если 1 то открывать клапана
    {
      analogWrite(InValve1, 255);
      analogWrite(InValve2, 255);
      delay(2);
      analogWrite(InValve3, 255);
      analogWrite(InValve4, 255);
    }
```

```
    if (ValveState == 2) // Если 2 то закрывать клапана
    {
      analogWrite(InValve1, 0);
      analogWrite(InValve2, 0);
      analogWrite(InValve3, 0);
      analogWrite(InValve4, 0);
    }
  }
  if (Valve == 2) // Если 2 то на сброс
  {
    if (ValveState == 1) // Если 1 то открывать клапана
    {
      analogWrite(OutValve1, 255);
      analogWrite(OutValve2, 255);
      delay(2);
      analogWrite(OutValve3, 255);
      analogWrite(OutValve4, 255);
    }
    if (ValveState == 2) // Если 2 то закрывать клапана
    {
      analogWrite(OutValve1, 0);
      analogWrite(OutValve2, 0);
      analogWrite(OutValve3, 0);
      analogWrite(OutValve4, 0);
    }
  }
}
```

Третья команда содержит в себе номер команды и переменные, такие как частота дискретизации, интервал времени, амплитуда пульсации, допуск, сдвиг, величина перерегулирования.

```
if (comNum == 3) // если номер команды равен 3
{
    // Массив необходимый для преобразования байтов в значение с плавающей запятой
     unsigned char sdvig[4];
     sdvig[0] = serialValues[4];
     sdvig[1] = serialValues[5];
     sdvig[2] = serialValues[6];
     sdvig[3] = serialValues[7];
    // Преобразование массива байтов в значение сдвига
     Shift = (*((float*)(sdvig)));
    // Массив необходимый для преобразования байтов в значение с плавающей запятой
     unsigned char lim[4];
     lim[0] = serialValues[8];
     lim[1] = serialValues[9];
     lim[2] = serialValues[10];
     lim[3] = serialValues[11];
    // Преобразование массива байтов в значение предела
     limit = (*((float*)(lim)));
    // Массив необходимый для преобразования байтов в значение с плавающей запятой
     unsigned char valVp[4];
     valVp[0] = serialValues[12];
     valVp[1] = serialValues[13];
     valVp[2] = serialValues[14];
     valVp[3] = serialValues[15];
    // Преобразование массива байтов в величину перерегулирования
     Vp = (*((float*)(valVp)));
```

```
        // Массив необходимый для преобразования байтов в значение с плавающей запятой
        unsigned char valUm[4];
        valUm[0] = serialValues[16];
        valUm[1] = serialValues[17];
        valUm[2] = serialValues[18];
        valUm[3] = serialValues[19];
       // Преобразование массива байтов в амплитуду пульсации
        Um = (*((float*)(valUm)));
       // Массив необходимый для преобразования байтов в значение с плавающей запятой
        unsigned char interval[4];
        interval[0] = serialValues[20];
        interval[1] = serialValues[21];
        interval[2] = serialValues[22];
        interval[3] = serialValues[23];
       // Преобразование массива байтов в интервал времени
        deltaTime = (*((float*)(interval)));
        // Массив необходимый для преобразования байтов в значение с плавающей запятой
        unsigned char Freq[4];
        Freq[0] = serialValues[24];
        Freq[1] = serialValues[25];
        Freq[2] = serialValues[26];
        Freq[3] = serialValues[27];
       // Преобразование массива байтов в частоту дискретизации
        Frequence = (*((float*)(Freq)));
    }
```

Четвертая команда представляет собой отправку значений сформированного и образцового давления, погрешности формирования давления, которая была представлена в пункте 3.2.1.

Вывод по главе:

1. В данной главе был разработан алгоритм программного обеспечения в виде блок-схемы, который состоит из двух основных функций – функция считывания давления и расчета погрешности, и функция для приема данных.
2. Разработан код программного обеспечения на языке C++.
3. Далее необходимо провести испытание с применением разработанной программы.

4 ПРОВЕДЕНИЕ ИСПЫТАНИЙ С ПРИМЕНЕНИЕМ ПРОГРАММЫ ВЕРХНЕГО УРОВНЯ

В данной главе необходимо описать проведенные испытания с применением разработанной программы. Также устранить недостатки и ошибки, найденные в процессе её тестирования.

Для отладки, проверки работоспособности программы и для моделирования работы блока клапанов и рабочего объема с датчиками, была сделана плата-имитатор, основанная на диодах и транзисторах (рисунок 4.1).

Использовать данную плату намного удобнее, чем реальный стенд, т.к. она более компактная и мобильная и работать с ней намного проще.

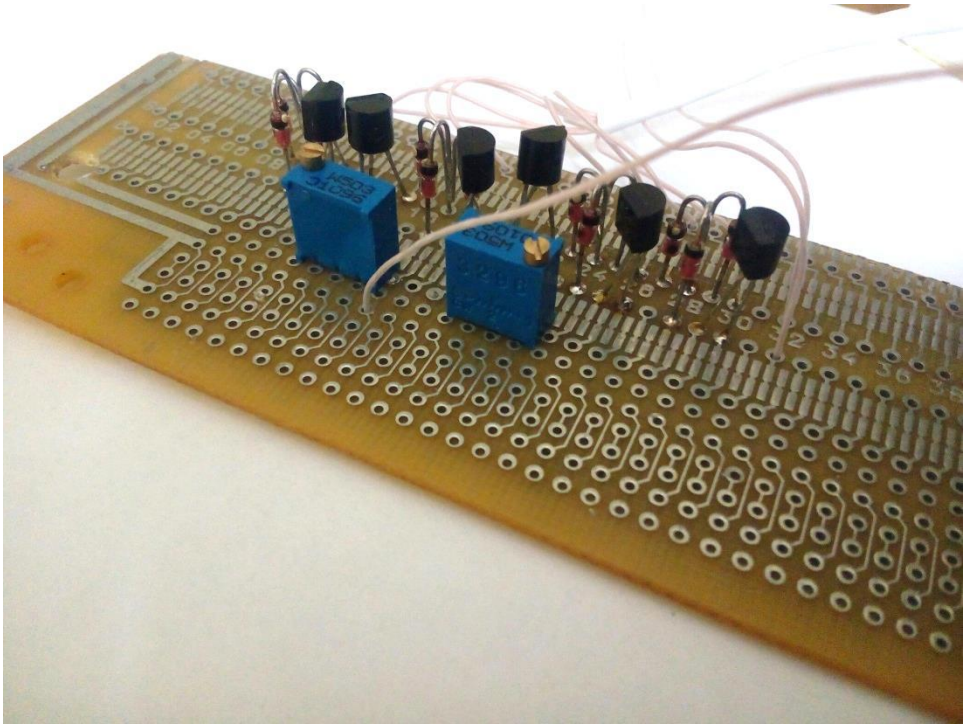


Рисунок 4.1 – Плата для моделирования работы блока клапанов

Плата состоит из 8 диодов, которые имитируют работу клапанов. 2 транзистора имитируют подачу давления в рабочий объем. Каскад из 4

транзисторов имитирует сброс давления. Далее идут два переменных резистора, которые имитируют регулировочные вентили, отвечающие за расход воздуха.

Далее перейдем непосредственно к тестированию программного обеспечения.

Для проведения экспериментов необходимо использовать программное обеспечение высокого уровня, разработанное в работе [12].

После запуска программы открывается окно с выбором устройства для работы (рисунок 4.2).

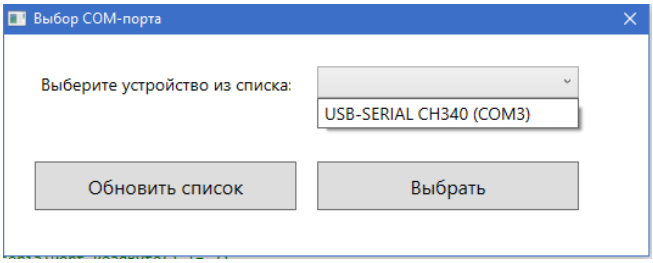


Рисунок 4.2 – Выбор устройства

Выбрав нужное устройство, открывается основное окно программы (рисунок 4.3).

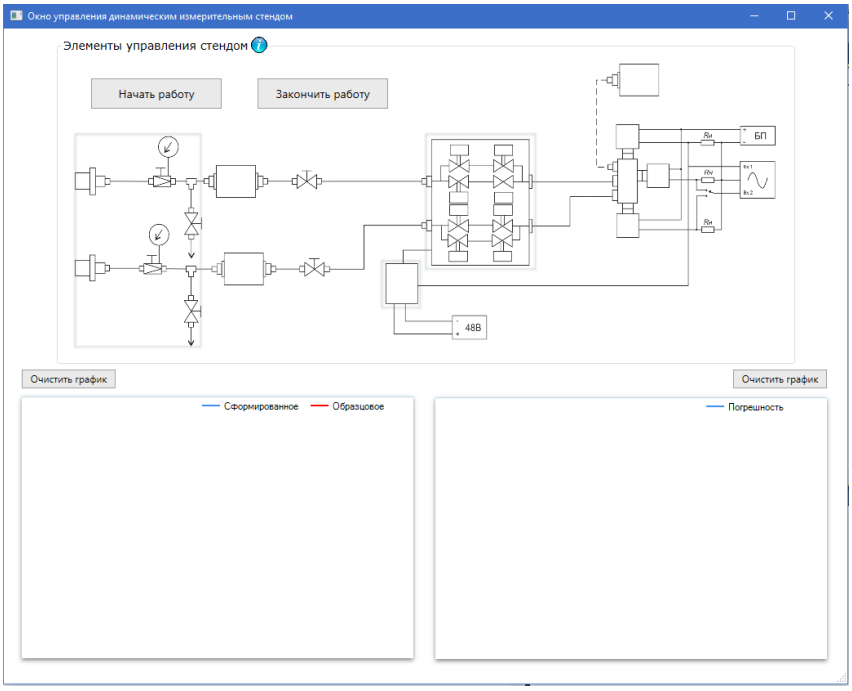


Рисунок 4.3 – Основное окно программы

Для того чтобы начать работу с программой необходимо нажать кнопку «Начать работу», программа начинает принимать данные: сформированное давление, образцовое давление и погрешность формирования давления.

Открывая контекстное меню необходимо задать максимальное и минимальное давление (рисунок 4.4).

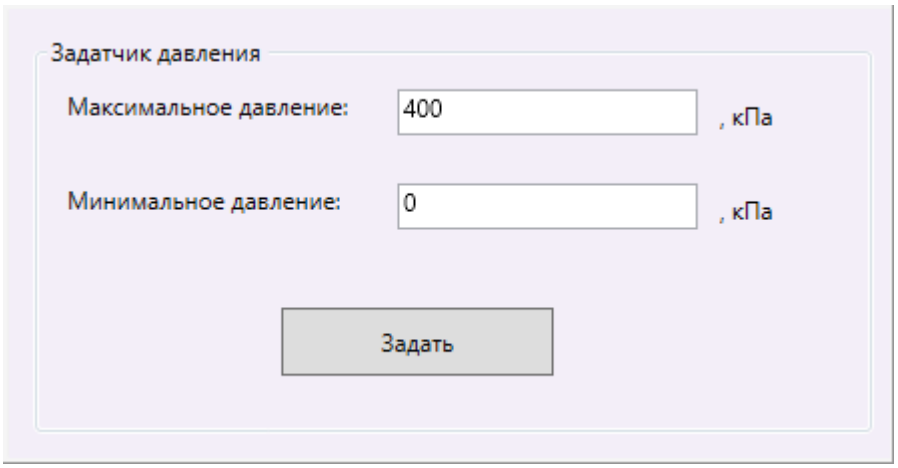


Рисунок 4.4 – Задатчик давления

Открывая следующее контекстное меню, необходимо задать значения для формирования опорного сигнала, такие как амплитуда пульсации и частота синусоиды (рисунок 4.5).

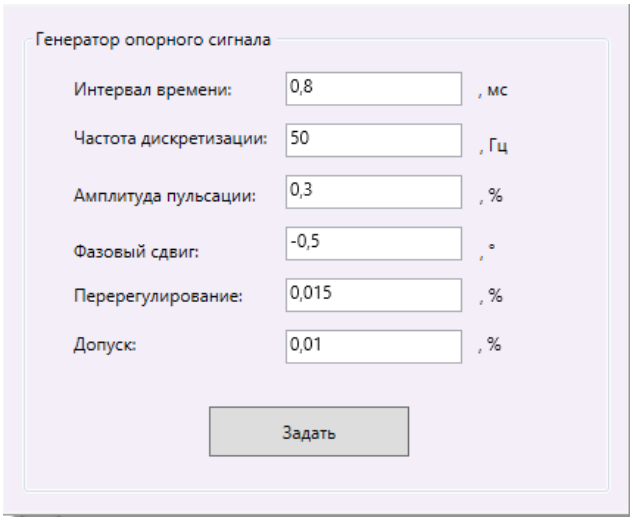


Рисунок 4.5 – Переменные опорного сигнала

Задав все необходимые данные, получаем следующие графики (рисунок 4.6).

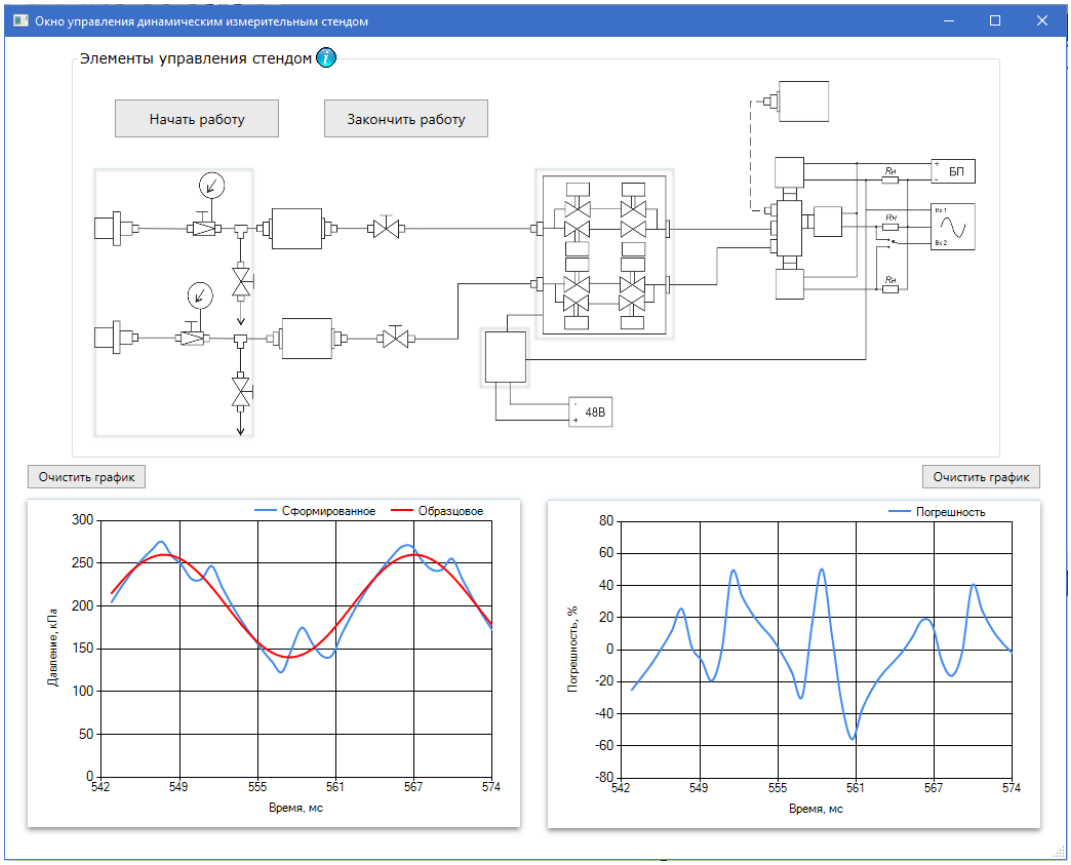


Рисунок 4.6 – Графики давления и погрешности

Проанализировав графики, можем сказать, что разработанная программа достаточно корректно осуществляет управление клапанами и формирует давление близкое к опорному сигналу.

Для эксперимента изменим некоторые параметры, такие как максимальное и минимальное давление и частота опорного сигнала (рисунок 4.7).

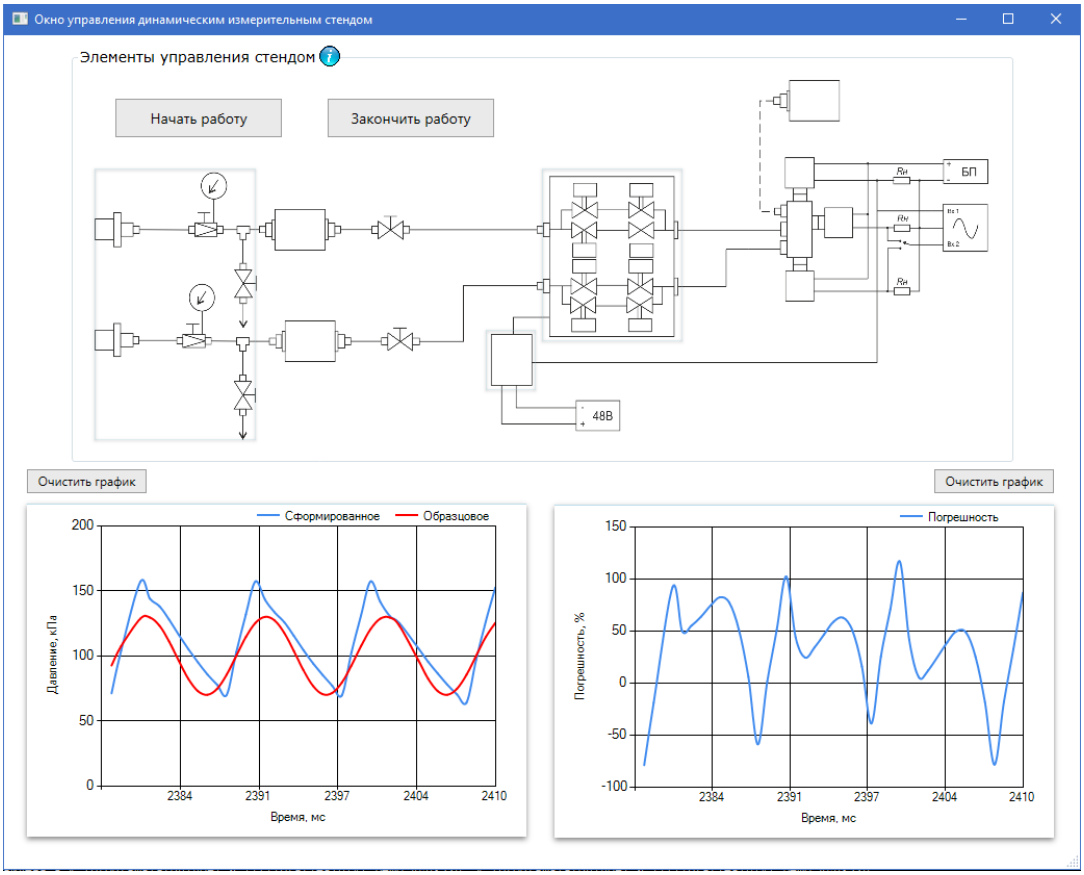


Рисунок 4.7 – Изменение параметров

Изменив параметры, видим, что программа работает не совсем идеально, но полученная погрешность допускается в работе с используемой платой-имитатором.

Далее проведем испытание ручного режима управления клапанами, который включается в следующем контекстном меню (рисунок 4.8).

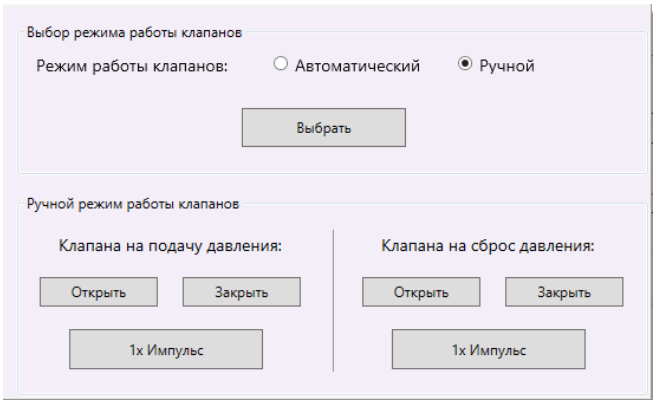


Рисунок 4.8 – Ручной режим работы клапанов

В ручном режиме можно полностью открыть клапана на подачу (рисунок 4.9), либо полностью открыть клапана на сброс (рисунок 4.10), тем самым получая не синусоидальное давление, а постоянное.

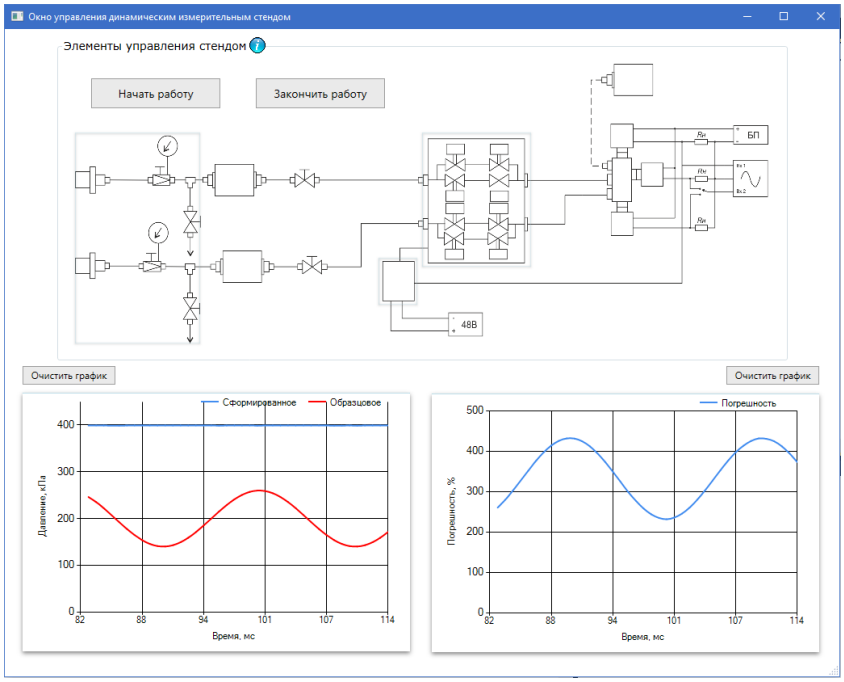


Рисунок 4.9 – Полностью открытые клапана на подачу

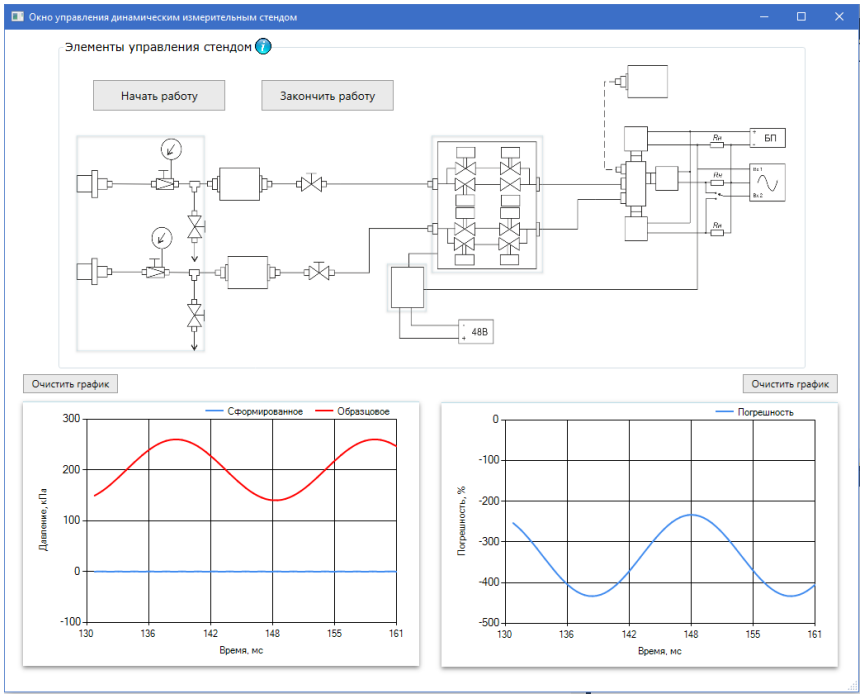


Рисунок 4.10 – Полностью открытые клапана на сброс

Также программа предусматривает возможность однократного открытия клапанов (рисунок 4.11).

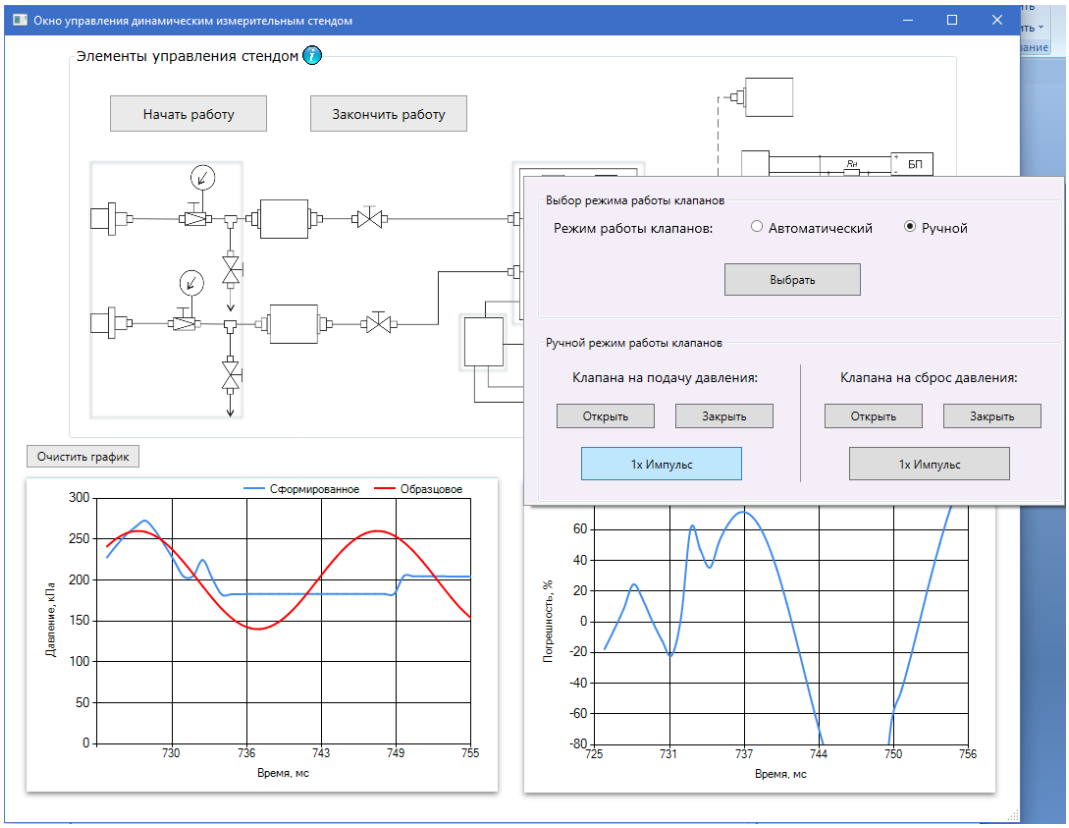


Рисунок 4.11 – Ручной режим управления клапанами

Одна из вариаций сформированного давления, полученная многократным нажатием кнопки «1х Импульс» для клапанов на подачу воздуха изображена на рисунке 4.12.

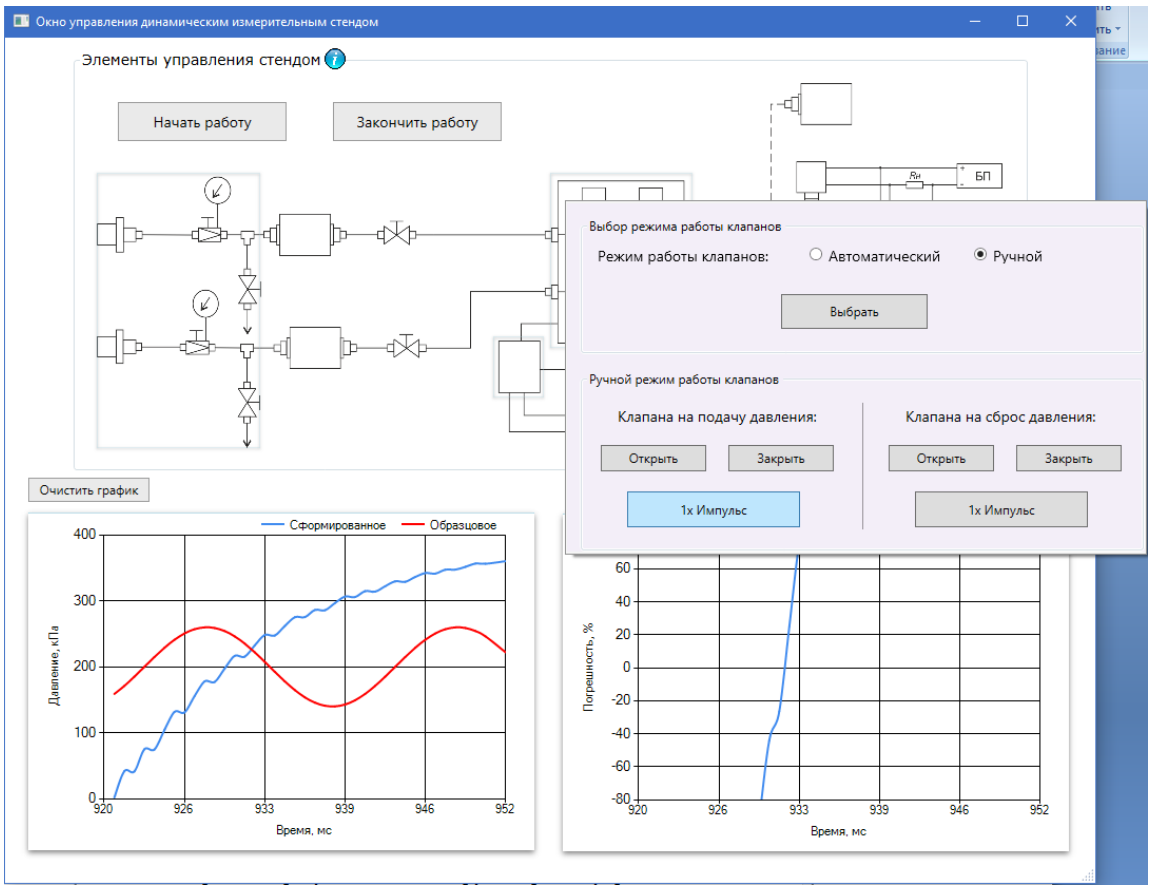


Рисунок 4.12 – Вариация открытия клапанов

Таким образом, из рисунков видим, что ручной режим управления клапанами позволяет получать сформированное давление произвольной формы.

Выводы по главе:

Проведя эксперименты с разработанными программными обеспечениями можно сказать, что программа работает корректно. Все функции работают исправно. Ошибок во время эксперимента не было замечено.

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе было разработано программное обеспечение нижнего уровня для динамического измерительного стенда.

Для достижения данной цели был проведен аналитический обзор существующих архитектур микроконтроллера и для поставленной задачи был выбран подходящий микроконтроллер с AVR архитектурой на базе отладочной платы Arduino mega 2560.

Следующим этапом работы было описание принципа работы динамического измерительного стенда. Была представлена структурная схема стенда и описана работа его отдельных элементов.

Далее были разработан алгоритм программного обеспечения, который состоит из двух основных функций – это функция для приема данных и функция считывания давления и расчета погрешности. Также был написан код программы.

Заключительной частью работы стало проведение испытаний с применением разработанной программы. Необходимо было отладить и проверить работоспособность программного обеспечения, используя плату имитирующую работу динамического измерительного стенда.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Хартов, В.Я. Микроконтроллеры AVR. Практикум для начинающих / В.Я. Хартов. – М.: МГТУ им. Н.Э. Баумана, 2007. – 240 с.
2. Васильев, А.Е. Микроконтроллеры. Разработка встраиваемых приложений / А.Е. Васильев. – СПб.: БХВ-Петербург, 2008. – 304 с.
3. Голубцов, М.С. Микроконтроллеры AVR: от простого к сложному / М.С. Голубцов. – М.: СОЛОН-Пресс, 2003. – 288 с.
4. ARM архитектура. − http://dic.academic.ru/dic.nsf/ruwiki/1264844
5. AVR учебный курс. − http://easyelectronics.ru/vvodnaya-ili-chto-takoe-mikrokontroller.html
6. AVR против STM. − http://www.avrki.ru/articles/content/avr_stm/
7. Структурная организация AVR. − http://masters.donntu.org/2005/fvti/kalinichenko/library/9.htm
8. Сравнение микроконтроллеров. − https://habrahabr.ru/post/120611/
9. Белов, А.В. Самоучитель разработчика устройств на микроконтроллерах avr / А.В. Белов. – СПб.: Наука и техника, 2008. – 544 с.
10. Грановский, В.А. Динамические измерения: Основы метрологического обеспечения. / В.А. Грановский. – Л.: Энергоатомиздат, 1984. – 224 с.
11. Методические указания к выполнению лабораторных работ по курсу «Мехатронные и роботизированные системы» / сост. И.С. Никитин. – Челябинск: Изд. ЮУрГУ, 2017. – 105 с.
12. Трусов, В.А. Разработка алгоритмов и системы команд для динамического измерительного стенда: пояснительная записка ВКР / В.А. Трусов. – Челябинск, 2017. – 64 с.
13. Popovic, M. Communication Protocol Engineering. / M. Popovic. – Boca Raton.: CRC Press, 2006. – 500 p.

ПРИЛОЖЕНИЕ А

Листинг программы

```
#include <EEPROM.h> // подключаем библиотеку

#define OutValve4 2
#define OutValve3 3
#define OutValve2 4
#define OutValve1 5
#define InValve4 6
#define InValve3 7
#define InValve2 8
#define InValve1 9
#define pin  A1
extern uint8_t SmallFont[];
unsigned char serialValues[40];

int AutoManMode = 1; // режим работы клапанов
int Valve = 0; // клапана на подачу или сброс
int ValveState = 0; // состояние клапанов (открыты/закрыты)

float AveragePressure = 200; // Среднее значение давления
float SinIn = 0; // Значение входной синусоиды
float oldSinIn = 0;
float oldoldSinIn = 0;
float SinTest = 0; // Значение образцевой синусоиды
float oldSinTest = 0;
float deltaTime = 0.8; // Интервал времени
float MaxPressure = 400; // Максимальное давление
float MinPressure = 0; // Минимальное давление
float Frequence = 50; // Частота дискретизации
float Shift = -0.5; // Сдвиг
float Um = 0.3; // Амплитуда пульсации
int i = 0;
float Delta=0; // Погрешность
float Pperemen = 0; // Сформированное давление
float oldPperemen = 0;
float limit = 0.01; // Допуск
```

```
float Uv = 2.5; // Расход воздуха
float Vp = 0.015; // Величина перерегулирования
int StartSend = 0;

float MaxSinIn = 0;
float ValveStatePlus = 0; // Состояние клапана на подачу (открыт/закрыт)
float OpenTimePlus = 0; // Время на которое открывается клапан
float DelayTimePlus = 0; // Задержка между открытием клапанов
float DelayTimeMinus =0; // Задержка между открытием клапанов
float ValveStateMinus = 0; // Состояние клапана на сброс (открыт/закрыт)
float OpenTimeMinus = 0; // Время на которое открывается клапан
float comNumber = 0; // Номер команды для передачи
int comNum = 0; // Номер команды для приема

float timeArray = 0;
float oldtimeArray = 0;

// Массив для отправки значений сформированного давления
union {float fvalue; byte bvalue[4];} fbvalue;
// Массив для отправки значений погрешности
union {float fpress; byte bpress[4];} fbpress;
// Массив для отправки значений образцового давления
union {float fSinTest; byte bSinTest[4];} fbSinTest;

long valve1 = 0;
long valve2 = 0;
long valve3 = 0;
long valve4 = 0;

/*************************************************/
void WriteEepromData()
{
   for (int addr = 0; addr < 1024; )
   {
     EEPROM.write(addr, MaxPressure);
    addr += sizeof(float);
    EEPROM.write(addr, MinPressure);
    addr += sizeof(float);
    EEPROM.write(addr, AveragePressure);
    addr += sizeof(float);
```

```
        EEPROM.write(addr, AutoManMode);
        addr += sizeof(int);
        EEPROM.write(addr, Valve);
        addr += sizeof(int);
        EEPROM.write(addr, ValveState);
        addr += sizeof(int);
        EEPROM.write(addr, Shift);
        addr += sizeof(float);
        EEPROM.write(addr, limit);
        addr += sizeof(float);
        EEPROM.write(addr, Vp);
        addr += sizeof(float);
        EEPROM.write(addr, Um);
        addr += sizeof(float);
        EEPROM.write(addr, deltaTime);
        addr += sizeof(float);
        EEPROM.write(addr, Frequence);
        addr += sizeof(float);
        EEPROM.write(addr, StartSend);
         }
    }
    /************************************************/
    void calcPressAndDelta() // расчет значений давления и погрешности
    {
            //if (StartSend == 9)
            //{
            oldtimeArray = timeArray;
            timeArray = (i - 2)*deltaTime;

            oldoldSinIn = oldSinIn;
            oldSinIn = SinIn;
            SinIn = AveragePressure + Um * AveragePressure * sin((2*PI*Frequence/1000) * timeArray);

            oldSinTest = SinTest;
            SinTest = AveragePressure + Um * AveragePressure * sin((2*PI*Frequence/1000)*timeArray -
Shift/6.28);

            //Pperemen = SinIn;
            //ValveStatePlus = 0;
            //OpenTimePlus = 0;
```

```
//ValveStateMinus = 0;
//OpenTimeMinus = 0;

oldPperemen = Pperemen;
Pperemen = (analogRead(pin)) / 2.5575;

//Serial.println("Pressure" + (String)Pperemen);
//Serial.println("SinIn" + (String)SinIn);
// Условие проверки для формирования массива значений 1го столбца ШИМ(+)
if ((oldPperemen) < (SinIn - AveragePressure * limit))
{
   ValveStatePlus = 1;
}
else
{
   ValveStatePlus = 0;
}

if ((oldSinIn - oldoldSinIn) < (SinIn - oldSinIn))
{
  MaxSinIn = SinIn - oldSinIn;
}
 OpenTimePlus = ValveStatePlus * abs((SinIn - oldSinIn) / (MaxSinIn)); // время на которое открывается клапан
DelayTimePlus = (1 - sqrt(OpenTimePlus)) * deltaTime; // задержка между открытием клапанов
  // Условие проверки для формирования массива значений 1го столбца ШИМ(-)
if ((oldPperemen) > (SinIn + AveragePressure * limit))
{
   ValveStateMinus = 1;
}
else
{
   ValveStateMinus = 0;
}

OpenTimeMinus = ValveStateMinus * abs((SinIn - oldSinIn) / (MaxSinIn)); // время на которое открывается клапан
DelayTimeMinus = (1 - sqrt(OpenTimeMinus)) * deltaTime; // задержка между открытием клапанов
```

```
                    //Serial.println("OpenTimePlus_" + String(OpenTimePlus));
                     // Создание массива значений сформированного давления
                     //Pperemen = oldPperemen + (MaxPressure - oldPperemen) * ValveStatePlus * Uv * (timeArray -
oldtimeArray - DelayTimePlus) / 1000 / Vp +
                     //                          (MinPressure - oldPperemen) * ValveStateMinus * Uv * (timeArray -
oldtimeArray - DelayTimeMinus) / 1000 / Vp;

                    if (DelayTimeMinus < 0)
                    {
                     DelayTimeMinus = 0;
                    }
                     if (DelayTimePlus < 0)
                    {
                     DelayTimePlus = 0;
                    }

                    //Serial.println("Pperemen_" + String(Pperemen));

                    // Создание массива значений погрешности формирования
                    //Delta[k - 1] = (Pperemen[k - 1] - SinTest[k - 1]) / Um / AveragePressure * 100;
                    Delta = (((Pperemen - SinTest) / Um) / AveragePressure) * 100;

                    //Serial.println("SinTest_" + String(SinTest));

                    if (AutoManMode == 1) // 1 - автоматический режим
                    {
                     if (ValveStatePlus == 1)
                     {
                       //analogWrite(InValve1, OpenTimePlus * 255);
                       //analogWrite(InValve2, OpenTimePlus * 255);
                       //analogWrite(InValve3, OpenTimePlus * 255);
                       //analogWrite(InValve4, OpenTimePlus * 255);

                       digitalWrite(InValve1, HIGH); // открываем 1ый клапан
                       delayMicroseconds(DelayTimePlus * 1000); // время задержки между открытием клапанов
                       digitalWrite(InValve2, HIGH); // открываем 2ой клапан
                       delayMicroseconds((1 - DelayTimePlus) * 1000); // время на которое клапан открывается
                       digitalWrite(InValve1, LOW); // закрываем 1ый клапан
                       delayMicroseconds(DelayTimePlus * 1000); // время через которое закроется 2ой клапан
                       digitalWrite(InValve2, LOW); // закрываем клапан
```

```
  delay(1);

  digitalWrite(InValve3, HIGH); // открываем 3ий клапан
  delayMicroseconds(DelayTimePlus * 1000); // время задержки между открытием клапанов
  digitalWrite(InValve4, HIGH); // открываем 4ый клапан
  delayMicroseconds((1 - DelayTimePlus) * 1000); // время на которое клапан открывается
  digitalWrite(InValve3, LOW); // закрываем клапан
  delayMicroseconds(DelayTimePlus * 1000); // время на которое открывается клапан
  digitalWrite(InValve4, LOW); // закрываем клапан

  //Serial.println("Delay_" + String(DelayTimePlus * 1000));
 }
 if (ValveStatePlus == 0)
 {
  /*analogWrite(InValve1, 0);
  analogWrite(InValve2, 0);
  analogWrite(InValve3, 0);
  analogWrite(InValve4, 0);*/
  digitalWrite(InValve1, LOW); // закрываем клапан
  digitalWrite(InValve2, LOW); // закрываем клапан
  digitalWrite(InValve3, LOW); // закрываем клапан
  digitalWrite(InValve4, LOW); // закрываем клапан
 }
 if (ValveStateMinus == 1)
 {
 digitalWrite(OutValve1, HIGH); // открываем 1ый клапан
  delayMicroseconds(DelayTimeMinus * 1000); // время задержки между открытием клапанов
  digitalWrite(OutValve2, HIGH); // открываем 2ой клапан
  delayMicroseconds((1 - DelayTimeMinus) * 1000); // время на которое клапан открывается
  digitalWrite(OutValve1, LOW); // закрываем 1ый клапан
  delayMicroseconds(DelayTimeMinus * 1000); // время через которое закроется 2ой клапан
  digitalWrite(OutValve2, LOW); // закрываем клапан

  delay(1);

  digitalWrite(OutValve3, HIGH); // открываем 3ий клапан
  delayMicroseconds(DelayTimeMinus * 1000); // время задержки между открытием клапанов
  digitalWrite(OutValve4, HIGH); // открываем 4ый клапан
  delayMicroseconds((1 - DelayTimeMinus) * 1000); // время на которое клапан открывается
```

```
  digitalWrite(OutValve3, LOW); // закрываем клапан
  delayMicroseconds(DelayTimeMinus * 1000); // время на которое открывается клапан
  digitalWrite(OutValve4, LOW); // закрываем клапан
 }
 if (ValveStateMinus == 0)
 {
  /*analogWrite(OutValve1, 0);
  analogWrite(OutValve2, 0);
  analogWrite(OutValve3, 0);
  analogWrite(OutValve4, 0);*/
  digitalWrite(OutValve1, LOW); // закрываем клапан
  digitalWrite(OutValve2, LOW); // закрываем клапан
  digitalWrite(OutValve3, LOW); // закрываем клапан
  digitalWrite(OutValve4, LOW); // закрываем клапан
  }
}
if (AutoManMode == 2) // 2 - ручной режим
{
 if (Valve == 1) // Если 1 то на подачу
 {
    if (ValveState == 1) // Если 1 то открывать клапана
    {
      digitalWrite(InValve1, HIGH);
      digitalWrite(InValve2, HIGH);
      delay(4);
      digitalWrite(InValve3, HIGH);
      digitalWrite(InValve4, HIGH);
    }
    if (ValveState == 2) // Если 2 то закрывать клапана
    {
      digitalWrite(InValve1, LOW);
      digitalWrite(InValve2, LOW);
      digitalWrite(InValve3, LOW);
      digitalWrite(InValve4, LOW);
    }
 }
 if (Valve == 2) // Если 2 то на сброс
 {
    if (ValveState == 1) // Если 1 то открывать клапана
    {
```

```
      digitalWrite(OutValve1, HIGH);
      digitalWrite(OutValve2, HIGH);
      delay(4);
      digitalWrite(OutValve3, HIGH);
      digitalWrite(OutValve4, HIGH);
    }
    if (ValveState == 2) // Если 2 то закрывать клапана
    {
      digitalWrite(OutValve1, LOW);
      digitalWrite(OutValve2, LOW);
      digitalWrite(OutValve3, LOW);
      digitalWrite(OutValve4, LOW);
    }
  }
}
comNum = 3; // номер команды для отправки данных
fbSinTest.fSinTest = SinIn;

fbvalue.fvalue = Pperemen;
fbpress.fpress = Delta;
byte values[16];
values[0] = 7;
values[1] = (byte)comNum;
for(int g = 0; g<4; g++)
{
  values[g+2] = fbSinTest.bSinTest[g];
}

comNum = 1; // номер команды для отправки данных
values[6] = (byte)comNum;
for(int g = 0; g<4; g++)
{
  values[g+7] = fbvalue.bvalue[g];
}
comNum = 2; // номер команды для отправки данных
values[11] = (byte)comNum;
for(int g = 0; g<4; g++)
{
  values[g+12] = fbpress.bpress[g];
}
```

```
            Serial.write(values, 16); // отправляем данные в COM порт
            i++;
}
/************************************************/
void recieveData() // Функция для получения данных
      {
        if (Serial.available() > 0) // Если в буфере есть данные
        {
          int serialIndex = 0; // Стартовый индекс
          while (Serial.available() > 0) // Пока во входном буфере есть данные
          {
           //mdt += Serial.read() + "+";
           delay(5);
             serialValues[serialIndex] = Serial.read(); // считывание текущего байта и запись его в массив
             serialIndex++; // Увеличение индекса на 1
           }
           comNum = (int)(serialValues[0]); // Считываем номер команды
          if (comNum == 1) // Если номер команды равен 1
          {
           unsigned char MaxP[4];
           MaxP[0] = serialValues[2];
           MaxP[1] = serialValues[3];
           MaxP[2] = serialValues[4];
           MaxP[3] = serialValues[5];
           MaxPressure = (*((float*)(MaxP)));

           unsigned char MinP[4];
           MinP[0] = serialValues[6];
           MinP[1] = serialValues[7];
           MinP[2] = serialValues[8];
           MinP[3] = serialValues[9];
           MinPressure = (*((float*)(MinP)));

           AveragePressure = (MaxPressure + MinPressure)/2;
          }
          if (comNum == 2) // Если номер команды равен 2
          {
           AutoManMode = (int)(serialValues[1]); // Определяем режим работы клапанов
           // Если равно 1 то автоматический режим
           // Если равно 2 то ручной режим
```

```
 Valve = (int)(serialValues[2]); // определяем какие клапана будем использовать
  // Если 1 то клапана на подачу
  // Если 2 то клапана на сброс

  ValveState = (int)(serialValues[3]); // определяем состояние клапанов
  // Если 1 то клапана открыть
  // Если 2 то клапана закрыть
}
   if (comNum == 3) // если номер команды равен 3
{
  unsigned char sdvig[4];
  sdvig[0] = serialValues[4];
  sdvig[1] = serialValues[5];
  sdvig[2] = serialValues[6];
  sdvig[3] = serialValues[7];
  Shift = (*((float*)(sdvig)));

  unsigned char lim[4];
  lim[0] = serialValues[8];
  lim[1] = serialValues[9];
  lim[2] = serialValues[10];
  lim[3] = serialValues[11];
  limit = (*((float*)(lim)));

  unsigned char valVp[4];
  valVp[0] = serialValues[12];
  valVp[1] = serialValues[13];
  valVp[2] = serialValues[14];
  valVp[3] = serialValues[15];
  Vp = (*((float*)(valVp)));

  unsigned char valUm[4];
  valUm[0] = serialValues[16];
  valUm[1] = serialValues[17];
  valUm[2] = serialValues[18];
  valUm[3] = serialValues[19];
  Um = (*((float*)(valUm)));

  unsigned char interval[4];
  interval[0] = serialValues[20];
```

```
  interval[1] = serialValues[21];
  interval[2] = serialValues[22];
  interval[3] = serialValues[23];
  deltaTime = (*((float*)(interval)));

  unsigned char Freq[4];
  Freq[0] = serialValues[24];
  Freq[1] = serialValues[25];
  Freq[2] = serialValues[26];
  Freq[3] = serialValues[27];
  Frequence = (*((float*)(Freq)));
}
if (comNum == 4) // Если номер команды равен 4
{
   digitalWrite(InValve1, HIGH); // открываем 1ый клапан
    digitalWrite(InValve2, HIGH); // открываем 2ой клапан
    delay(1); // время на которое клапан открывается
    digitalWrite(InValve1, LOW); // закрываем клапан
    digitalWrite(InValve2, LOW); // закрываем клапан

    delay(1); // время задержки между открытием клапанов

    digitalWrite(InValve3, HIGH); // открываем 3ий клапан
    digitalWrite(InValve4, HIGH); // открываем 4ый клапан
    delay(1); // время на которое клапан открывается
    digitalWrite(InValve3, LOW); // закрываем клапан
    digitalWrite(InValve4, LOW); // закрываем клапан
}
if (comNum == 5) // Если номер команды равен 5
{
    digitalWrite(OutValve1, HIGH); // открываем 1ый клапан
    digitalWrite(OutValve2, HIGH); // открываем 2ой клапан
    delay(1); // время на которое клапан открывается
    digitalWrite(OutValve1, LOW); // закрываем клапан
    digitalWrite(OutValve2, LOW); // закрываем клапан

    delay(1); // время задержки между открытием клапанов

    digitalWrite(OutValve3, HIGH); // открываем 3ий клапан
    digitalWrite(OutValve4, HIGH); // открываем 4ый клапан
```

```
                delay(1); // время на которое клапан открывается
                digitalWrite(OutValve3, LOW); // закрываем клапан
                digitalWrite(OutValve4, LOW); // закрываем клапан
              }
              if (comNum == 8)
              {
                //StartSend = 8;
              }
              if (comNum == 9)
              {
               calcPressAndDelta();
              }
            }
        }

void setup()
{
   Serial.begin(57600);

  pinMode(OutValve1, OUTPUT);
  pinMode(OutValve2, OUTPUT);
  pinMode(OutValve3, OUTPUT);
  pinMode(OutValve4, OUTPUT);
  pinMode(InValve1, OUTPUT);
  pinMode(InValve2, OUTPUT);
  pinMode(InValve3, OUTPUT);
  pinMode(InValve4, OUTPUT);

  for (int addr = 0; addr < 1024; )
    {
        EEPROM.put(addr, MaxPressure);
     addr += sizeof(float);
     EEPROM.put(addr, MinPressure);
     addr += sizeof(float);
     EEPROM.put(addr, AveragePressure);
     addr += sizeof(float);
     EEPROM.put(addr, AutoManMode);
     addr += sizeof(int);
     EEPROM.put(addr, Valve);
     addr += sizeof(int);
```

```
      EEPROM.put(addr, ValveState);
      addr += sizeof(int);
      EEPROM.put(addr, Shift);
      addr += sizeof(float);
      EEPROM.put(addr, limit);
      addr += sizeof(float);
      EEPROM.put(addr, Vp);
      addr += sizeof(float);
      EEPROM.put(addr, Um);
      addr += sizeof(float);
      EEPROM.put(addr, deltaTime);
      addr += sizeof(float);
      EEPROM.put(addr, Frequence);
      addr += sizeof(float);
      EEPROM.put(addr, StartSend);
         }
  }
void loop()
{
recieveData();
}
```