Министерство науки и высшего образования Российской Федерации
Рязанский государственный радиотехнический университет имени В.Ф.Уткина

Промышленные сети

Методические указания по лабораторным работам
по курсу «Промышленные сети»

Рязань 2022

Методические указания предназначены для выполнения практических работ, целью которых является изучение методов взаимодействия с периферийными устройствами ЭВС. Содержат описание архитектуры микроконтроллера семейства Atmel, введение в интегрированную среду разработки программ IDE Arduino для этого семейства, методов работы с современными периферийными устройствами, пояснения к разработке учебных программ и задания для самостоятельной работы.

ОГЛАВЛЕНИЕ

1. ЦЕЛЬ И СОДЕРЖАНИЕ РАБОТ 5
2. АРХИТЕКТУРА МАКЕТНЫХ ПЛАТ ДЛЯ МИКРОКОНТРОЛЛЕРОВ 5
2.1. Архитектура открытой платформы макетных плат Arduino 6
2.2. Архитектура микроконтроллера ATmega32 8
3. ВВЕДЕНИЕ В IDE ARDUINO 12
3.1. Обзор учебной среды IDE Arduino 12
3.2. Язык С для Arduino 17
3.3. Структура программы в среде Arduino, переменные, операторы 18
3.4. Работа с последовательным портом 21
3.5. Ветвления и циклы в программе 22
3.6. Таймеры 24
3.7. Порты ввода/вывода 25
3.8. Генератор ШИМ 26
3.9. Генератор случайных чисел 26
3.10. Аппаратные прерывания 26
4. БЫСТРЫЙ СТАРТ 27
4.1. Запуск Arduino и создание файла проекта 27
4.2. Компиляция и проверка 28
4.3. Загрузка проекта 28
4.4. Отладка по последовательному порту и вывод графиков 29
5. ДОКУМЕНТИРОВАНИЕ РАЗРАБОТКИ НА СХЕМАХ 29
5.1. Документирование электрических схем 30
5.2. Документирование алгоритмов для микроконтроллера 32
5.3. Элементная база практических работ 37
5.3.1. Матричная клавиатура 37
5.3.2. Интерфейсы UART, SPI и I2C 39
5.3.3. Светодиодные индикаторы 44
5.3.4. Жидкокристаллический дисплей 49
5.3.5. Ультразвуковой сонар (измерение расстояния) 51
6. ЗАДАНИЯ ПРАКТИЧЕСКИХ РАБОТ 55
6.1. Практическая работа №1 «Быстрый старт» 55
6.2. Практическая работа №2 «Основы программирования МК» 55
6.3. Практическая работа №3 «Матричная клавиатура» 56
6.4. Практическая работа №4 «Семисегментный индикатор» 57
6.5. Практическая работа №5 «Светодиодный индикатор» 58
6.6. Практическая работа №6 «Жидкокристаллический дисплей» 60
6.7. Практическая работа №7 «Измерение физических величин» 62
6.8. Практическая работа №8 «Проектирование электронного устройства» 63
7. СОДЕРЖАНИЕ ОТЧЁТА 64
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 64
ПРИЛОЖЕНИЕ 65

1. ЦЕЛЬ И СОДЕРЖАНИЕ РАБОТ

Целью работ является изучение методов взаимодействия с периферийными устройствами ЭВС, а также архитектуры микроконтроллеров популярного семейства Atmel и свободно распространяемой интегрированной среды IDE Arduino, предназначенной для разработки программного обеспечения для этого семейства. Каждая практическая работа рассчитана на 1 час домашней подготовки и 2 часа практических занятий.

Для подготовки к практическим работам необходимо ознакомиться с заданием в разделе 6 и выполнить «домашнее задание».

В лаборатории в соответствии с разделом 6, выполняется практическая работа, и после выполнения работы оформляется отчёт с указанным в разделе 7 содержанием.

2. АРХИТЕКТУРА МАКЕТНЫХ ПЛАТ ДЛЯ МИКРОКОНТРОЛЛЕРОВ

В микропроцессорной технике выделился самостоятельный класс интегральных схем – микроконтроллеры, которые предназначены для встраивания в приборы различного назначения. От класса однокристальных микропроцессоров их отличает наличие встроенной памяти, развитые средства взаимодействия с внешними устройствами [1].

В настоящее время получили большое распространение так называемые **отладочные платы** - печатные платы, на которых размещают микроконтроллеры, интерфейсы по его программированию и удобные коннекторы-выводы контактов микроконтроллера для подключения к исследуемой (разрабатываемой) схеме «беспаечным» методом. Для учебных целей создана линейка отладочных плат Arduino. Семейство Arduino – несколько моделей так простых отладочных плат в которых используются микроконтроллеры AVR (UNO, Nano, Mega, Leonardo), в современных моделях установлены более мощные ARM Cortex для более серьёзных проектов.

Отладочная плата семейства Arduino предназначена в первую очередь для обучения программированию микроконтроллеров и созданию микроконтроллерных устройств учениками и студентами без суровой инженерной подготовки.

В общем виде отладочная плата для микроконтроллера состоит из центрального микроконтроллера (Microcontroller) с дополнительными окружающими его компонентами и цепями для взаимодействия с компьютером, которые можно также использовать для программирования микроконтроллера. Для подключения и компьютеру и программирования микроконтроллера в плату встроен преобразователь интерфейса USB в логику TTL (USB - TTL). Для обеспечения управления подачей питающего напряжения от USB/DC_IN/Vin, а также регулирования/преобразования напряжений 12/9/5/3.3 Вольт на плате имеются цепи питания (Power Circuit). Для подключения к микроконтроллеру устройств ввода/вывода на плате имеются «беспаечные» контакты (Input/Output Ports). Структура аппаратной части типовой отладочной платы показана на рисунке 1.

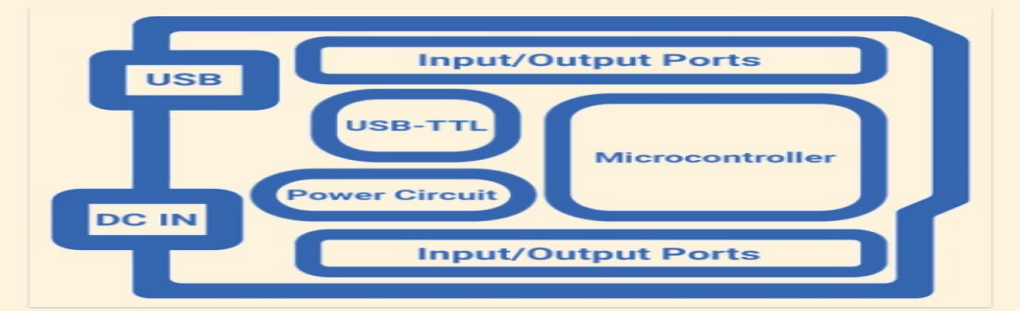


Рисунок 1 – типовая архитектура отладочных плат.

2.1. Архитектура открытой платформы макетных плат Arduino

Ардуино является открытой платформой, поэтому модельный ряд постоянно пополняется неофициальными платами от других производителей, такие платы называют “Arduino-совместимыми”. С ними можно работать в официальной программе Arduino IDE, писать на том же языке с тем же набором команд и даже использовать те же библиотеки. В качестве примера: это платы Teensy, платы на базе МК esp32 и esp8266 (Wemos, NodeMCU), различных китайских клонов и так далее.

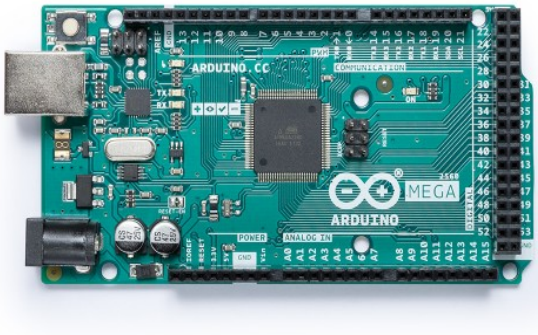
В данном практическом курсе, для простоты (и дешевизны) будем использовать микроконтроллеры AVR. На рисунках представлена линейка отладочных плат UNO, Nano, Mega, Leonardo.



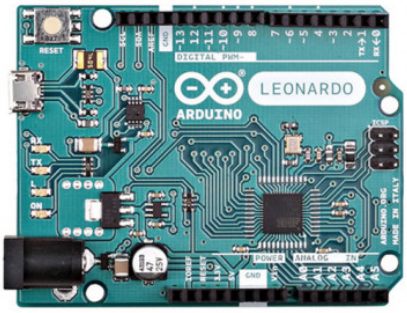
а)



б)



в)



г)

Рисунок 1 – (продолжение) отладочные платы UNO (а), Nano (б), Mega (в), Leonardo(г).

Ниже приведены характеристики этих отладочных плат [2].

Arduino Uno - контроллер построен на **ATmega328** . Платформа имеет 14 цифровых вход/выходов (6 из которых могут использоваться как выходы ШИМ), 6 аналоговых входов, кварцевый генератор 16 МГц, разъем USB, силовой разъем, разъем ICSP и кнопку перезагрузки. Для работы необходимо подключить платформу к компьютеру посредством кабеля USB, либо подать питание при помощи адаптера AC/DC или батареи.

В отличие от всех предыдущих плат, использовавших FTDI USB микроконтроллер для связи по USB, новый **Ардуино Uno** использует микроконтроллер **ATmega8U2** .

Характеристики:

Микроконтроллер	ATmega328
Рабочее напряжение	5 В
Входное напряжение (рекомендуемое)	7-12 В
Входное напряжение (предельное)	6-20 В
Цифровые Входы/Выходы	14 (6 из которых могут использоваться как выходы ШИМ)
Аналоговые входы	6
Постоянный ток через вход/выход	40 мА
Постоянный ток для вывода 3.3 В	50 мА
Флеш-память	32 Кб (ATmega328) из которых 0.5 Кб используются для загрузчика
ОЗУ	2 Кб (ATmega328)
EEPROM	1 Кб (ATmega328)
Тактовая частота	16 МГц

Платформа Nano, построенна на микроконтроллере ATmega328 (Arduino Nano 3.0) или ATmega168 (Arduino Nano 2.x), имеет небольшие размеры и может использоваться в лабораторных работах. Она имеет схожую с Arduino Duemilanove функциональность, однако отличается сборкой. Отличие заключается в отсутствии силового разъема постоянного тока и работе через кабель Mini-B USB. Nano разработана и продается компанией Gravitech.

Краткие характеристики

Микроконтроллер	Atmel ATmega168 или ATmega328
Рабочее напряжение (логическая уровень)	5 В
Входное напряжение (рекомендуемое)	7-12 В
Входное напряжение (предельное)	6-20 В
Цифровые Входы/Выходы	14 (6 из которых могут использоваться как выходы ШИМ)
Аналоговые входы	8
Постоянный ток через вход/выход	40 мА
Флеш-память	16 Кб (ATmega168) или 32 Кб (ATmega328) при этом 2 Кб используются для загрузчика
ОЗУ	1 Кб (ATmega168) или 2 Кб (ATmega328)
EEPROM	512 байт (ATmega168) или 1 Кб (ATmega328)
Тактовая частота	16 МГц
Размеры	1.85 см x 4.2 см

Arduino Mega построена на микроконтроллере ATmega2560. Плата имеет 54 цифровых входа/выходов (14 из которых могут использоваться как выходы ШИМ), 16 аналоговых входов,4 последовательных порта UART, кварцевый генератор 16 МГц, USB коннектор, разъем питания, разъем ICSP и кнопка перезагрузки. Для работы необходимо подключить платформу к компьютеру посредством кабеля USB или подать питание при помощи адаптера AC/DC, или аккумуляторной батареей. Arduino Mega 2560 совместима со всеми платами расширения, разработанными для платформ Uno или Duemilanove.

Краткие характеристики

Микроконтроллер	ATmega2560
Рабочее напряжение	5В
Входное напряжение (рекомендуе-	7-12В

мое)

Входное напряжение (предельное)	6-20В
Цифровые Входы/Выходы	54 (14 из которых могут работат также как выходы ШИМ)
Аналоговые входы	16
Постоянный ток через вход/выход	40 mA
Постоянный ток для вывода 3.3 В	50 mA
Флеш-память	256 KB (из которых 8 KB используются для загрузчика)
ОЗУ	8 KB
Энергонезависимая память	4 KB
Тактовая частота	16 MHz

Arduino Leonardo — контроллер на базе **ATmega32u4** (техническое описание, pdf). Платформа имеет 20 цифровых вход/выходов (7 из которых могут использоваться как выходы ШИМ и 12 как аналоговые входы), кварцевый генератор 16 МГц, разъем micro-USB, силовой разъем, разъем ICSP и кнопку перезагрузки. Для работы необходимо подключить платформу к компьютеру посредством кабеля USB, либо подать питание при помощи адаптера AC/DC или батареи.

В отличие от всех предыдущих плат **ATmega32u4** имеет встроенную поддержку для USB соединения, это позволяет задать как **Leonardo** будет виден при подключение к компьютеру, это может быть клавиатура, мышь, виртуальный серийный / COM порт.

Характеристики:

Микроконтроллер	ATmega32u4
Рабочее напряжение	5 В
Входное напряжение (рекомендуемое)	7-12 В
Входное напряжение (предельное)	6-20 В
Цифровые Входы/Выходы	20 (7 из которых могут использоваться как выходы ШИМ)
Аналоговые каналы	12
Постоянный ток через вход/выход	40 мА
Постоянный ток для вывода 3.3 В	50 мА
Флеш-память	32 Кб (ATmega32u4) из которых 4 Кб используются для загрузчика
ОЗУ	2 Кб (ATmega32u4)
EEPROM	1 Кб (ATmega32u4)
Тактовая частота	16 МГц

2.2. Архитектура микроконтроллера ATmega32

Микроконтроллер семейства ATmega32 – это высокопроизводительные, мало потребляющие AVR 8- битные микроконтроллеры с гарвардской архитектурой имеющие следующие основные характеристики [3]:

– Развитая RISC архитектура:

- 131 исполняемых команд, большинство за один машинный такт
- 32 рабочих регистра общего назначения

- полностью статический режим работы
- производительность до 16 MIPS при 16 МГц
- встроенный 2-х тактовый умножитель

– Энергонезависимая память программ и данных:

- 32К байт внутрисистемно самопрограммируемой FLASH памяти с количеством циклов перепрограммирования до 10 000.
- Опционно загрузочная область памяти с независимыми ключевыми битами, внутрисистемное программирование встроенной загрузочной программой, правильное чтение в процессе записи.
- 1024 байт EEPROM с допустимым количеством циклов стирания записи до 100 000.
- 2К байт внутренней SRAM
- программируемый ключ защиты программ

– JTAG (IEEE1149.1 совместимый) интерфейс

- Сканирование памяти в соответствии с JTAG стандартом
- Встроенная поддержка отладчика
- Программирование FLASH, EEPROM, охранных и ключевых бит через JTAG интерфейс

– Периферийные функции

- два 8-битных таймера/счётчика с программируемым предделителем и режимом сравнения
- один 16-битный таймер/счётчик с программируемым предделителем, режимом сравнения и захвата
- счётчик реального времени с программируемым генератором
- четыре ШИМ генератора
- 8-и канальный, 10-и битный АЦП
- байт- ориентированный, двухпроводный интерфейс
- программируемый USART
- Master/Slave SPI последовательный интерфейс
- Программируемыи Watchdog таймер с программируемым генератором
- Встроенный аналоговый компаратор

– Специальные функции

- Reset по включению питания и выключение при снижении напряжения питания
- Внутренний калиброванный RC генератор
- Внешние и внутренние источники прерывания
- Шесть экономичных режимов: Idle, подавления шумов АЦП, экономичный, режим Выкл. , режим ожидания и режим расширенного ожидания.

– 32 программируемых вывода вход-выход и 1 вход
– 40 выводной корпус PDIP, 44 выводной корпус TQFP, и 44 контактный MLF
– Напряжение питания:

- 2.7 В до 5.5 В для ATmega32L
- 4.5 В до 5.5 В для Atmega32

– Тактовая частота:

- 0-8 МГц Atmega32L
- 0-16 МГц Atmega32

Ниже приведена блок- схема микроконтроллера (Рисунок 2) и схемы расположения выводов (Рисунок 3).

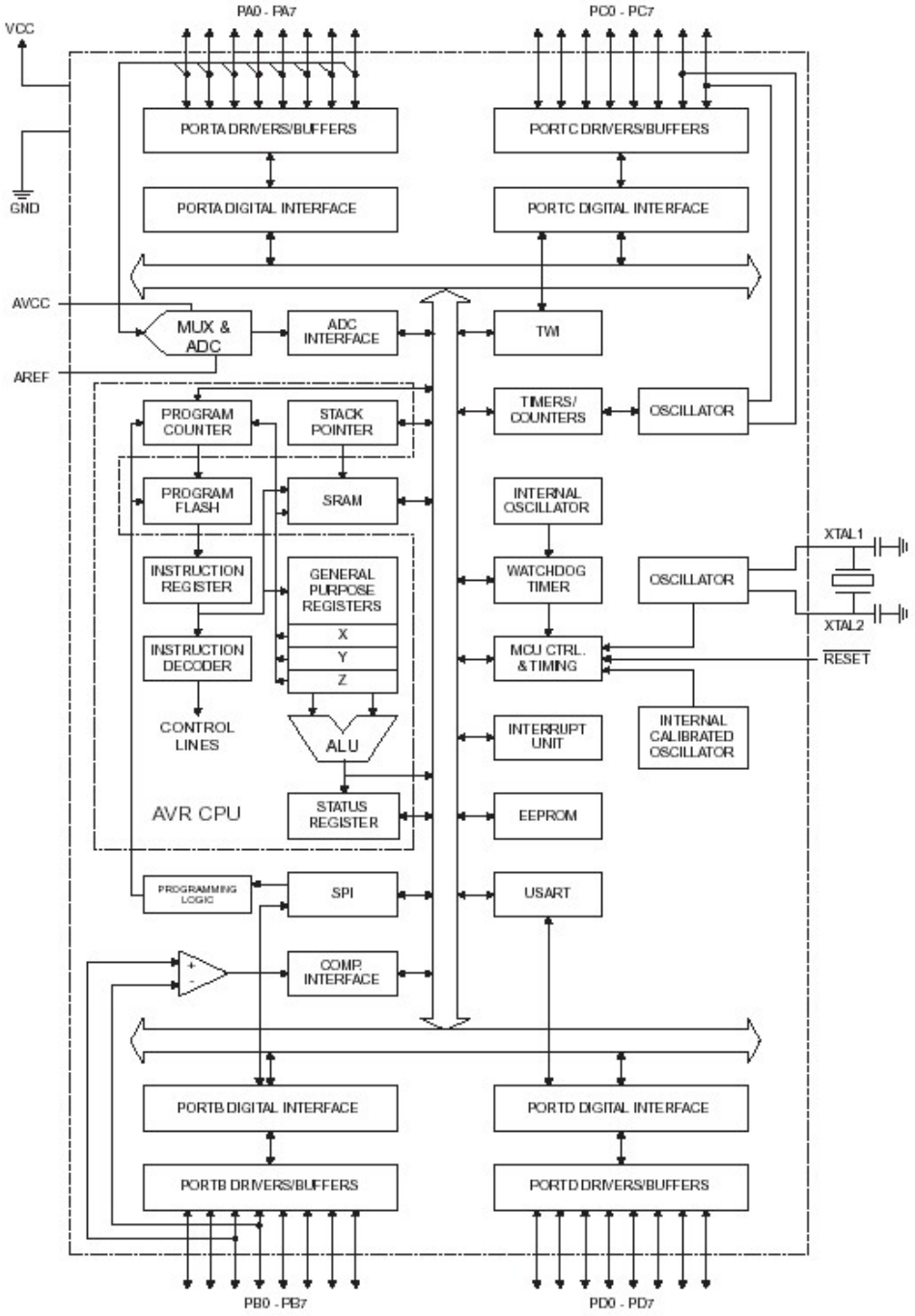


Рисунок 2 - блок- схема микроконтроллера

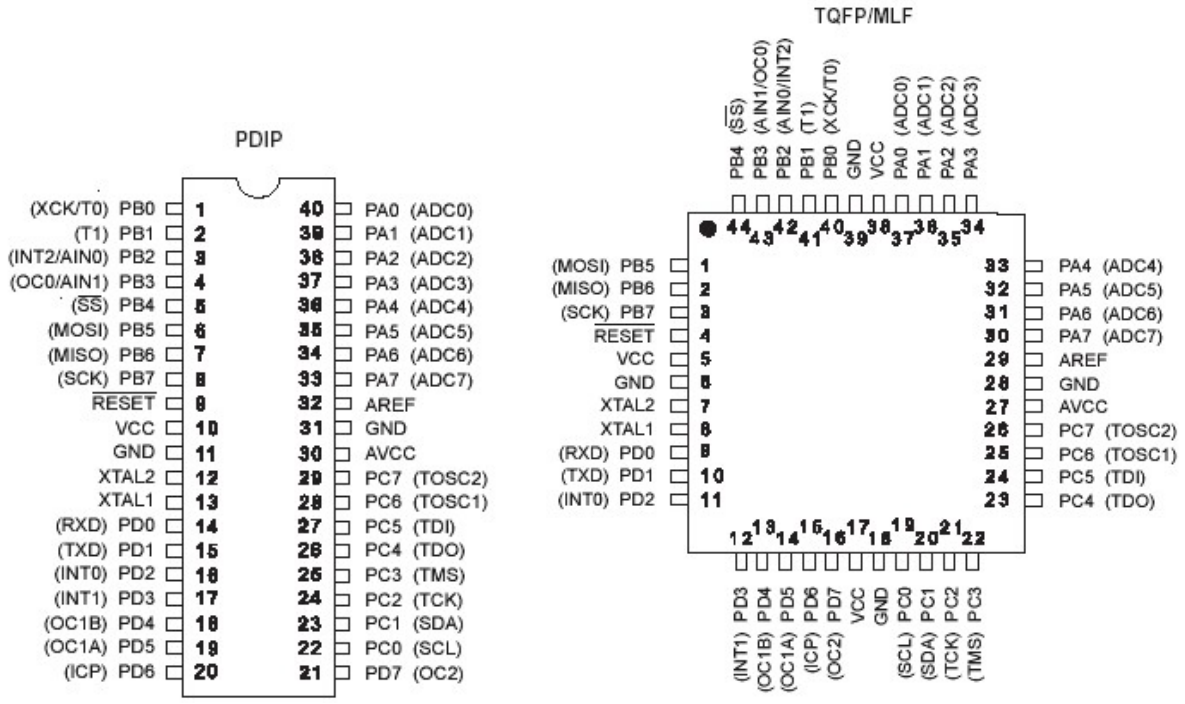


Рисунок 3 - Расположение выводов.

Описание:

Atmega32/L является КМОП 8- битным микроконтроллером построенным на расширенной AVR RISC архитектуре. Используя команды исполняемые за один машинный такт, контроллер достигает производительности в 1 MIPS на рабочей частоте 1 МГц, что позволяет разработчику эффективно оптимизировать потребление энергии за счёт выбора оптимальной производительности.

AVR ядро сочетает расширенный набор команд с 32 рабочими регистрами общего назначения. Все 32 регистра соединены с АЛУ, что обеспечивает доступ к двум независимым регистрам на время исполнения команды за один машинный такт. Благодаря выбранной архитектуре достигнута наивысшая скорость кода и соответственно высокая производительность в 10 раз превосходящая скорость соответствующего CISC микроконтроллера.

ATmega32/L содержит 32Кбайт внутрисистеммно программируемой FLASH памяти программ, допускающей чтение во время записи, 1024 байт EEPROM, 2К байт SRAM , 32 рабочих регистра, JTAG интерфейс сканирования внутренних регистров, встроенную систему отладки и программирования, три гибких таймера- счётчика с модулем сравнения, внутренние и внешние прерывания, последовательный программируемый интерфейс USART, байт-ориентированный двухпроводный последовательный интерфейс, 8-и канальный, 10-и битный АЦП с дифференциальным программируемым усилителем (только для TQFP), программируемый Watchdog таймер с внутренним генератором, порт SPI и шестью режимами сбережения энергии. В режиме Idle ЦПУ не функционирует в то время как функционируют USART, двухпроводный интерфейс, АЦП, SRAM, таймеры- счётчики, SPI порт и система прерываний. В Atmega32 существует специальный режим подавления шума АЦП, при этом в целом в спящем режиме функционирует только АЦП и асинхронный таймер для уменьшения цифровых шумов преобразования. В режиме Выкл. процессор сохраняет содержимое всех регистров, замораживает генератор тактовых сигналов, приостанавливает все другие функции кристалла до прихода внешнего прерывания или поступления внешней команды Reset. В режиме ожидания работает один тактовый генератор, при остановке остальных функций контроллера. Благодаря быстрому переходу

в нормальный режим работы в том числе и по внешнему прерыванию Atmega32 успешно приспосабливается к внешним условиям работы и требует меньше энергии, чаще оказываясь в режиме Выкл. В расширенном режиме ожидания в рабочем состоянии находятся основной генератор и асинхронный генератор.

Микросхемы выпускаются при использовании Atmel технологии энергонезависимой памяти высокой плотности. Встроенная ISP FLASH память позволяет перепрограммировать область программной памяти внутрисистеммно через последовательный SPI интерфейс стандартным программатором, или используя загрузочную программу из энергонезависимой памяти работающую в AVR ядре. Комбинация расширенной 8- и битной RISC архитектуры ЦПУ и твёрдотельной FLASH памяти обеспечивают Atmega32 высокую гибкость и экономическую эффективность во встраиваемых системах управления.

3. ВВЕДЕНИЕ В IDE ARDUINO

IDE Arduino – интегрированная среда разработки программного обеспечения для современных однокристальных микроконтроллеров с использованием платформы отладочных плат. Она включает в себя всё, что нужно для создания, редактирования, компиляции, трансляции, компоновки, загрузки и отладки программ:

- стандартный интерфейс Windows,
- полнофункциональный редактор исходных текстов с выделением синтаксических элементов цветом,
- организатор проекта,
- транслятор с языка С,
- монитор последовательного порта и плоттер последовательного порта,
- отладчик,
- встроенную справочную систему.

3.1. Обзор учебной среды IDE Arduino

Меню IDE Arduino имеет следующие компоненты:

- Файл;
- Настройка среды
- Правка;
- Скетч;
- Инструменты;
- Помощь.

Рассмотрим более подробно меню программы.

1) Элемент меню **Файл** предназначен для работы с файлом скетча:

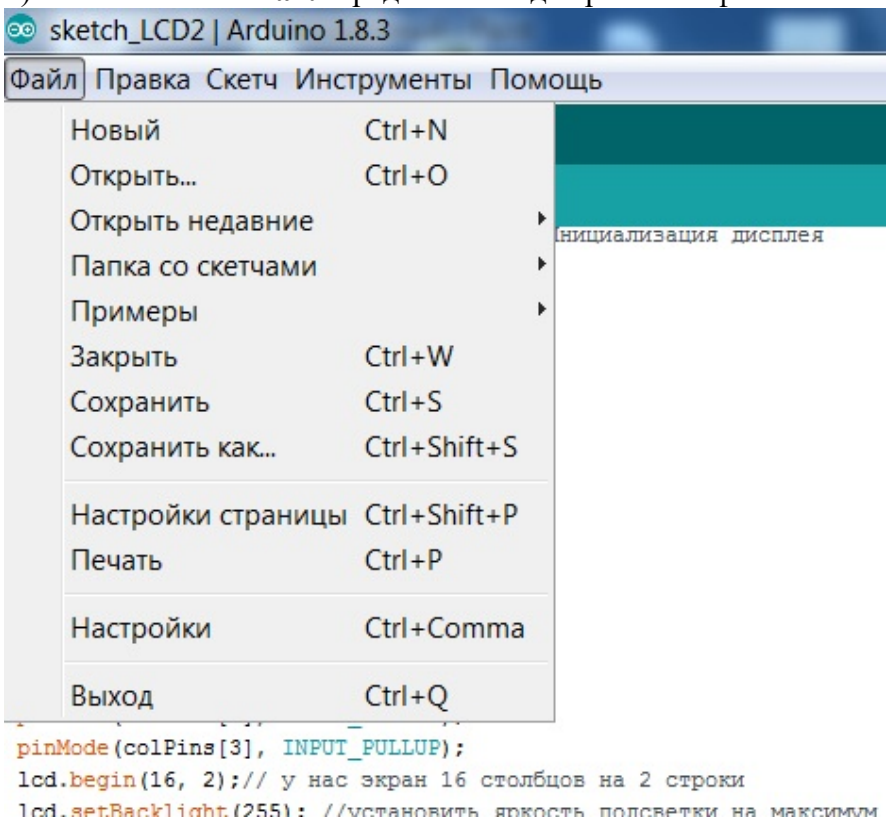


Рисунок 4 - Элемент меню **Файл**.

Основным, заслуживающим внимания компонентом этого подменю, является элемент «Настройка», вызывающий следующую экранную форму.

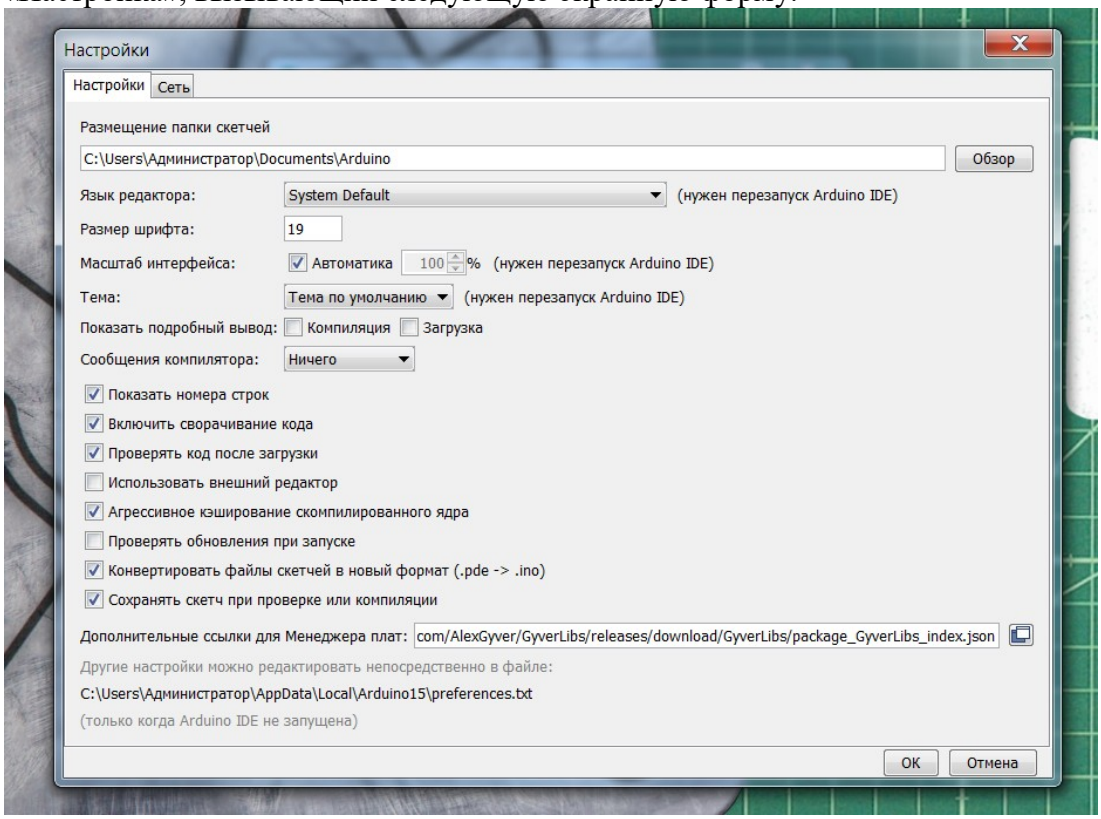


Рисунок 5 – Экранная форма **Настройки**.

Экранная форма «Настройки» предназначена для:

– Размещение папки скетчей – куда по умолчанию сохраняются скетчи;
– Показать подробный вывод – показывает подробный лог при компиляции и загрузке, нужно при поиске багов и непонятных ошибок;
– Использовать внешний редактор – запрещает редактирование кода из Arduino IDE, чтобы редактировать его в других редакторах, например Notepad++. Редактируем там, а загружаем через IDE;
– Дополнительные ссылки для менеджера плат – сюда вставляются ссылки на пакеты для работы с другими платами, например, основанными на ESP8266 или ATtiny85.

2) Элемент меню **Правка** предназначен для работы в редакторе среды:

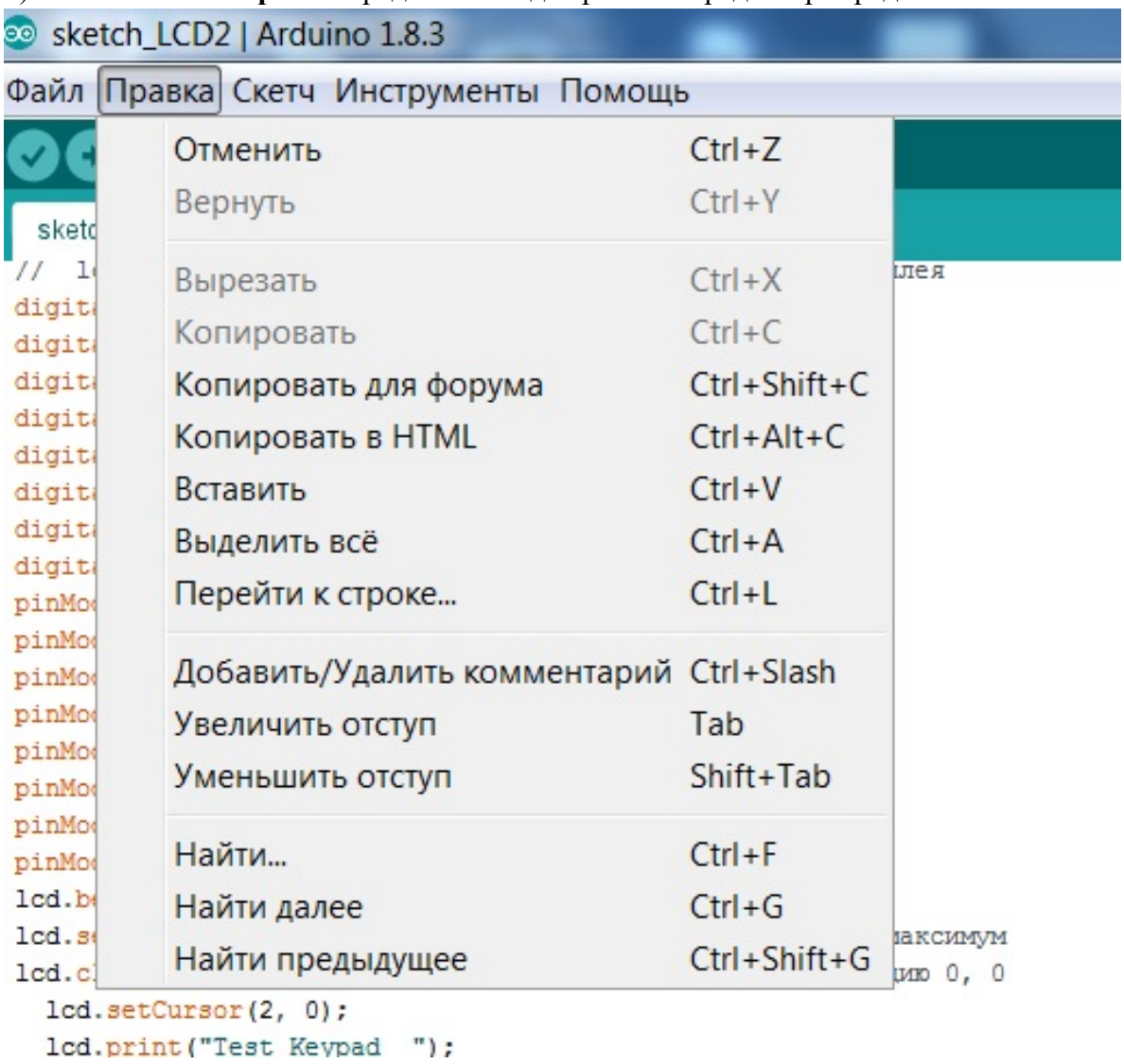


Рисунок 6 - Элемент меню **Правка**.

3) Элемент меню **Скетч** предназначен для работы с компилятором и загрузчиком:

– Проверить/компилировать – проверка кода и компиляция (то же, что кнопка “галочка”);
– Загрузка – загрузка откомпилированного кода в микроконтроллер, через USB (то же, что кнопка “стрелочка”);
– Загрузить через программатор – загрузить скетч напрямую в микроконтроллер

через программатор на отладочной плате;

– Экспорт бинарного файла – компилирует код и сохраняет в бинарник;
– Показать папку скетча;
– Подключить библиотеку – подключает в код выбранную библиотеку.

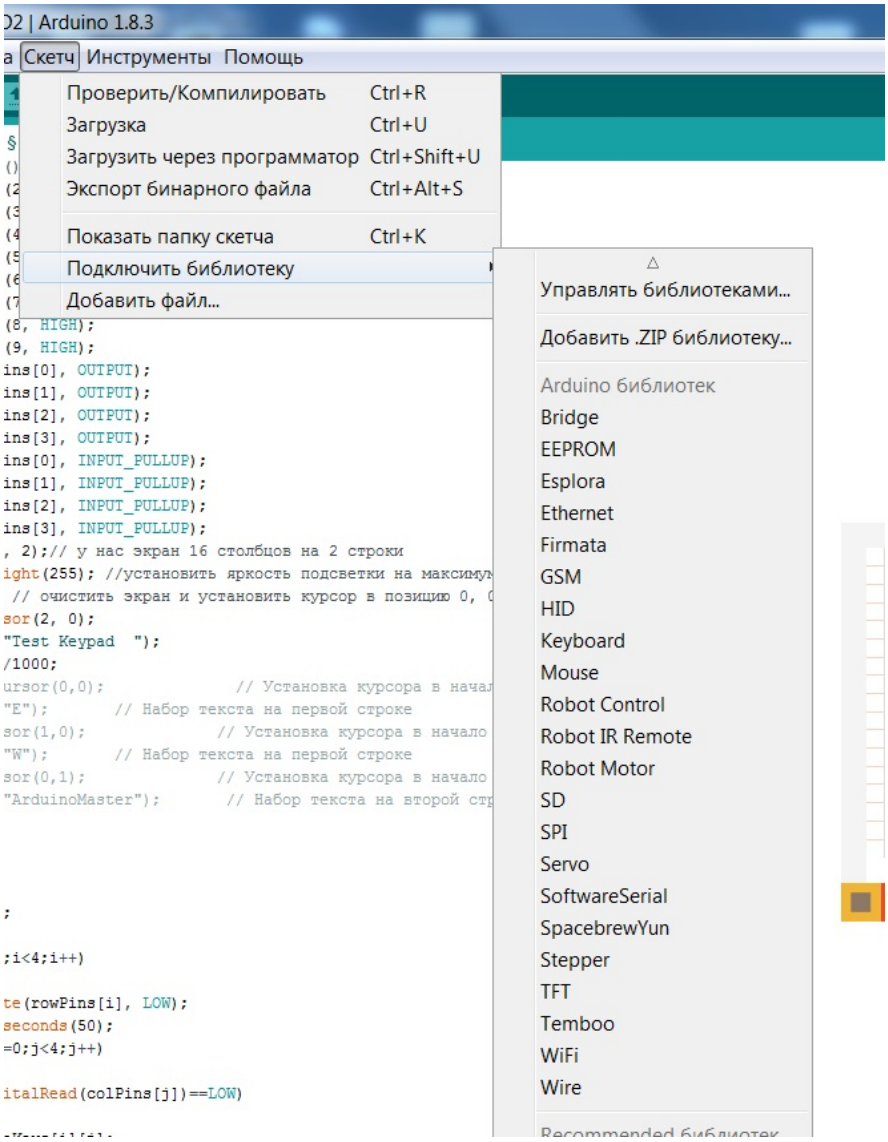


Рисунок 7 - Элемент меню **Скетч**.

– Управлять библиотеками… – открывает менеджер библиотек, из которого можно устанавливать библиотеки.
– Добавить .zip библиотеку – установка библиотеки из zip файла, если вы его откуда-то скачали.
– Добавить файл… – подключает в код внешний файл.

4) Элемент меню **Инструменты** предназначен для использования важных инструментов:

- АвтоФорматирование – выравнивает код по табуляции. Это крайне важный инструмент, используйте его (комбинация Ctrl+T) как можно чаще.
- Архивировать скетч – сохраняет скетч как .zip архив.
- Исправить кодировку и перезагрузить – полезная функция, когда скачал чей-то код из интернета, а там испортилась кодировка.
- Управлять библиотеками… – открывает менеджер библиотек, из которого можно устанавливать библиотеки.
- Монитор порта - монитор последовательного порта для отображения отладки (serial).
- Плоттер по последовательному соединению – встроенный построитель графиков по идущим в порт данным (данные должны разделяться через пробел).
- Плата – выбор платы, куда хотим загрузить прошивку.
- Порт – COM порт, к которому подключена плата. Это всегда порт, отличный от COM1 (системный порт)
- Программатор – выбор программатора для загрузки кода через программатор.
- Записать загрузчик – прошивает загрузчик, соответствующий выбранной плате и процессору в микроконтроллер при помощи программатора.

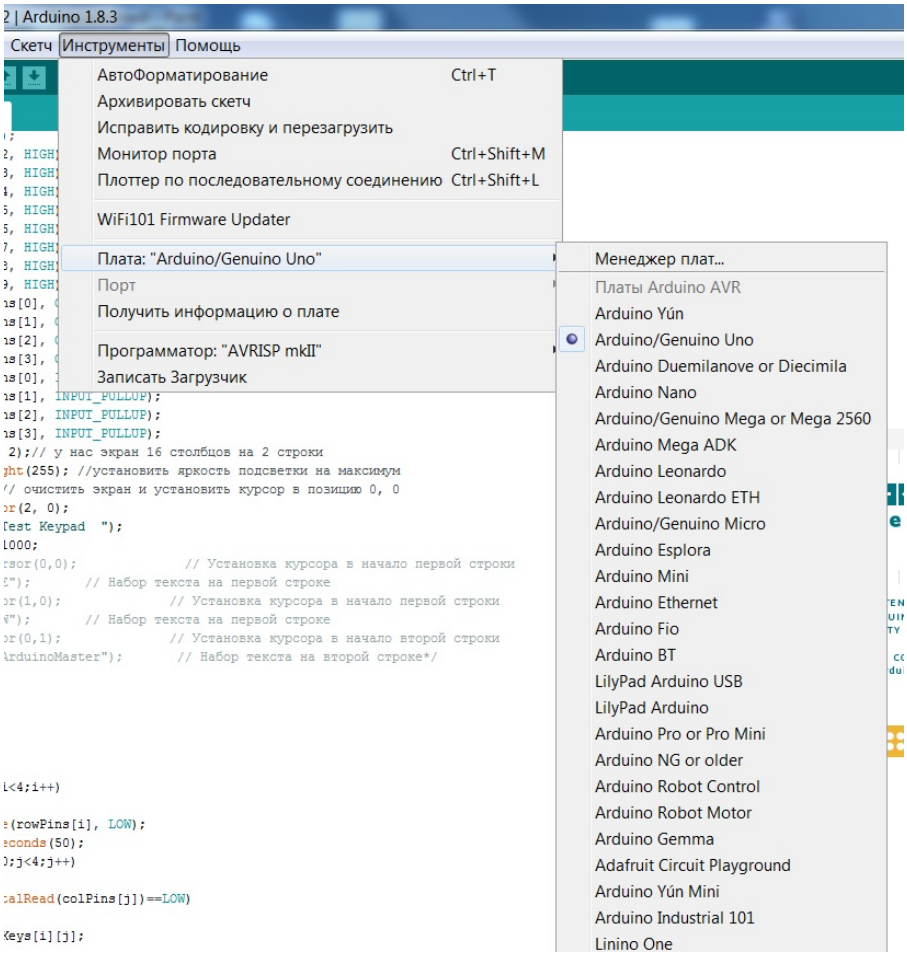


Рисунок 8 - Элемент меню **Инструменты**.

5) Элемент меню **Помощь** предназначен вызова On-line справок:

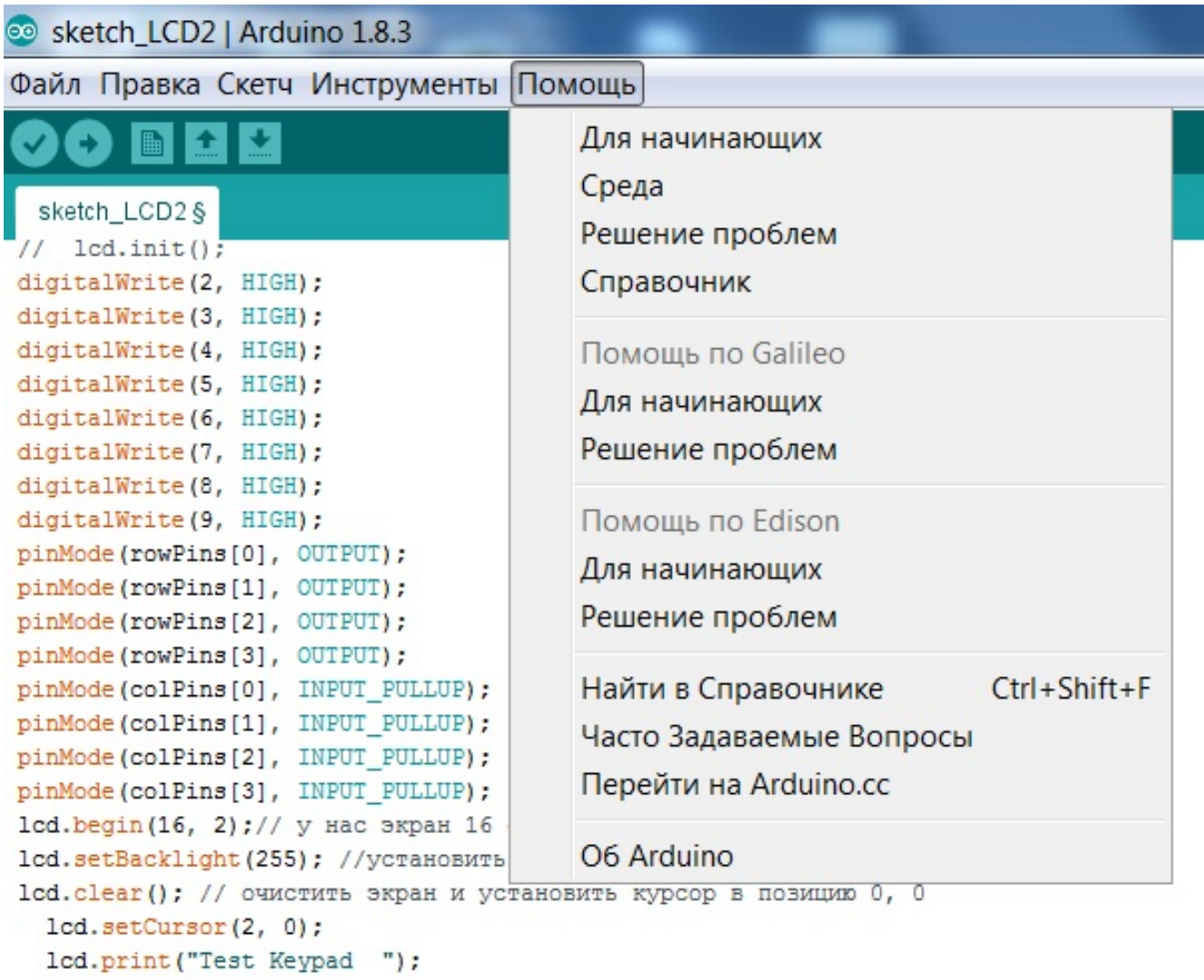


Рисунок 9 - Элемент меню **Помощь**.

3.2. Язык С для Arduino

Язык С - универсальный язык программирования, который обеспечивает эффективность кода, элементы структурного программирования и имеет богатый набор операторов. Универсальность, отсутствие ограничений реализации делают язык С удобным и эффективным средством программирования для широкого разнообразия задач. Множество прикладных программ может быть написано легче и эффективнее на языке С, чем на других более специализированных языках.

С для Arduino - полная реализация стандарта ANSI (Американского национального института стандартов), насколько это возможно для архитектуры отладочных плат Arduino.

3.3. Структура программы в среде Arduino, переменные, операторы

Программа для микроконтроллера в среде Arduino упрощенно выглядит следующим образом, показанным на рисунке:

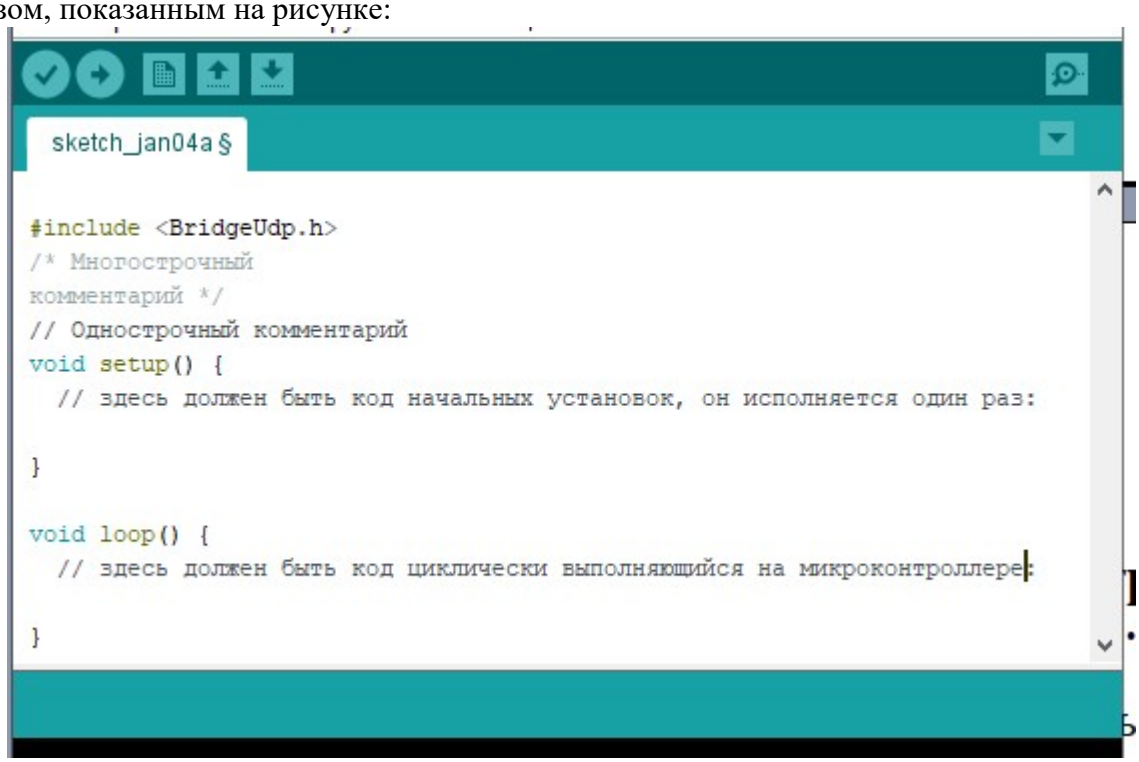


Рисунок 10 – Структура программы.

Для начала необходимо запомнить 6 основных элементов программы:

- **// ТЕКСТ** - это однострочный комментарий (любой комментарии не занимают память, важно снабжать программу достаточными комментариями);
- **/* ТЕКСТ */** - это многострочный комментарий (комментарии не занимают память);
- **#include <БИБЛИОТЕКА>** - подключить файл (библиотеку);
- **void setup() {}** – начальная загрузка, всё находящееся внутри {} будет выполнено 1 раз при загрузке Ардуино;
- **void loop() {}** – цикл микроконтроллера, всё находящееся внутри {} бесконечно повторяется от начала до конца;
- После каждого «действия» (оператора) ставится точка с запятой «;».

Объявление переменных:

В любом месте программы могут быть объявлены переменные с необходимым типом.

Переменная объявляется вот таким образом: <тип данных> <имя>;

Пример: **int val;** // объявить переменную val

Также можно сразу присвоить значение <тип данных> <имя> = <значение>;

Пример: **int val = 200;** // объявить переменную val и присвоить ей число 200

Также можно объявить несколько переменных одного типа сразу

Пример: int val = 200, val2, val3, val4 = 5; // объявить переменные

В нижеприведенной таблице приведены типы данных, применяемых в языке С для

Arduino.

Таблица 1 - типы данных.

Название	Альтернативное название	Вес	Диапазон	Особенность
boolean		1 байт	0 или 1	Логическая переменная, может принимать значения **true** (1) и **false** (0)
char	int8_t	1 байт	-128... 127	Хранит номер символа из таблицы символов ASCII
byte	uint8_t	1 байт	0... 255	
int	int16_t	2 байта	-32 768... 32 767	
unsigned int	uint16_t	2 байта	0... 65 535	
word		2 байта	0... 65 535	То же самое, что unsigned int
long	int32_t	4 байта	-2 147 483 648... 2 147 483 647	- 2 миллиарда... 2 миллиарда
unsigned long	uint32_t	4 байта	0... 4 294 967 295	0... 4 миллиарда...
float		4 байта	-3.4028235E+38... 3.4028235E+38	Хранит числа с плавающей точкой (десятичные дроби). **Точность: 6-7 знаков**
double		4 байта		То же самое, что float

Особенности float (необходимо помнить):

- Что бы вычисления были нецелыми необходимо в выражениях указывать только значение с точкой, даже если оно целое (10.0);
- Делить тоже только на числа с точкой, даже если они целые (переменная / 2.0);
- При делении целочисленного типа с целью получить число с плавающей точкой, перед переменной необходимо написать (float), т.е. привести целочисленную переменную в переменную с плавающей точкой;
- Необходимо помнить, что вы работаете с микроконтроллером в котором нет математического сопроцессора. Все операции с плавающей точкой трансформируются в сложные унифицированные процедуры, которые оперируют с целыми числами. Поэтому операции с числами типа float занимают гораздо больше времени, чем с целыми. Если нужна высокая скорость вычислений, лучше применять всякие хитрости, в стиле выполнения всех вычислений типом long, и результат уже переводить во float.

Типы переменных:

Глобальная переменная – объявляется ВНЕ функций, например в самом начале скетча или между функциями. Обращаться к глобальной переменной (использовать её значение) можно использовать ВЕЗДЕ.

Локальная переменная – объявляется ВНУТРИ функции, и обращаться к ней можно только внутри этой функции.

Локальных переменных может быть несколько с одинаковым именем (внутри разных функций), но разными значениями. Это связано с тем, что локальная переменная выгружается из оперативной памяти микроконтроллера при выходе из функции.

Константы:

Константа может быть объявлена следующими конструкциями:

1 вариант:

const <тип> <имя> = <значение>; - объявить константу

const int val = 200; // объявить константу val и присвоить ей число 200

2 вариант:

#define <имя> <значение> - объявить константу через define (при этом в конце строки точка запятой не нужна)

#define val 200 // определить константу val и присвоить ей число 200

Второй вариант отличается от первого тем, что константа, объявленная через #define, работает немного по-другому: на этапе компиляции кода указанное ИМЯ заменяется на его значение, все это сохраняется в памяти микроконтроллера вместе с программой.

В первом случае константа помещается в сегмент данных.

И в том и другом случае при попытке сменить значение константы после её объявления, вы получите ошибку.

Создание массива

Массив объявляется следующим образом:

<тип данных> <имя массива>[<число элементов>];

<тип данных> <имя массива>[<число элементов>] = {элемент1, элемент2…};

Если не указываются элементы, то обязательно нужно указать размер массива, чтобы под него выделилось место в памяти. Размер можно не указывать в том случае, если сразу указываются все элементы. Примеры:

```
int ArrInt[10];
int ArrInt2[] = {2, 4, 1, 2, 3};
int ArrInt3[5] = {-2, 4, 1, 3, 2};
char mess[7] = "message";
```

Математические операторы

№	Математическая операция	Описание
1	**+, -, *, /**	сложить, вычесть, умножить, разделить
2	**pow(x, a);**	возвести "x" в степень "a" (x_a), **pow** может возводить в дробную степень
3	**sq(x);**	возвести число "x" в квадрат (x_2)
4	**sqrt(x);**	взять квадратный корень числа "x"
5	**abs(x);**	найти модуль числа, |x|
6	**sin(x), cos(x), tan(x);**	синус, косинус, тангенс
7	**round(x);**	математическое округление (если после запятой больше или равно 5, то округляем в большую сторону)
8	**ceil(x);**	округлить в бОльшую сторону
9	**floor(x);**	округлить в меньшую сторону
10	**x += a;**	прибавить "a" к "x"
11	**x -= a;**	вычесть "a" из "x"
12	**x *= a;**	домножить "x" на "a"
13	**x /= a;**	разделить "x" на "a"
14	**x++;**	увеличить "x" на 1
15	**x--;**	уменьшить "x" на 1

3.4. Работа с последовательным портом

За работу с последовательным портом отвечают операторы библиотеки Serial. Serial - объект библиотеки Serial для работы с последовательным портом (COM портом). Наиболее важны следующие операторы:

1) Serial.begin(<скорость>); - открыть порт

Пример: Serial.begin(115200); // открыть порт на 115200 БОД

Скорость, установленная в begin(), должна быть равна скорости монитора порта (в самом мониторе правый нижний угол).

2) Serial.print(x); - вывод в порт. Где x- переменные, цифры, и текст в кавычках " "

3) Serial.println(); - вывод с переводом строки, аналогично.

Одна передача данных локализуется переводом строки, поэтому для отображения на графике **Плоттер последовательного порта** переменные разделяются “ “ (пробелом).

4) Serial.println(val, n); - вывод переменной val (типа float) с n числом знаков после запятой

5) Serial.println(val, <базис>); // вывод с указанным базисом:

• DEC - десятичный (человеческие числа)

• HEX - 16-ричная система

• OCT - 8-ричная система

• BIN - двоичная система

Вывод данных производится в **Монитор порта** работающий на скорости 115200 БОД

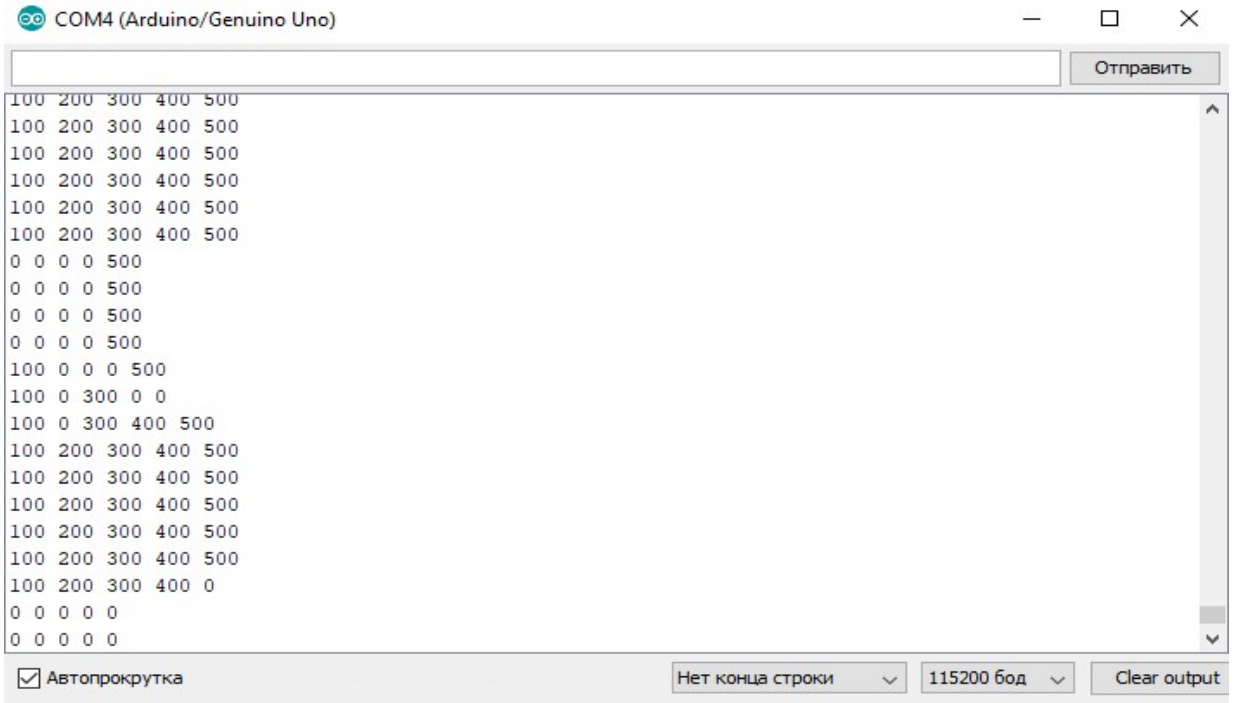


Рисунок 11 – Монитор порта.



Рисунок 12 – Плоттер.

Для передачи данных от компьютера (в мониторе порта) в микроконтроллер используются другие операторы, с помощью которых данные попадают в буфер с объёмом 64 байта, и ждут обработки CPU.

1) Serial.available(); - проверить буфер на наличие входящих данных

2) Serial.read(); - Считывает очередной доступный байт из буфера последовательного соединения. Фактически функция считывает входящие данные в символьном формате в кодировке ASCII

3) Serial.parseInt(); - прочитать данные в целочисленном формате.

4) Serial.parseFloat(); - прочитать данные в формате с плавающей точкой.

5) Serial.setTimeout(); - поставить таймаут для parseInt и parseFloat. Устанавливает **time** (миллисекунды) таймаут ожидания приёма данных для этих функций. *По умолчанию равен 1000 мс (1 секунда).*

6) Serial.flush(); - очистить буфер порта

3.5. Ветвления и циклы в программе

Для обеспечения ветвлений в программе очень часто необходимо использовать условия с применением операторов сравнения и логических операторов. В языке С используются следующие операторы сравнения и логические операторы.

Операторы сравнения

a == b - если a равно b

a != b - если a не равно b

> - если a больше b (строго)
< - если a меньше b (строго)
>= - если a больше или равна b
<= - если a меньше или равна b

Логические операторы

&& - логическое И (одно условие И второе)
| | - “два прямых слеша” логическое ИЛИ (либо одно, либо второе)
! - отрицание (например if (!val) - если val - ложь, т.е. 0).

Условный оператор «если»

1) **if () {}** - условный оператор «если, то», проверяет истинность выражения в круглых скобках () и если оно верно, то выполняет код в фигурных скобках {};

2) **if () {} else {}** - условный оператор «если (), то{}, иначе{}», проверяет истинность выражения в круглых скобках () и если оно верно, то выполняет код в фигурных скобках {} до **else**, иначе выполняет код в фигурных скобках после **else**;

3) **if () {} else if () {} else if () {} … else if () {} else {}** - условный оператор «если (), то{}, иначе если{} иначе {}», проверяет истинность выражения в круглых скобках () и если оно верно, то выполняет код в фигурных скобках {}, иначе происходят проверки других условий и если оно верно, то выполняет код в фигурных скобках {}, если же эти условия не истинны , то выполняет код в фигурных скобках после **else**;

В условии может быть как логическое выражение (a > b), так и логическая переменная со значением true или false. Или обычная переменная со значением 1 или 0.

Цикл «счетчик»

for (counter; condition; change) {} – конструкция цикла for, где

- counter – переменная цикла, обычно создают новую «локальную», в стиле int i =0;
- condition – условие, при котором продолжает выполняться цикл, например i < 12;
- change – изменение, т.е. увеличение или уменьшение переменной цикла, например i++, i--, i +=2;

Пример:

```
for (int i =0; i < 12; i++)
{ // фактически это реализации от 0 до 12 счетчика который используется в цикле
      Serial.println(i); // вывести в монитор порта числа от 0 до 12
}
```

Цикл «с предусловием»

while (condition) {} – конструкция цикла с предусловием, где

- condition – условие, при котором выполняется блок кода, заключённый в {}

Пример:

```
while (flag) {
// какой-то кусок кода, который выполняется, пока flag равен логической 1
}
```

Цикл «с постусловием»

do {} while (condition) – конструкция цикла с постусловием (в отличие от предыдущего цикла, выполнится хотя бы один раз, даже если условие не выполняется), где

- condition – условие, при котором выполняется блок кода, заключённый в {}

Пример:

```
do {
// какой-то кусок кода, который выполняется, пока flag равен логической 1
```

```
} while (flag);
```

Прерывание циклов

break – выход из цикла

continue – пропустить ход и начать цикл

Условный оператор «выбор»

Выбор ветвления программы в зависимости от значения целочисленной переменной или символьной переменной осуществляется с помощью конструкции «**switch .. case**»

```
switch (val) // проверяем целочисленную переменную val
{
  case 1: // если она равна 1, выполнить код здесь
        break;
  case 2: // если она равна 2, выполнить код здесь
        break;
…
  case n: // если она равна n, выполнить код здесь
        break;
  default: // если что-то ещё, выполнить код здесь (default необязателен)
        break;
}
```

3.6. Таймеры

Функции задержек

delay() - задержка, в скобках указывается число миллисекунд (в 1 сек 1'000 миллисекунд). Максимальное значение типа unsigned long (4 байта), 4’294’967’295 мс, или около 1200 часов, или 50 суток.

delayMicroseconds() - задержка, в скобках указывается число микросекунд (в 1 сек 1'000'000 микросекунд). Максимальное значение 16'383 мкс, или 16 миллисекунд.

Задержки приостанавливают основной процесс ***loop()*** на указанное время. Для того, чтобы микроконтроллер не простаивал, можно во время задержки продолжить выполнение других задач.

Процедурой, которая выполняется во время задержки является ***void yield () {}***.

Функции таймера

millis() - возвращает количество миллисекунд, прошедших с момента включения микроконтроллера.

- Макс. значение: 4’294’967’295 мс или 50 суток.
- Разрешение: 1 миллисекунда.

micros() - возвращает количество микросекунд, прошедших с момента включения микроконтроллера.

- Макс. значение: 4’294’967’295 мкс или 70 мин
- Разрешение: 4 микросекунды

3.7. Порты ввода/вывода

Режимы работы цифровых портов

Каждый аналоговый и цифровой ПИН может работать как INPUT (ВХОД) так и OUTPUT (ВЫХОД).

По умолчанию все ПИНы работают как INPUT.

Все ПИНы цифровые, но у некоторых есть АЦП (аналогово-цифровой преобразователь), эти ПИНы называют «аналоговыми». Условно их обозначают символом «А».

Для изменения режима используется оператор pinMode.

pinMode(pin, mode) - настроить порт

- pin - номер порта. Цифровые: 0 – 13. Аналоговые: 14 - 19, либо A0 - A5
- mode - режим работы порта INPUT / OUTPUT / INPUT_PULLUP:
  - INPUT - вход, принимает сигнал;
  - OUTPUT - выход, выдаёт 0 или 5 Вольт;
  - INPUT_PULLUP - вход с подтяжкой к 5 В.

Генерация цифрового сигнала

digitalWrite(pin, signal) - подать цифровой сигнал

- pin - номер порта. Цифровые: 0 – 13. Аналоговые: 14 - 19, либо A0 - A5
- signal - какой сигнал подаём LOW/ HIGH
  - LOW, или 0 (ноль), или false - 0 Вольт
  - HIGH, или 1, или true - 5 Вольт

Чтение цифрового сигнала

digitalRead(pin) - прочитать цифровой сигнал

- pin - номер порта. Цифровые: 0 – 13. Аналоговые: 14 - 19, либо A0 - A5

Чтение аналогового сигнала

analogRead(pin) - прочитать аналоговый сигнал (оцифровать)

- pin - номер пина. Аналоговые: 0 - 7

Функция возвращает значение 0.. 1023 в зависимости от напряжения на пине от 0 до опорного напряжения. По умолчанию опорное напряжение равно напряжению питания, при питании от USB это около 4.7 Вольт.

Можно выбрать источник опорного напряжения:

analogReference(DEFAULT) – напряжение питания как опорное (5-4.7 Вольт);

analogReference(INTERNAL) – встроенное опорное на 1.1 Вольт;

analogReference(EXTERNAL) – опорное напряжение подаётся на пин Aref.

Изменение диапазона измеренных значений

map(val, min, max, new_min, new_max) - возвращает величину в новом диапазоне

- val - входная величина
- min, max - минимальное и максимальное значение на входе в map
- new_min, new_max – соответственно мин. и макс. значения на выходе

Например, если нам нужно перевести цифровой диапазон датчика в процентный необходимо выполнить операцию : val=map(analogRead(0), 0, 1023, 0, 100);

constrain(val, min, max) - ограничить диапазон переменной val до min и max

- val - входная величина
- min, max - минимальное и максимальное значение на входе.

Например, если нам нужно ограничить цифровой диапазон датчика в диапазоне от 50 до 950 необходимо выполнить операцию : val=constrain(analogRead(0), 50, 950);

3.8. Генератор ШИМ

analogWrite(pin, duty) - генерация ШИМ сигнала необходима для управления, например, яркостью светодиода.

- pin – ПИН, на котором необходимо генерировать ШИМ
- duty – величина 0.. 255, соответствует скважности ШИМ 0.. 100%

ШИМ пины Arduino NANO, UNO: 3, 5, 6, 9, 10, 11

ШИМ пины Arduino MEGA: все до 13

Частота ШИМ по умолчанию равна 490 Гц на пинах 3, 11, 9, 10 и 980 Гц на пинах 5 и 6. Для увеличения частоты ШИМ используются прямые команды микроконтроллеру.

3.9. Генератор случайных чисел

randomSeed(value) - функция, задающая начало отсчёта генератору псевдослучайных чисел.

- value – любое число типа long.

Для получения максимально случайной последовательности чисел необходимо в качестве опорного числа взять сигнал с неподключенного аналогового пина, например A0:

randomSeed(analogRead(A0));

random(min, max) - функция, возвращающая случайное число в диапазоне от min до (max – 1)

random(max) - то же самое, но возвращает от 0 до (max – 1)

3.10. Аппаратные прерывания

attachInterrupt(pin, function, state) - подключить прерывание

detachInterrupt(pin) - отключить прерывание

- pin – пин прерывания, для NANO и UNO это пины D2 и D3, соответствуют номерам 0 и 1
- function – название функции, которая будет вызвана при срабатывании прерывания
- state – режим обработки, их несколько:

LOW - вызывает прерывание, когда на порту LOW

CHANGE - прерывание вызывается при смене значения на порту, с LOW на HIGH и наоборот

RISING - прерывание вызывается только при смене значения на порту с LOW на HIGH

FALLING - прерывание вызывается только при смене значения на порту с HIGH на LOW

Пример: Кнопочная клавиатура подключена к D2 и GND. Для обработки нажатой клавиши прерывание подключено на ПИН D2 и обрабатывается при нажатии клавиши с помощью вызова функции myInterrupt

```
void setup() {
  pinMode(2, INPUT_PULLUP); // пин D2 подтянут к питанию
  attachInterrupt(0, myInterrupt, FALLING);
}
void myInterrupt() { // функция обработчика прерываний для обнаружения сканкода
}
```

4. БЫСТРЫЙ СТАРТ

“Быстрый старт” – это обычный приём разработчиков современных программных средств. Цель состоит в том, чтобы, не углубляясь пока в подробности, дать новичку или достаточно опытному пользователю первое представление о программном средстве, дать возможность быстро получить конкретный результат. Полное представление, знания и умения появятся позже в процессе работы и изучения справочных материалов.

В качестве примера возьмём простейшую программу, с которой начинают изучение языков программирования многие поколения студентов. “Hello World” - программа, которая выдаёт в последовательный порт (UART) микроконтроллера строку символов “Hello World” (“Привет Мир”).

4.1. Запуск Arduino и создание файла проекта

IDE Arduino запускается из стартового меню Windows подобно остальным приложениям (Рисунок 13).

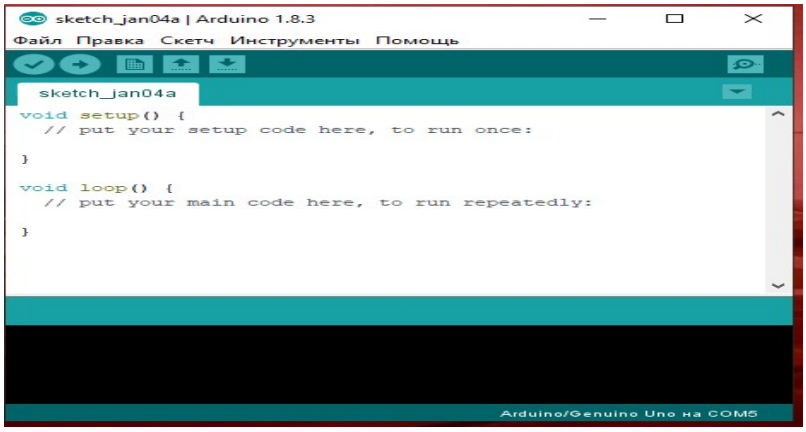


Рисунок 13 – Запуск программы.

В среде создаем простую программу выдающую в последовательный порт (UART) микроконтроллера строку символов “Hello World” при загрузке микроконтроллера, а в жизненном цикле микроконтроллера осуществляющая управление встроенным светодиодом на плате (включающая и выключающая светодиод с частотой 1 Гц) и выдающая по последовательному порту изменение значения счетчика секунд.

Вводим текст программы 1 приведенного в приложении. Производим сохранение проекта, выбрав меню Файл->Сохранить как..

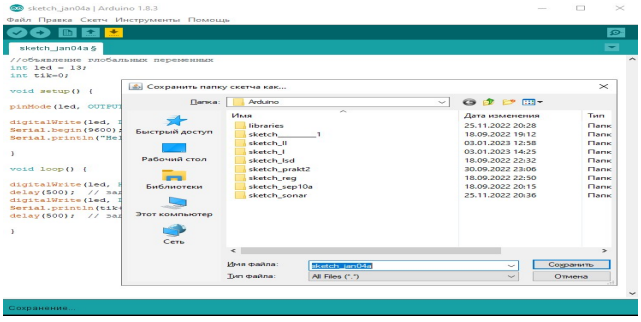


Рисунок 14 – Сохранеие проекта.

4.2. Компиляция и проверка

Для проведения проверки / компиляции проекта необходимо:

в меню Инструменты в установить Плата: «Arduino/Genuino Uno», Порт: «COM… Arduino/Genuino Uno»

2) нажать на кнопку «галочка» :

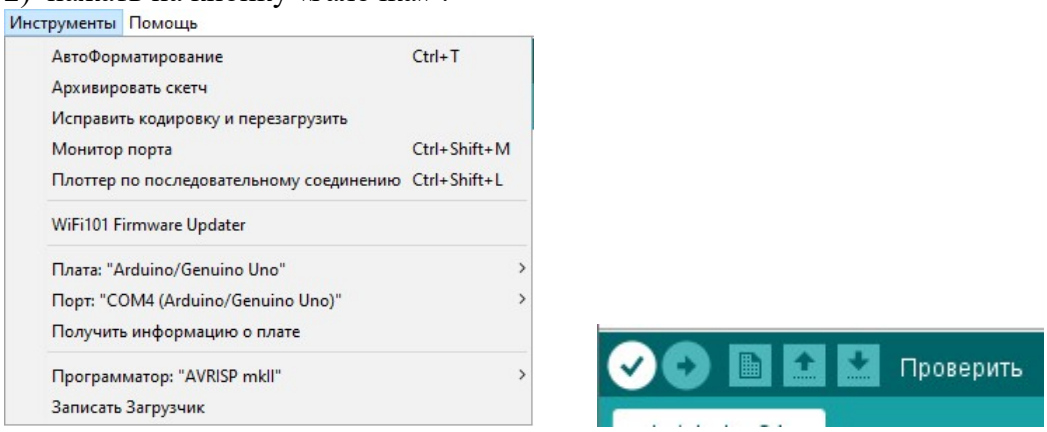


Рисунок 15 – Компиляция проекта.

В результате будет проведена проверка проекта и выданы данные о результатах компиляции

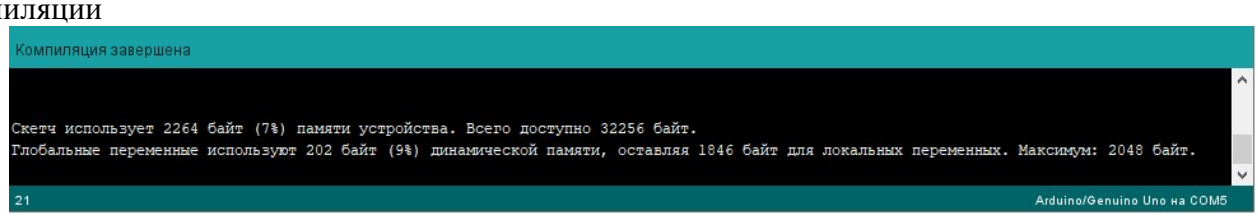


Рисунок 16 – Результаты компиляции проекта.

Проверка проекта проведена успешно, если память микроконтроллера распределена успешно и ошибки не выявлены.

4.3. Загрузка проекта

Для загрузки проекта через программатор необходимо нажать на кнопку «Стрелочка»

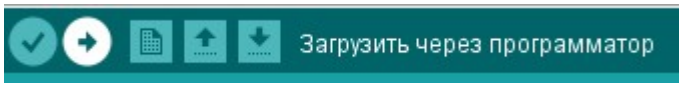


Рисунок 17 – Загрузка проекта в микроконтроллер.

В результате проект будет загружен в микроконтроллер и выведена информация по утилизации памяти, а светодиод на отладочной плате начнет мигать с частотой 1 Гц.

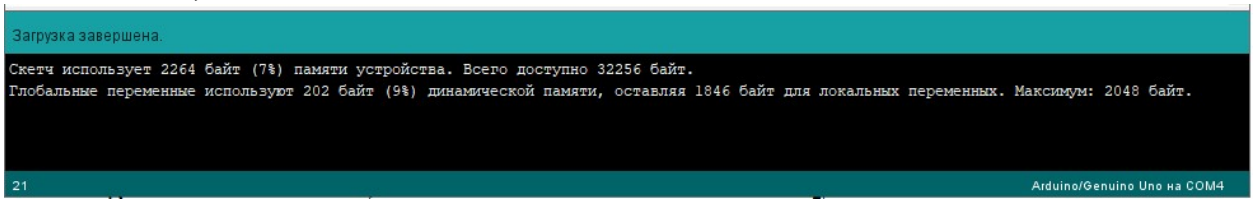


Рисунок 18 – Результаты загрузки проекта.

4.4. Отладка по последовательному порту и вывод графиков

Просмотр результатов работы программы “Hello World” может быть осуществлен с помощью двух инструментов: «Монитор порта» и «Плоттер по последовательному соединению», при этом скорость обмена должна соответствовать 9600бод.

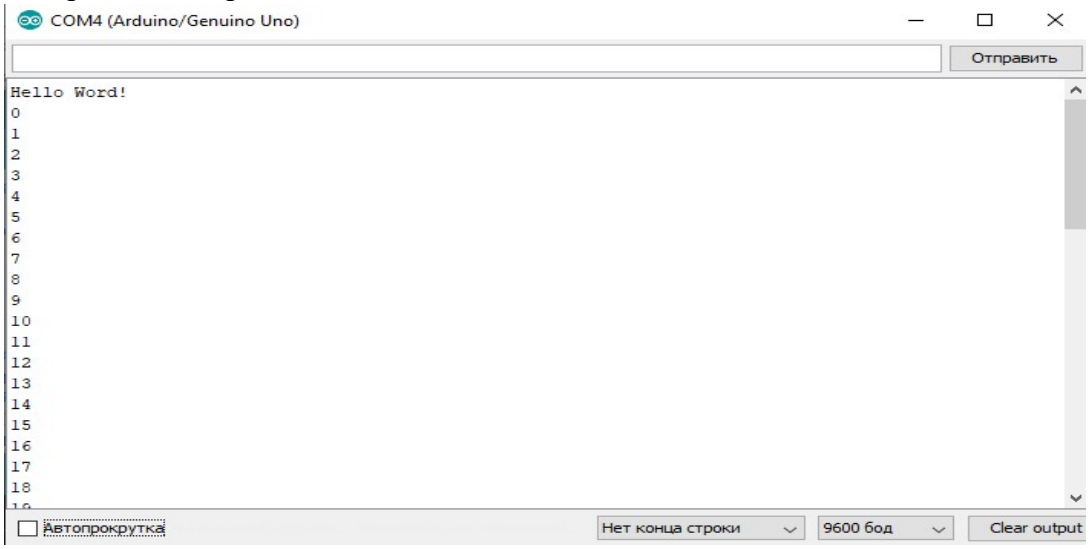


Рисунок 19 – Монитор порта с результвтами работы программы “Hello World”.

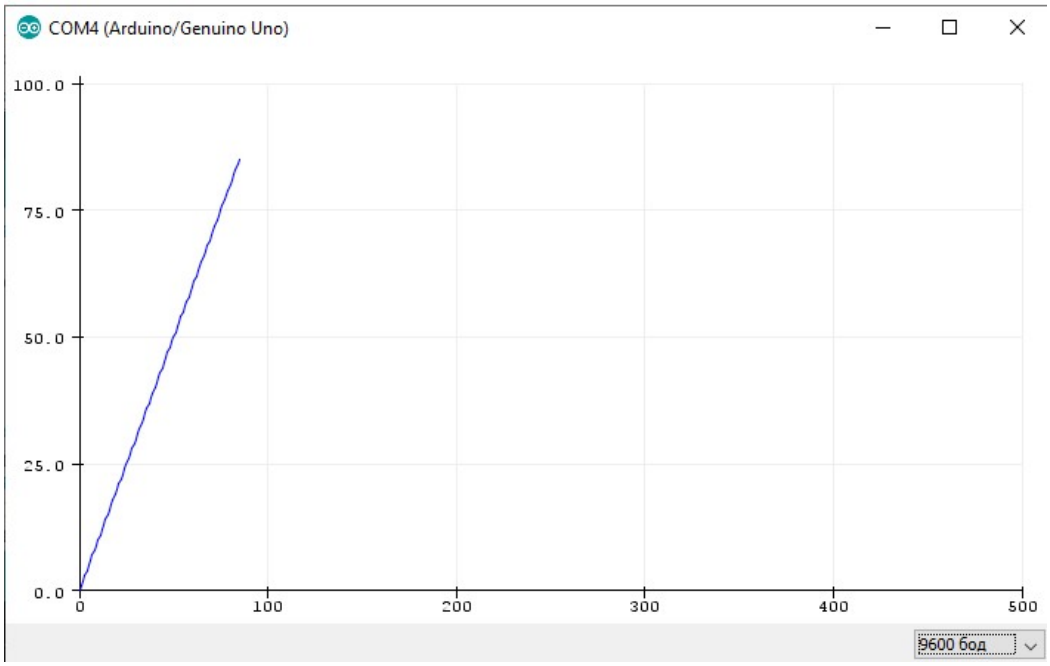


Рисунок 20 – Плоттер по последовательному соединению с результвтами работы программы “Hello World”.

Результат работы программы представлен на рисунках (Рисунок 19 и Рисунок 20).

5. ДОКУМЕНТИРОВАНИЕ РАЗРАБОТКИ НА СХЕМАХ

Схема — *это абстракция какого-либо процесса или системы, наглядно отображающая наиболее значимые части*. Схемы широко применяются с древних времен до настоящего времени — чертежи древних пирамид, карты земель, схемы функционирования

приборов, схемы бизнес-процессов. Очевидно, древние мореплаватели хотели обмениваться картами и поэтому выработали единую систему обозначений и правил их выполнения. Аналогичные соглашения выработаны для изображения электрических схем и схем-алгоритмов, которые закреплены ГОСТ и международными стандартами.

5.1. Документирование электрических схем

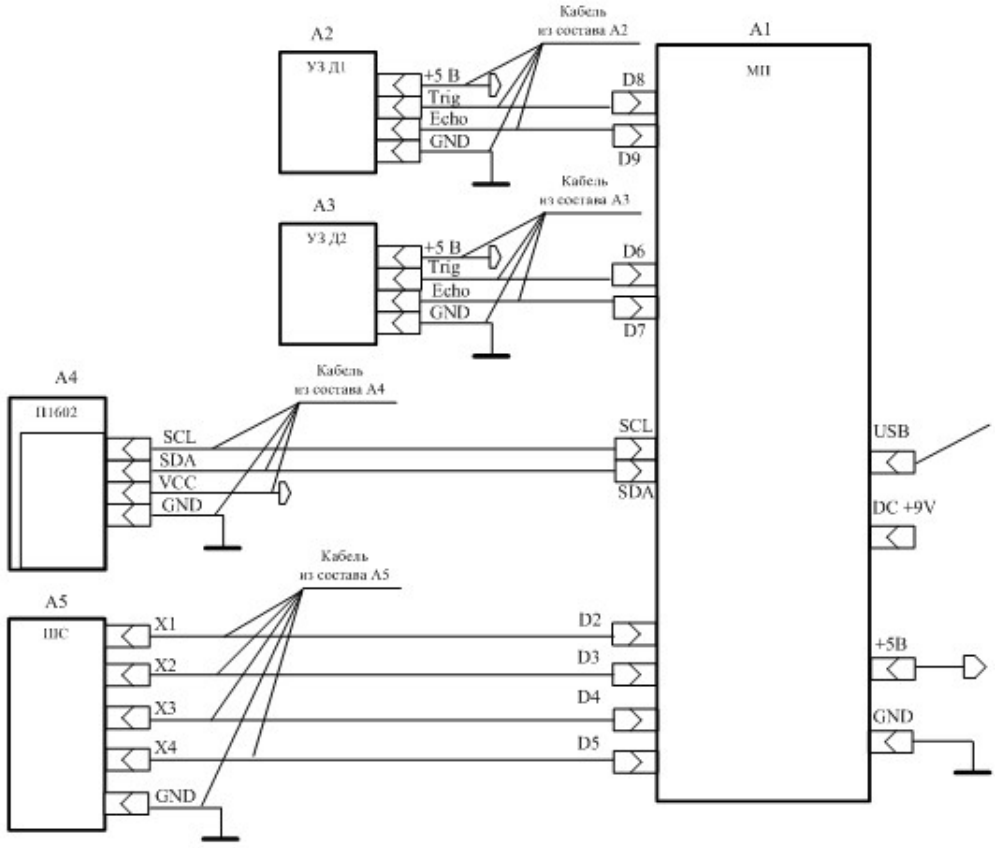
Основные правила выполнения вручную или автоматизированным способом электрических схем изделий всех отраслей промышленности и энергетических сооружений содержит ГОСТ 2.702-75. Полный перечень типов схем приведены ниже.

Тип схемы	Определение	Код типа схемы
Схема электрическая структурная	Документ, определяющий основные функциональные части изделия, их назначение и взаимосвязи	Э1
Схема электрическая функциональная	Документ, разъясняющий процессы, протекающие в отдельных функциональных цепях изделия (установки) или изделия (установки) в целом	Э2
Схема электрическая принципиальная (полная)	Документ, определяющий полный состав элементов и взаимосвязи между ними и, как правило, дающий полное (детальное) представления о принципах работы изделия (установки)	Э3
Схема электрическая соединений (монтажная)	Документ, показывающий соединения составных частей изделия (установки) и определяющий провода, жгуты, кабели или трубопроводы, которыми осуществляются эти соединения, а также места их присоединений и ввода (разъемы, платы, зажимы и т.п.)	Э4
Схема электрическая подключения	Документ, показывающий внешние подключения изделия	Э5
Схема общая	Документ, определяющий составные части комплекса и соединения их между собой на месте эксплуатации	Э6
Схема электрическая расположения	Документ, определяющий относительное расположение составных частей изделия (установки), а при необходимости, также жгутов (проводов, кабелей), трубопроводов, световодов и т.п.	Э7
Схема электрическая объединенная	Документ, содержащий элементы различных типов схем одного вида	Э0
Примечание — Наименования типов схем, указанные в скобках, устанавливают для электрических схем энергетических сооружений.		

Для целей разработки и макетирования схем автоматизации вычислений, сбора данных измерений от датчиков, сопряжения с объектом автоматизации могут использоваться любые из приведенных схем, однако в большинстве случаев потребуется несколько схем, что создаст избыточность и сложность. Поэтому лучше всего для макетирования создать документ, содержащий элементы различных типов схем одного вида, а значит это схема

Э0.

Пример схемы Э0 приведен ниже.



Сокращения:
МП – макетная плата;
УЗ Д – ультразвуковой датчик;
П1602 – панель 1602;
ШС – шина сигнала;
ИУЗ – измерение уровня заправки;
РКН – ракета космического назначения.

Таблица 1

Поз. обозна-чение	Наименование	Кол.
A1	Макетная плата NANO (контроллер ATmega320)	1
A2	Ультразвуковой датчик HC-SR04 (измерительный)	1
A3	Ультразвуковой датчик HC-SR04 (калибровочный)	1
A4	Панель 1602	1
A5	Шина управляющего сигнала	1

Рисунок 21 – Пример схемы электрической объединенной.

Для документирования результатов практических работ будем использовать правила разработки схем электрических объединенных.

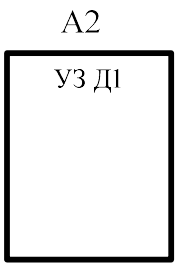
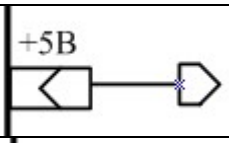
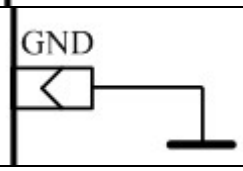
Основными частями схемы являются:

- Элемент схемы (устройство, прибор);

- Проводник (провод, жгут, кабель);
- Разъем элемента схемы (для макетных элементов схемы, т.к. все соединения беспаечный);
- Разъем проводника;
- Электропитание и заземление.

На схеме эти элементы имеют следующие обозначения:

Таблица – обозначения на объединенных схемах.

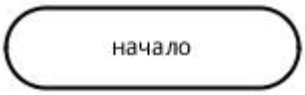
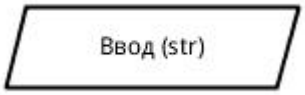
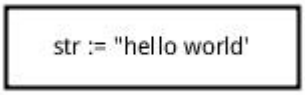
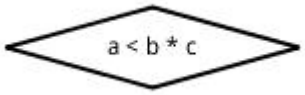
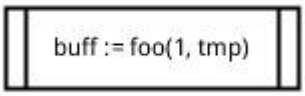
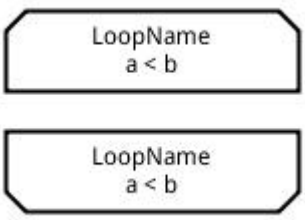
Наименование	Обозначение	Примечание
Элемент схемы		Элемент схемы обозначается в виде квадрата, внутри квадрата вносятся условные обозначения устройства или прибора. Сверху вносится позиционное обозначение элемента и в таблице 1 раскрывается полное наименование элемента и общее количество элементов.
Разъем элемента схемы		Каждый элемент схемы имеет не менее одного разъема. Не используемые разъемы не вносятся на схему, а используемые обязательно обозначаются латинскими буквами и цифрами. Обозначения разъемов нужно брать с прибора.
Разъем проводника		
Проводник		
Электропитание		
Заземление		

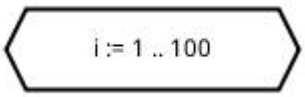
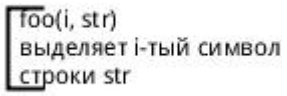
5.2. Документирование алгоритмов для микроконтроллера

На территории Российской Федерации действует *единая система программной документации (ЕСПД)*, частью которой является Государственный стандарт — **ГОСТ 19.701-90 «Схемы алгоритмов программ, данных и систем»** [4].

Ниже в таблице приведены основные символы, принятые для описания алгоритмов программ, в том числе и для микроконтроллеров. Обязательным требованием для всех схем алгоритмов – это одинаковая ширина символов.

Таблица - обозначения символов на схеме алгоритмов.

Символ	Описание применения
 Терминатор начала и конца работы алгоритма	С этого специального символа начинается и заканчивается любая функция. Тип возвращаемого значения и аргументов функции обычно указывается в комментариях к блоку терминатора.
 Операции ввода и вывода данных	Обобщенный символ ввода/вывода. Подробности ввода/вывода указываются в комментариях.
 Выполнение операций над данными	В блоке операций обычно размещают одно или несколько (ГОСТ не запрещает) операций присваивания, не требующих вызова внешних функций.
 Блок, иллюстрирующий ветвление алгоритма	Блок в виде ромба имеет один вход и несколько подписанных выходов: 1) В случае, если блок имеет 2 выхода (соответствует оператору ветвления), на них подписывается результат сравнения — «да/нет». 2) Если из блока выходит большее число линий (оператор выбора), внутри него записывается имя переменной (switch), а на выходящих дугах — значения этой переменной (case).
 Вызов внешней процедуры	Вызов внешних процедур и функций помещается в прямоугольник с дополнительными вертикальными линиями.
 Начало и конец цикла	Символы начала и конца цикла содержат имя и условие. Условие может отсутствовать в одном из символов пары. Расположение условия, определяет тип оператора, соответствующего символам на языке высокого уровня — оператор с предусловием (while) или постусловием (do … while).

Символ	Описание применения
 Подготовка данных	Символ «подготовка данных» в произвольной форме (в ГОСТ нет ни пояснений, ни примеров), задает входные значения. Используется обычно для задания циклов со счетчиком.
 Соединитель	В случае, если блок-схема не умещается на лист, используется символ соединителя, отражающий переход потока управления между листами. Символ может использоваться и на одном листе, если по каким-либо причинам тянуть линию не удобно.
 Комментарий	Комментарий может быть соединен как с одним блоком, так и группой. Группа блоков выделяется на схеме пунктирной линией.

Для использования этих знаков в MS Visio необходимо активировать панель инструментов «Flowchart». Единственный элемент, который отсутствует в «Flowchart» - это конец цикла, но он легко получается путем вращения на 180 элемента «Loop limit».

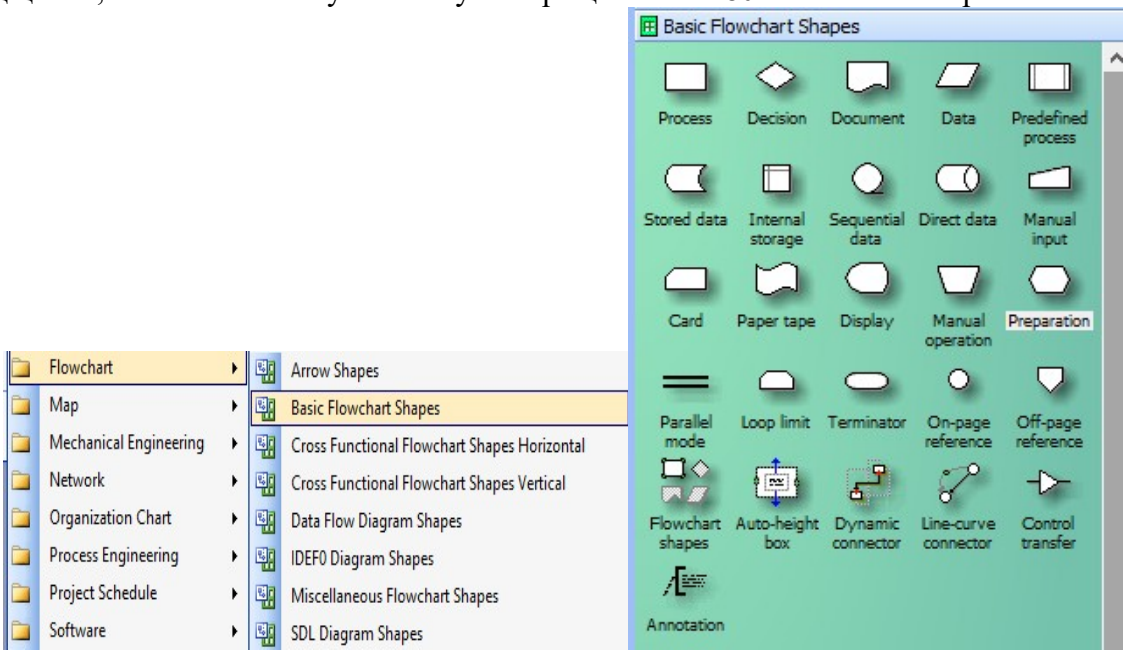


Рисунок 22 – инструменты MS Visio для создания блок-схем алгоритмов.

В качестве примеров блок-схем можно привести шаблоны алгоритмов для функций ветвления.

1) **if (condition) {operatorsT} else { operatorsF}**

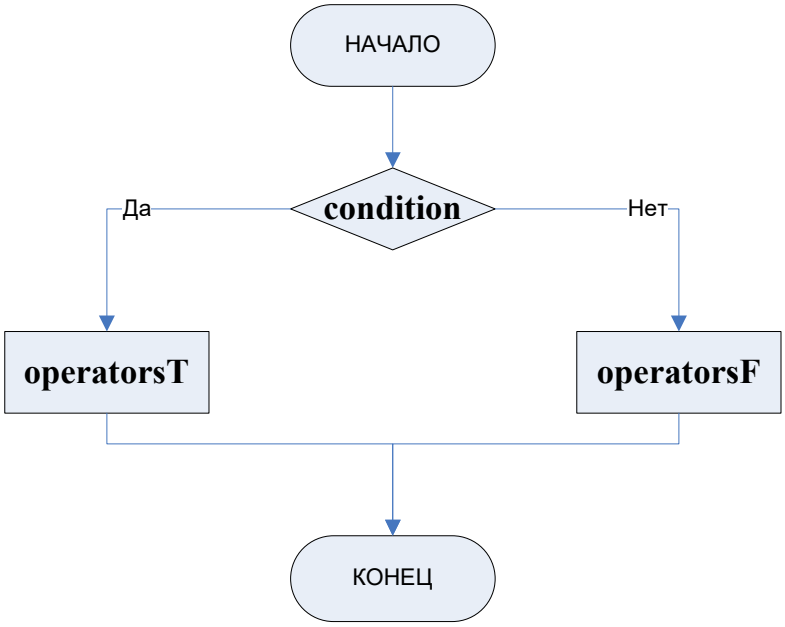


Рисунок 23 – блок-схема алгоритма условного оператора «если».

2) **for (counter; condition; change) { operators }**

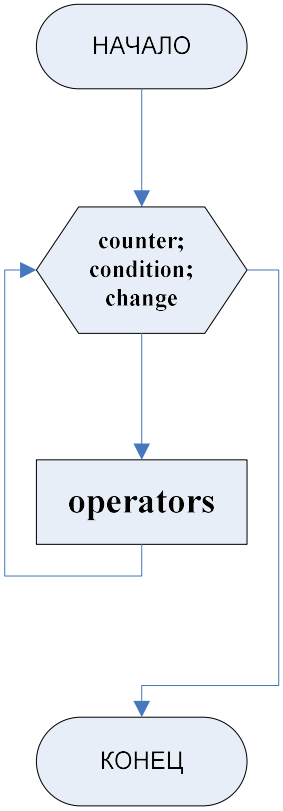


Рисунок 24 – блок-схема алгоритма цикла «счетчик».

3) **while (condition) { operatorsT } // цикл A**

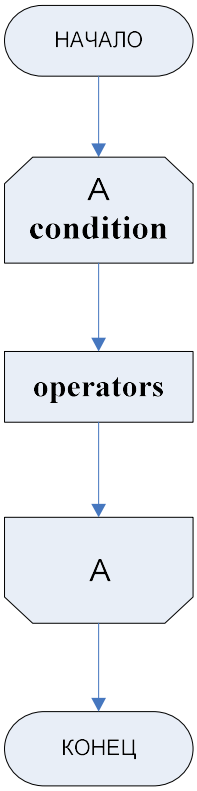


Рисунок 25 – блок-схема алгоритма цикла с предусловием.

4) **do { operatorsT } while (condition) // цикл B**

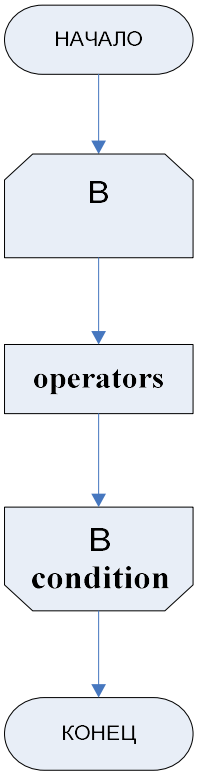


Рисунок 26 – блок-схема алгоритма цикла с постусловием.

```
5) switch (val)
{
   case 1: operators1;
           break;
   case 2: operators2;
           break;
…
   case n: operatorsN;
           break;
   default: operators;
           break;
}
```

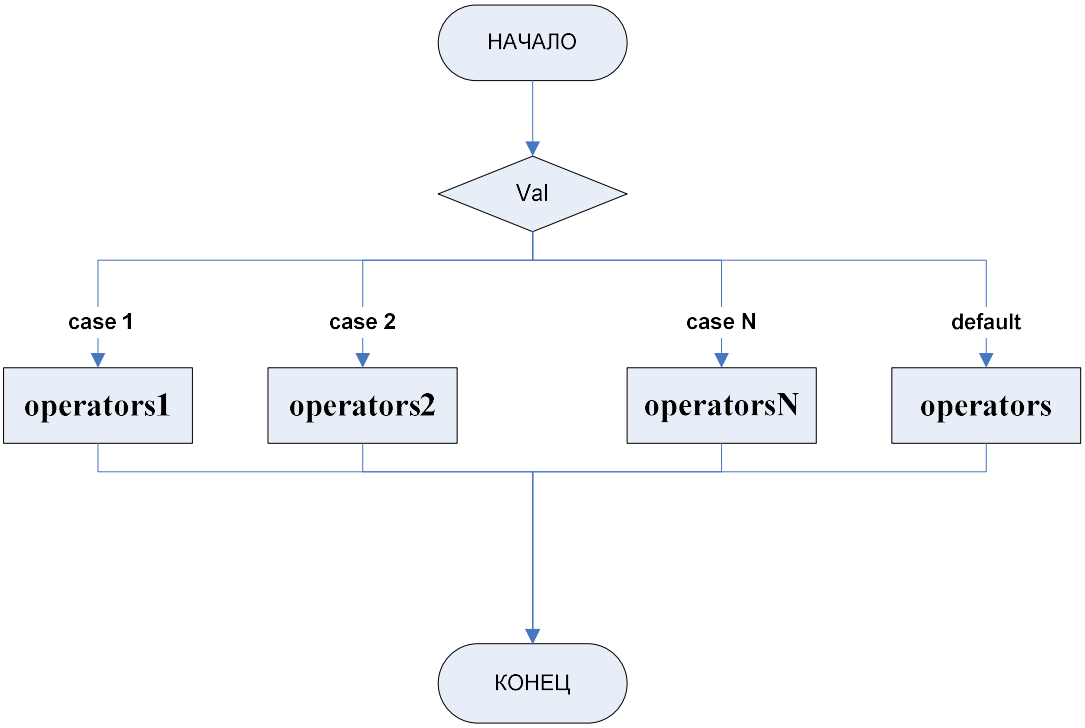


Рисунок 27 – блок-схема алгоритма условного оператора «выбор».

5.3. Элементная база практических работ

Для проведения практических работ используется наиболее распространенные современные устройства ввода и вывода информации. На практических работах будут изучены основные интерфейсы и протоколы обмена данными внешних устройств с микроконтроллером.

5.3.1. Матричная клавиатура

Благодаря небольшой толщине, пленочные клавиатуры легко и быстро размещаются на плоских поверхностях, достаточно просто приклеить их на двухсторонний скотч. Частично герметичное исполнение защищает от брызг жидкостей, а поверхность легко отмывается от внешних загрязнений. Ресурс относительно небольшой, но компенсируется быстрой заменой без удара по бюджету.

Кнопка - простейшее устройство передачи информации от человека контроллеру. С ее помощью мы можем осмысленно установить на конкретном пине ноль или единицу, тем самым сообщив программе команду на то или иное действие. Иногда для проекта хва-

тает всего одной кнопки, чаще двух-трех, но бывают случаи, когда кнопок надо много, например для ввода пароля из цифр или букв, или комбинации того и другого. Или мы создаем прибор, требующий большого количества настроек, ввода параметров, набора порядка действий и так далее. Для таких задач мало и дюжины кнопок, а ведь каждой требуется отдельный пин, которых на микроконтроллере не так уж много. Когда мы понимаем, что для одной отдельной кнопки слишком расточительно выделять аж целый пин, нам на помощь приходят иные способы организации массива этих самых кнопок. Один из способов экономии ПИНов является “матрицирование”, а клавиатуры, собранные под него - “матричными”. Устроено все просто, несколько кнопок соединяются в прямоугольную матрицу, располагаясь в узлах рядов и столбцов из проводников.

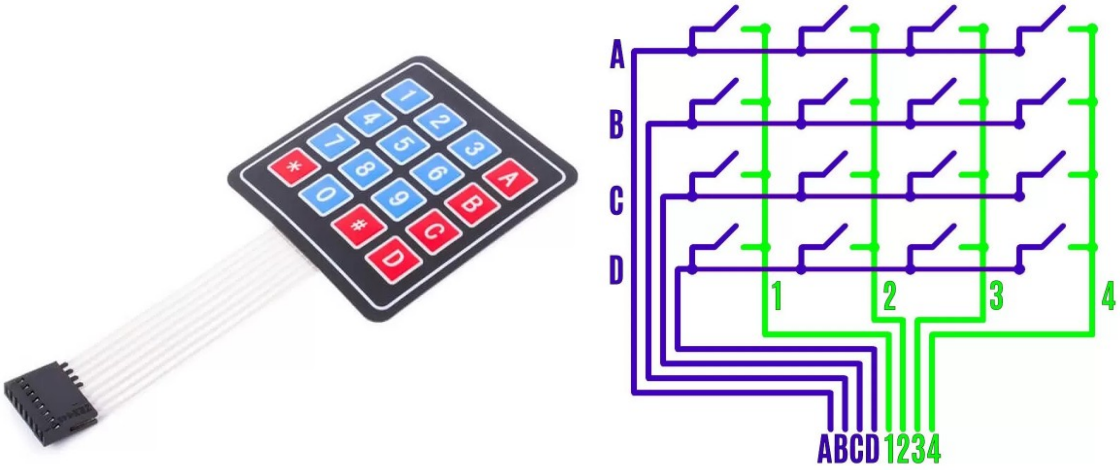


Рисунок 28 – матричная клавиатура.

Каждая кнопка, при нажатии на нее, замыкает контакты между конкретным столбцом и конкретным рядом, создавая цепь, которую можно программно обнаружить. Например, если мы нажмем верхнюю левую кнопку на схеме, изображенной выше, мы замкнем контакты A и 1. И так, как легко убедиться, любой кнопке соответствует своя индивидуальная пара контактов. При этом контактов всего 8, а кнопок аж целых 16! Уже заметна неплохая экономия в 8 пинов. Нетрудно подсчитать, что добавляя ряды и/или столбцы, мы будем еще больше увеличивать отношение между кнопками и пинами. Так, матрица 5x5 сэкономит нам уже 15 ножек контроллера.

Однако не все так просто. Мы не можем подать одинаковый сигнал на ряды и считывать их на столбцах, в этом случае мы могли бы определять только вертикаль, в которой нажата кнопка. Но если мы подадим сигнал, к примеру, только на ряд A, то все кнопки верхнего ряда будут отслеживаться на контактах 1, 2, 3 и 4, кнопки же в остальных рядах в этот момент никакого влияния на результат не окажут, ни нажатые, ни отжатые. Если подадим на ряд B, то отследим все кнопки только второго сверху ряда и так далее. Таким образом, для того чтобы составить общую картину, нам придется пройтись по всем рядам и всем столбцам, причем не важно в каком порядке, это дело вкуса, главное, чтобы сделать это как можно быстрее, пока с кнопки не убрали палец, и ни одной не пропустить. Этот полезный способ опроса называется “динамическим сканированием”.

В практической работе № 3 будет исследована наиболее популярная клавиатура, применяемая в простых устройствах сопряжения с объектом автоматизации и сбора информации.

5.3.2. Интерфейсы UART, SPI и I2C

UART - Универсальный асинхронный приёмопередатчик (Univsersal Asynchronos Reciever-Transmitter) - это физическое устройство приёма и передачи данных по двум проводам. Оно позволяет двум устройствам обмениваться данными на различных скоростях. В спецификацию UART не входят аналоговые уровни на которых ведётся общение между устройствами, UART это протокол передачи бит данных (протокол логического уровня), электрическую спецификацию на себя берут другие стандарты (протоколы физического уровня), такие как TTL (transistor-transistor logic — транзисторно-транзисторная логика), RS-232, RS-422, RS-485 и другие (RS [англ.recommended standard] — рекомендованный стандарт). На данный момент в микроконтроллерах используется в основном TTL (или точнее CMOS) UART для соединения не более двух устройств. Для передачи данных на значительное расстояние используются в основном RS протоколы. В данном пособии UART называем последовательным портом и работаем через библиотеку *Serial*.



Рисунок 29 – UART интерфейс.

У каждого устройства, поддерживающего UART обычно обозначены два вывода: RX и TX. TX — означает transmit (передаю), RX — receive (принимаю). Отсюда становится понятно, что RX одного устройства нужно подключать к TX другого.

В скетче UART инициализируется функцией begin() в коде процедуры setup(): *«Serial.begin(скорость передачи);»*.

Если подключить RX одного устройства к RX другого, то оба устройства будут слушать друг друга, вы соединили их входы (естественно ничего не услышат). Если соединить TX и TX - это уже более опасно, это выходы низкого сопротивления устройств и если на одном будет логическая единица, а на втором ноль — по проводу пойдёт ток короткого замыкания (это зависит от конкретной программной или аппаратной реализации). Хотя в современных чипах от этого есть защита, на всякий случай, не стоит на неё ориентироваться. Так же необходимо объединить референсные уровни двух устройств (**GND-GND**), если не подразумевается гальваническая развязка.

Рассмотрим передачу данных с использованием UART.

При отсутствии передачи линия удерживается в состоянии логической единицы (в случае TTL это 5 вольт или “Vcc”). Как только передающее устройство притягивает линию к 0 (GND или 0 вольт), это сигнализирует принимающему устройству о том, что сейчас будет передача данных. Этот сигнал обычно называют «**старт-битом**»

При появлении старт-бита на линии принимающее устройство начинает отсчитывать такты (время) в соответствии с установленной скоростью и считывать состояния линии через определённые промежутки времени в соответствии с установленным количеством бит данных. Обычно, (в том числе и в Atmel) установленным количеством бит данных является **байт** данных (8 бит).

По завершении передачи данных принимающее устройство ожидает «**стоп-бит**», который должен быть на уровне логической единицы. Если по завершении передачи байта данных удерживается логический ноль, значит, переданные данные неверны. Если логический ноль удерживается время, превышающее длину байта в 1,5 раза, такое состояние именуется break (разрыв линии, исторически пошло от устройств, использующих токовую

петлю для передачи данных). Некоторые передатчики вызывают это состояния специально перед посылкой пакета данных. Некоторые приёмники считают такое состояние за неправильно выставленную скорость и сбрасывают свои настройки на установки "по умолчанию".

Основной характеристикой передачи данных с использованием UART – это скорость передачи данных или скорость изменения логических уровней. Скорость изменения логических уровней (импульсов) на линии принято измерять в бодах. Единица измерения названа так в честь французского изобретателя Жана Мориса Эмиля Бодо.

Скорость при использовании UART может быть любой, единственное требование — скорости передающего и принимающего должны быть одинаковы. Стандартная скорость UART принята за 9600 бод. Микроконтроллеры Atmel без проблем и лишних настроек могут принимать и передавать данные на скоростях до 115200 бод.

Так как при передаче данных присутствуют синхронизирующие биты, именуемые старт-бит и стоп-бит, не совсем корректно говорить, что скорость 9600 бод равна 9600 битам в секунду. Если речь идёт о полезных данных, то реальная скорость на 20% ниже. Например, если выставлены параметры 8-N-1 и 9600 бод, то на передачу одного байта уходит десять бит, и 9600/10 = 960 байт, что равно 7680 битам в секунду.

Дополнительно можно указать, что прием данных по UART можно осуществлять не только в среде Arduino, но и с использованием утилиты **PuTTY**. PuTTY - эмулятор терминала, который в прошлом был единственным способом работы с мейнфреймными компьютерами. Для работы с отладочной платой через терминал в утилите необходимо выбрать COM – порт , скорость передачи данных (указанной при инициализации протокола командой begin).

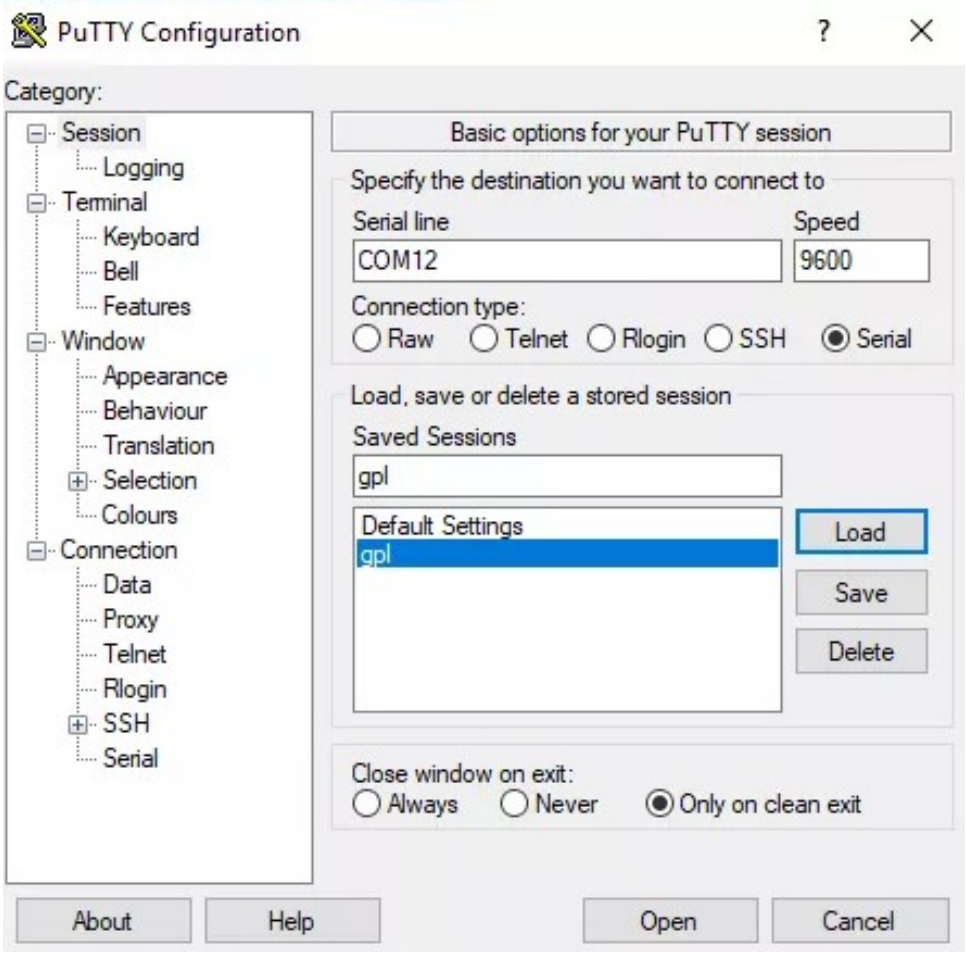


Рисунок 30 – конфигурация программы PuTTY.

С левой стороны выберите "Logging" (Ведение журнала), затем выберите "Printable output" (Вывод на печать) и "Always" (Всегда). Нажмите кнопку "Browse" (Обзор) и найдите место и имя файла, куда вы можете сохранить файл, который вы записали в журнал. Лучше всего выбрать расширение имени файла **.csv** (например, '**logging.csv**'), чтобы MS Excel мог его сразу распознать.

В практических работах № 1 и 2 будут исследованы наиболее важные функции работы по UART, применяемые в простых устройствах сопряжения с объектом автоматизации и сбора информации.

I²C (IIC), (Inter-Integrated Circuit) — последовательная асимметричная шина для связи между интегральными схемами внутри электронных приборов.

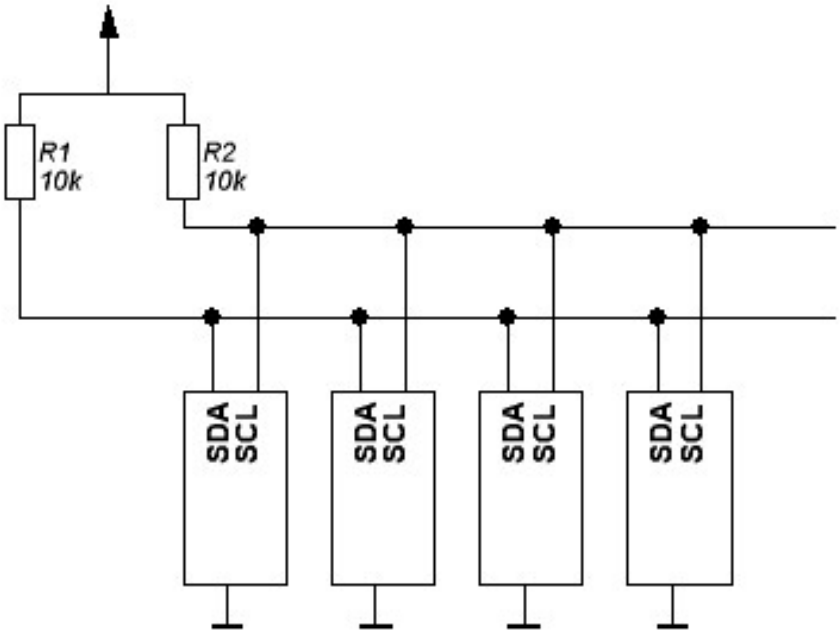


Рисунок 31 – **I²C** интерфейс.

В отличие от UART, i2c является протоколом синхронной связи, это означает, что обмен данными происходит по общему для всех связанных устройств сигналу синхронизации. Генерацией сигнала занимается только единое главное устройство, оно же Ведущее (Master). Устройство “говорит” и “дает слово” остальным, которые называются Ведомыми (Slave). Все ведомые устройства имеют уникальный номер, даже если такое устройство на всю сеть одно. Мастер номера не имеет, его и так все знают. Ведомые молчат, слушают, что говорит ведущий и откликаются только тогда, когда ведущий их об этом просит, называя их по имени (номеру), поэтому в сети всегда царит образцовый порядок.

Обращение мастера начинается с падения уровня на шине данных SDA, что является стартовым сигналом для ведомых. Повышение уровня SDA при высоком тактовой сигнале является для них стоп-командой. Все, что происходит между этими событиями, называется “сообщение”, то есть передача данных.



Рисунок 32 – сигналы последовательной синхроггой переодачи.

Передача/Прием сигналов осуществляется прижиманием линии в 0, в единичку устанавливается сама, за счет подтягивающих резисторов. Эти резисторы в схемах I2C устанавливать необходимо всегда, стандартом установлено сопротивление 10к Ом. Чем больше резистор, тем дольше линия восстанавливается в единицу (идет перезаряд паразитной емкости между проводами) и тем сильней заваливаются фронты импульсов, а значит скорость передачи падает. Именно поэтому у I2C скорость передачи намного ниже чем у SPI. Обычно IIC работает либо на скорости 10кбит/с — в медленном режиме, либо на 100кбит/с в быстром. Но в реальности можно плавно менять скорость вплоть до нуля. Ни в коем случае нельзя переключать вывод микроконтроллера в OUT и дергать ногу на +5В, так как можно получить короткое замыкание и «сжечь» либо контроллер, либо какое-нибудь устройство на шине.

Вся передача данных состоит из «стартового бита», «сообщения» и «стоп-бита». Порядок изменения уровня на шинах задает тип посылки. При этом сообщение намного сложнее чем сообщение UART:



Рисунок 33 – формат сообщения I2C .

Разберем сообщение:

Кадр адреса - в зависимости от разновидности протокола, тот самый идентификатор конкретного ведомого устройства, к которому обращается ведущий.

Read/write bit (бит чтения или записи) - указывает что мы хотим делать - передать данные или получить их.

ACK/NACK - биты подтверждения или не подтверждения приема информации каждого кадра. ACK единственный бит, который генерируется в линии SCL ведомым устройством. К сожалению, без дополнительных мер мы не можем воспользоваться этой информацией о доставке, она лишь сокращает время передачи, обрывая ее, когда принимающее устройство перестает отвечать.

В практической работе № 6 будут исследованы наиболее важные функции работы по i2c, применяемый в интерфейсе жидкокристаллического дисплей 1602.

Выводы, отвечающие за интерфейс I2C на платах Arduino на базе различных контроллеров разнятся:

Arduino Mega	*Arduino Uno/Nano/Pro Mini*	I2C модуль
GND	GND	GND
5V	5V	VCC
20 (SDA)	A4	SDA
21 (SCL)	A5	SCL

SPI — последовательный синхронный стандарт передачи данных в режиме полного дуплекса, предназначенный для обеспечения простого и недорогого высокоскоростного сопряжения

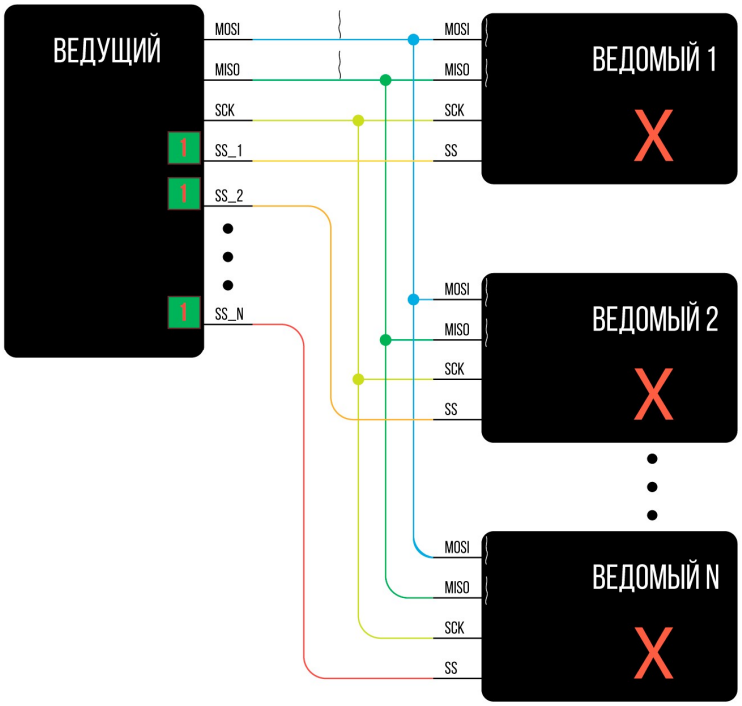


Рисунок 34 – **SPI** интерфейс.

Название интерфейс SPI является аббревиатурой от “Serial Peripheral Bus”, что можно перевести как “шина для подключения периферийных устройств”. Отсюда вытекает ее главное назначение - связать одно главное устройство - Ведущее (Master) - с одним или несколькими Ведомыми (Slave). Ведущий в этом интерфейсе всегда один, только он руководит всем процессом и только он может формировать тактовые импульсы. Если в нашем случае Ведущим всегда является микроконтроллер (эту роль может исполнять и компьютер, но это другая история), то Ведомыми в подавляющем большинстве случаев являются именно периферические устройства. Датчики, дисплеи, микросхемы ЦАП и АЦП, RFID-ридеры, модули беспроводной связи, включая приемо-передатчики WiFi и Bluetooth, GPRS-адаптеры и так далее. Для связи контроллеров и/или компьютеров обычно не используется, но вообще это возможно. Особенно этот интерфейс востребован там, где требуется высокая скорость передачи данных и не менее высокая надежность. SPI самый быстрый из всех имеющихся в нашем распоряжении и самый “легкий” с точки зрения потребляемых ресурсов. Расплатой за это является использование большего количеств проводов, чем для других интерфейсов. Здесь требуется аж 3 штуки только непосредст-

венно для передачи данных, не зря же его второе имя 3-ware.

В отличие от I2C и UART, SPI требует больше сигналов для работы, но может работать на более высоких скоростях. Скорость может достигать десятков мегагерц, что позволяет передавать большие объемы данных в потоковом режиме. Все шины однонаправлены, это упрощает задачу преобразования уровней. Шина не поддерживает адресацию, вместо этого используется сигнал SS (slave select - выбор ведомого), который также иногда называется CS (chip select), CE (chip enable) или SE (slave enable). Поддерживает только одного ведущего на шине. Ведущий устанавливает скорость обмена данными и другие параметры, такие как полярность и фаза тактирования. Обмен данными происходит в режиме полного дуплекса, что означает устройства на шине могут одновременно передавать и принимать данные. Интерфейс использует следующие сигналы (в номенклатуре AVR, для получения точного названия сигналов обратитесь к технической документации микросхемы, с которой работаете):

MISO (master in slave out) - вход ведущего, выход ведомого

MOSI (master out slave in) - выход ведущего, вход ведомого

SCK (serial clock) - сигнал тактирования

SS (slave select) - сигнал выбор ведомого.

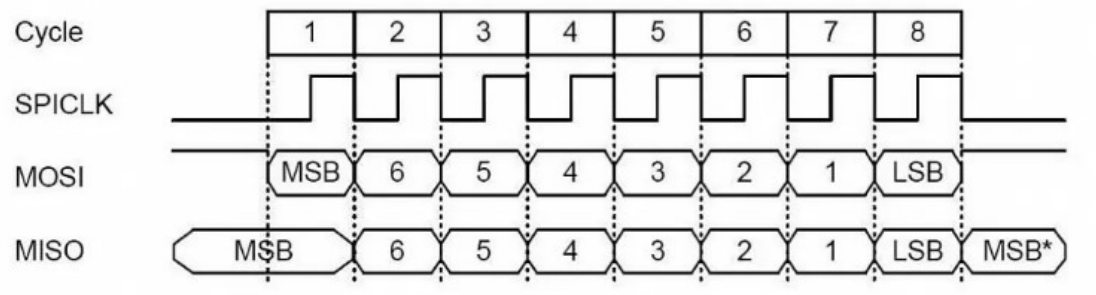


Рисунок 35 – сигналы SPI.

В практической работе № 5 будут исследованы наиболее важные функции работы по SPI, применяемый в микросхеме MAX2719 для графического светодиодного индикатора 8x8.

5.3.3. Светодиодные индикаторы

Светодиодные (полупроводниковые) индикаторы представляют собой твердотельные приборы, работающие на p-n - переходах и в основном применяются для отображения знаковой информации при относительно небольших размерах символа и ограниченном числе знакомест. Одной из отличительных особенностей светодиодных индикаторов является их совместимость по электрическим характеристикам с обычными интегральными микросхемами. При напряжении питания 3.5 В светодиодные индикаторы обладают малой инерционностью (менее 50 нс) и небольшими габаритами. Матричные светодиодные индикаторы являются основой построения панелей.

ветодиодные индикаторы изготовляют в виде семи- или более сегментных, буквенно-цифровых, шкальных и матричных индикаторов. Информационные светодиодные панели больших размеров в основном изготавливают из отдельных светодиодов, расположенных по строкам и столбцам. Необходимый светодиод зажигается при подаче напряжения на нужную колонку и столбец. В информационных панелях большого размера для повышения емкости выводимой информации широко используют «бегущую строку».

Преимущества светодиодных индикаторов как устройств отображения видимой ин-

формации следующие:

- низкое (порядка 3 В) управляющее напряжение;
- простое сопряжение с выходами интегральных микросхем;
- большие углы обзора;
- высокое (до 50 нс) быстродействие;
- высокая механическая прочность;
- возможность считывания информации на больших (до 10 м) расстояниях даже в полной темноте.

К недостаткам светодиодных индикаторов следует отнести:

- невозможность создания информационных панелей больших размеров с малыми размерами пикселя;
- затруднение при считывании информации в условиях прямого падения световых лучей;
- относительно большая потребляемая мощность (от 10 до 50 мВт на светодиод) и большие токи (от 3 до 20 мА на светодиод).

Для отображения буквенно-цифровой информации широкое применение нашли семисегментные светодиодные индикаторы с общим катодом или общим анодом представленные на рисунке 36. Чертеж внешнего вида подключения семисегментного индикатора с общим анодом представлен на рисунке 37

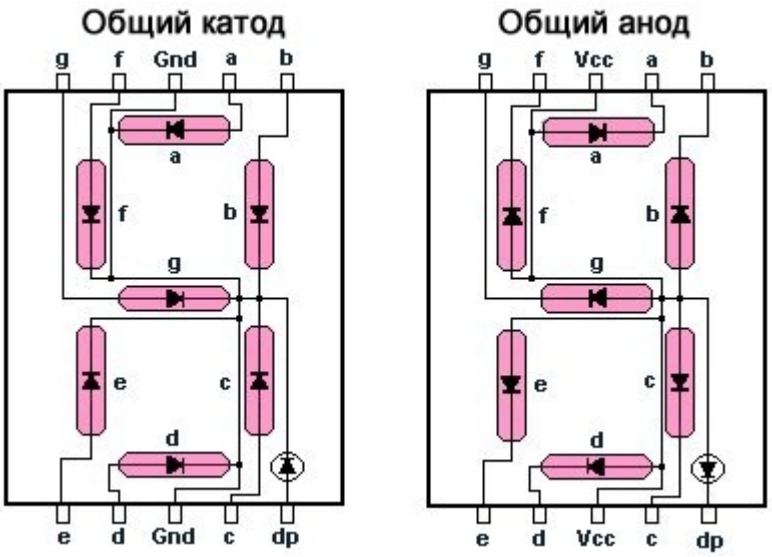


Рисунок 36 – семисегментный индикатор (с общим катодом и общим анодом).

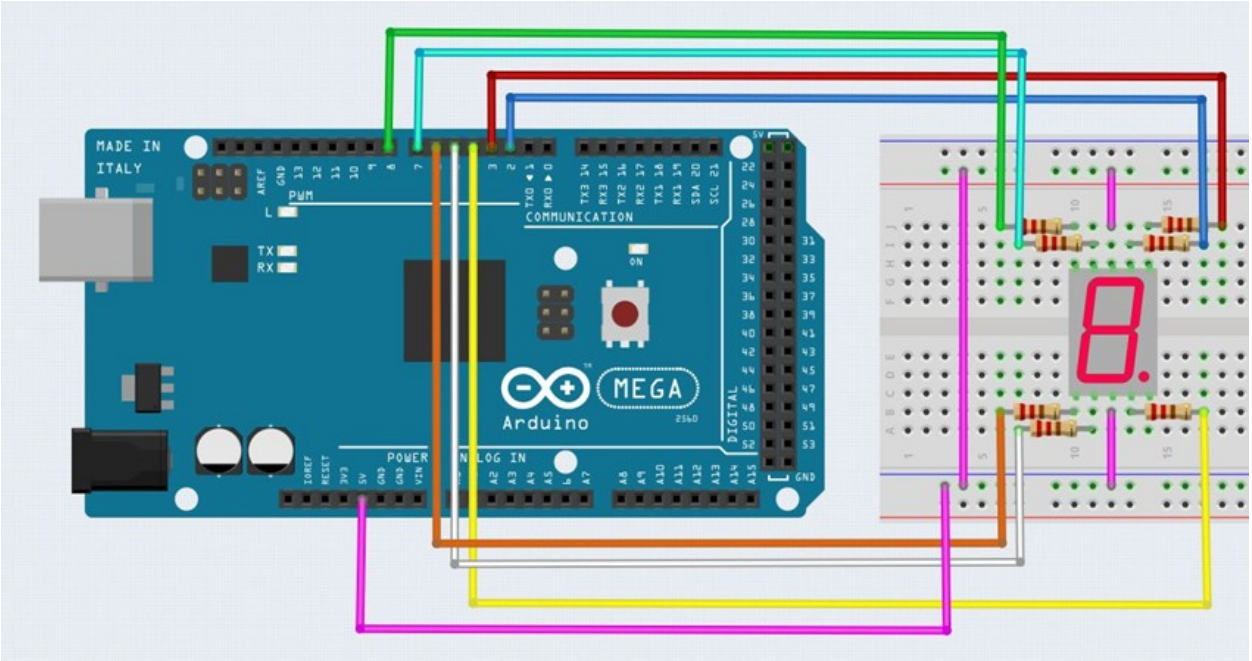


Рисунок 37 – подключение семисегментного индикатора.

Если расположить светодиоды в виде двухмерной матрицы как пиксель на экране монитора, то уже получится графический индикатор. То есть такой, на котором можно отобразить не только число, но и какое-то изображение.

Такая матрица светодиодов называется матричным светодиодным индикатором. Разрешение матричного индикатора — это количество точек по горизонтали и вертикали. Например, самые распространенные индикаторы имеют разрешение 8×8 точек.

Если требуется светодиодная матрица с большим разрешением, то её просто-напросто составляют из нескольких 8×8 индикаторов. Разберем как можно подключить такую матрицу.

Конечно, можно бы было, как и в случае семисегментного индикатора, соединить все светодиоды общим катодом или анодом. В этом случае нам бы потребовалось либо 64 вывода контроллера, либо 8 сдвиговых регистров. Оба варианта весьма расточительны.

Более правильный вариант — объединить светодиоды в группы по 8 штук с общим катодом. Пусть это будут столбцы матрицы. Затем, параллельные светодиоды в этих столбцах объединить снова в группы по 8 штук уже с общим анодом.

Данная схема подключения сокращает количество задействованных пинов контроллера с 64 до жалких 16, что тоже немало, но уже по силам почти всем Ардуино.

Если же количество свободных для матрицы пинов ограничено, или если матриц несколько, или они большого размера (в смысле количества пикселей и цветов), используются различные ухищрения и приспособления. Например, сдвиговые регистры типа 74HC595. У каждого из них по 8 выводов, значит для матрицы 8x8 потребуется таких два, соединенных в каскад, но от контроллера будет достаточно всего три пина. Кроме того, продляя каскад можно добавлять матрицы еще и еще, используя для управления всего лишь те же самые 3 пина. Еще больше облегчить жизнь способна специализированная микросхема MAX2719, специально созданная для одноцветных матриц 8x8, она имеет все 16 выводов и самостоятельно поддерживает динамическую индикацию. Кроме того, MAX2719 тоже способна соединяться каскадами, образуя цепочку матриц при тех же трех пинах на входе. На базе данной микросхемы промышленным способом выпускаются светодиодные модули из одной и более матриц. Под них существуют готовые библиотеки,

что делает вывод изображений и надписей с различными эффектами, например “бегущая строка”, предельно простыми.

MAX7219 – микросхема-драйвер для управления 7-сегментными индикаторами и матрицами 8x8. Благодаря встроенной динамической индикации и настройке тока этот чип в десятки раз упрощает работу и с теми, и с другими.

Управление драйвером осуществляется по интерфейсу SPI, для реализации которого потребуется выделить всего 3 линии ввода/вывода микроконтроллера, максимальная частота тактирования равна 10 МГц. Помимо семисегментных индикаторов, с помощью драйвера можно управлять светодиодной матрицей.

К драйверу можно подключить до 8-ми индикаторов с общим катодом. Выводы DIN, CLK, CS (LOAD для MAX7221) используются для передачи данных по интерфейсу SPI, к выводу ISET подключается резистор, задающий максимальный ток через сегменты индикатора, минимальное сопротивление резистора равно 9,53 кОм, при этом максимальный ток составит 40 мА. Вывод DOUT используется для каскадного подключения нескольких драйверов, этот вывод подключают к входу DIN второго драйвера, тем самым, данные, загруженные в первый драйвер, последовательно загружаются во второй.

Передача данных на драйвер осуществляется пакетами, состоящими из двух байт, первый байт – адрес регистра к которому идет обращение, второй байт непосредственно данные, которые надо записать в этот регистр. Во время передачи пакета данных, необходимо установить низкий логический уровень на линии CS.

Всего имеется 14 регистров, с помощью которых осуществляется управление драйвером. При задании адреса старший полубайт не имеет значения. В следующей таблице показаны адреса регистров и их назначение:

Адрес	Название	Назначение
0xX0	No-Op	Пустой регистр, изменение значения регистра не влияет на работу драйвера.
0xX1	Digit 0	Данные для 0-го индикатора
0xX2	Digit 1	Данные для 1-го индикатора
0xX3	Digit 2	Данные для 2-го индикатора
0xX4	Digit 3	Данные для 3-го индикатора
0xX5	Digit 4	Данные для 4-го индикатора
0xX6	Digit 5	Данные для 5-го индикатора
0xX7	Digit 6	Данные для 6-го индикатора
0xX8	Digit 7	Данные для 7-го индикатора
0xX9	Decode Mode	Регистр управления режимом декодирования данных
0xXA	Intensity	Регистр настройки яркости свечения индикаторов
0xXB	Scan Limit	Регистр настройки количества активных индикаторов
0xXC	Shutdown	Регистр управления режимом работы
0xXF	Display Test	Регистр для тестирования драйвера



Рисунок 38 – матричный светодиодный индикатор.

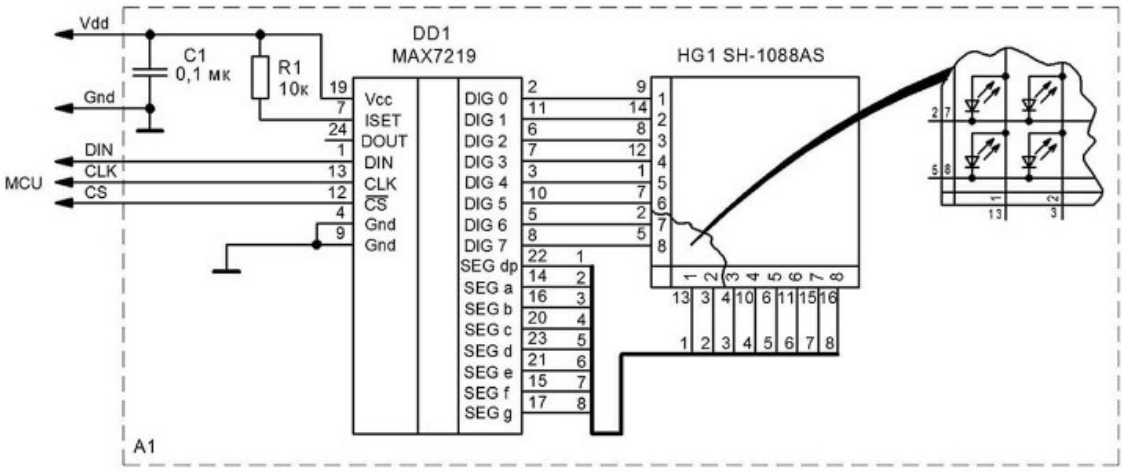


Рисунок 39 – схема подключения матричного светодиодного индикатора к микросхеме управления.

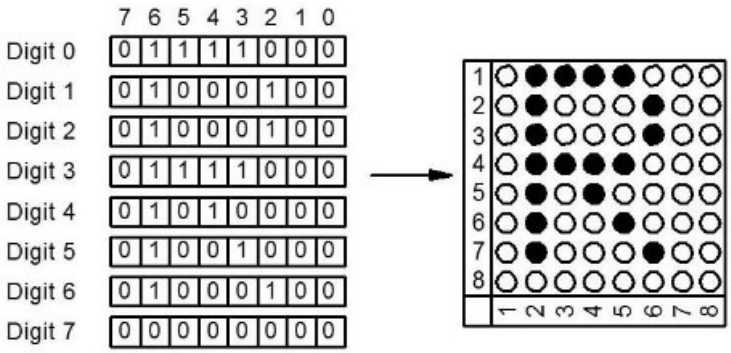


Рисунок 40 – принцип управления матричным светодиодным индикатором.

В практической работе № 5 будут исследованы принципы работы с матричными светодиодными индикаторами.

5.3.4. Жидкокристаллический дисплей

Жидкокристаллический дисплей, ЖКД, (liquid crystal display, LCD, плоский индикатор) — плоский монитор на основе жидких кристаллов

Каждый пиксел ЖК-дисплея состоит из слоя молекул между двумя прозрачными электродами, и двух поляризационных фильтров, плоскости поляризации которых (как правило) перпендикулярны. В отсутствие жидких кристаллов свет, пропускаемый первым фильтром, практически полностью блокируется вторым.

Поверхность электродов, контактирующая с жидкими кристаллами, специально обработана для изначальной ориентации молекул в одном направлении. В TN-матрице эти направления взаимно перпендикулярны, поэтому молекулы в отсутствие напряжения выстраиваются в винтовую структуру. Эта структура преломляет свет таким образом, что до второго фильтра плоскость его поляризации поворачивается, и через него свет проходит уже без потерь. Если не считать поглощения первым фильтром половины неполяризованного света — ячейку можно считать прозрачной. Если же к электродам приложено напряжение — молекулы стремятся выстроиться в направлении поля, что искажает винтовую структуру. При этом силы упругости противодействуют этому, и при отключении напряжения молекулы возвращаются в исходное положение. При достаточной величине поля практически все молекулы становятся параллельны, что приводит к непрозрачности структуры. Варьируя напряжение, можно управлять степенью прозрачности. Если постоянное напряжение приложено в течении долгого времени — жидкокристаллическая структура может деградировать из-за миграции ионов. Для решения этой проблемы применяется переменный ток, или изменение полярности поля при каждой адресации ячейки (непрозрачность структуры не зависит от полярности поля). Во всей матрице можно управлять каждой из ячеек индивидуально, но при увеличении их количества это становится трудновыполнимо, так как растёт число требуемых электродов. Поэтому практически везде в простых ЖКД применяется адресация по строкам и столбцам. Проходящий через ячейки свет может быть естественным — отражённым от подложки (в ЖК-дисплеях без подсветки). Но чаще применяют искусственный источник света, кроме независимости от внешнего освещения это также стабилизирует свойства полученного изображения. Таким образом полноценный ЖК-монитор состоит из электроники, обрабатывающей входной видеосигнал, ЖК-матрицы, модуля подсветки, блока питания и корпуса. Именно совокупность этих составляющих определяет свойства монитора в целом, хотя некоторые характеристики важнее других.

Для создания дешевых устройств на основе микроконтроллеров в основном используются буквенно-цифровые LCD 1602, 1604 или графические LCD 12864. Для простоты освоения методов работы с LCD, при проведения практических работ, будем применять LCD 1602. Общий вид LCD 1602 и способ подключения по параллельному интерфейсу представлен на рисунке 41. На рисунке 42 представлен способ подключения LCD 1602 по последовательному интерфейсу I2C с использованием интерфейсной платы.

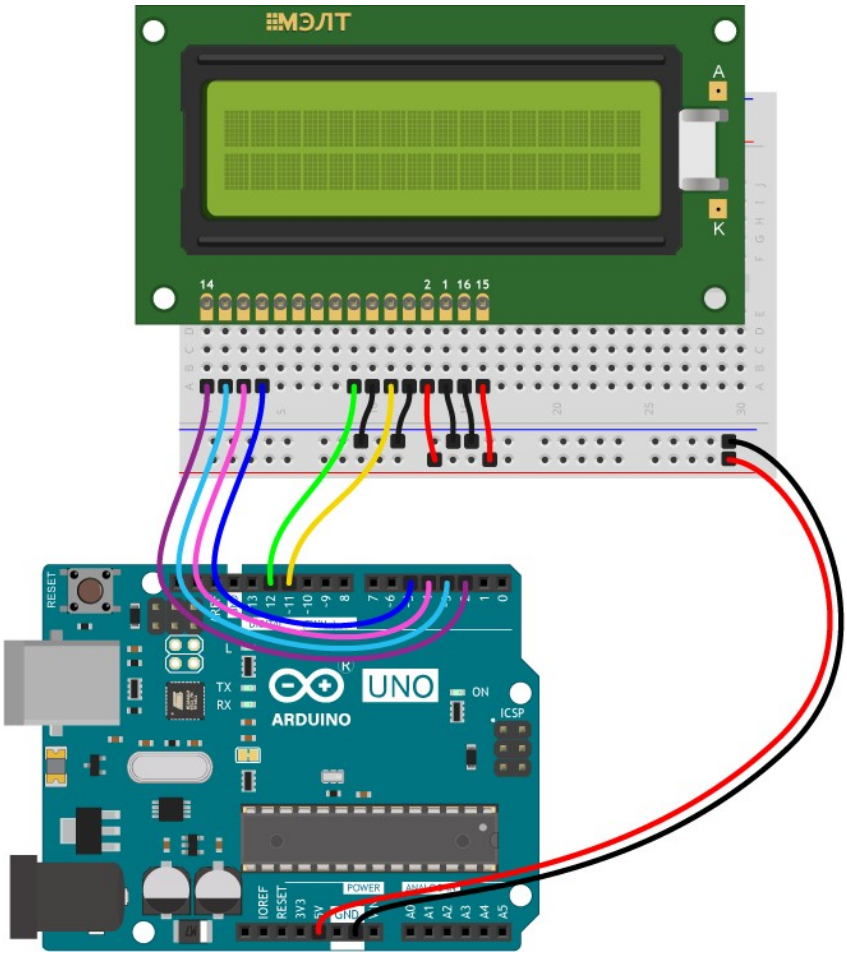


Рисунок 41 – подключение ЖКД по параллельному порту.

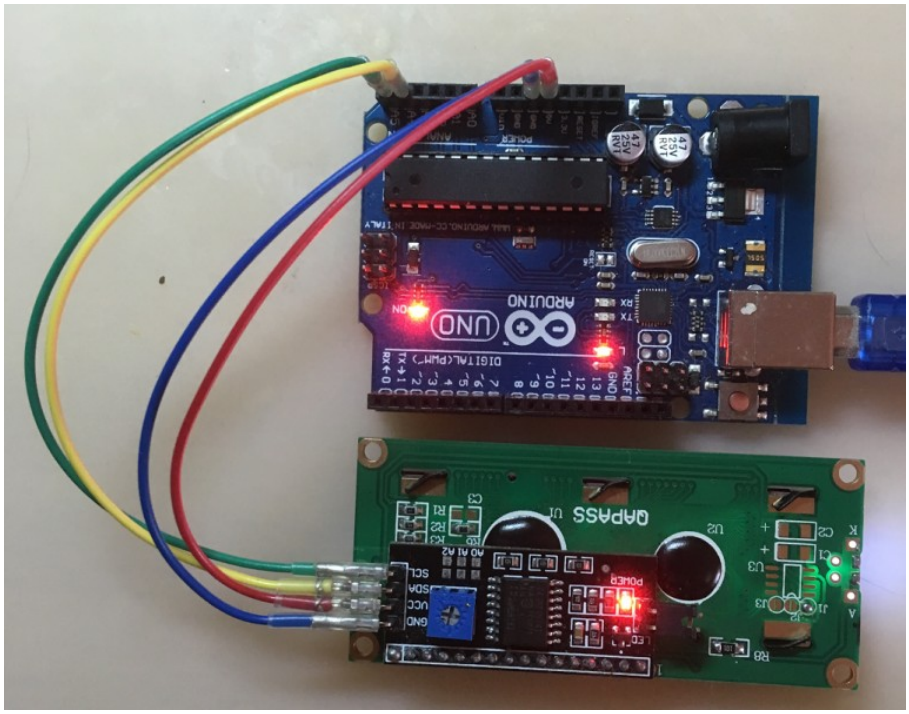


Рисунок 42 – подключение ЖКД по последовательному интерфейсу I2C с использованием интерфейсной платы.

Подключение LCD1602 к Arduino
Модуль оборудован четырех-пиновым разъемом стандарта 2.54мм
SCL: последовательная линия тактирования (Serial CLock)
SDA: последовательная линия данных (Serial DAta)
VCC: «+» питания
GND: «-» питания

5.3.5. Ультразвуковой сонар (измерение расстояния)

В данном разделе, на примере ультразвукового сонара, рассмотрим типовой процесс измерения физических величин.

Ультразвуковой сонар – датчик, построенный на основе излучения и приема звуковой волны в ультразвуковом диапазоне. Волна покидает излучатель, отражается от поверхности вещества и возвращается в приемник. Ультразвуковое устройство просчитывает расстояние до вещества на основе подсчета времени от передачи до приема сигнала и скорости звука в среде.

Рассмотрим простейший ультразвуковой датчик расстояния HC-SR04. Он представляет собой прибор бесконтактного типа, и позволяет обеспечить высокоточное измерение и стабильность. Спектр дальности его измерения составляет от 2 до 400 см. На его работоспособность не проявляет серьезного воздействия электромагнитные излучения и солнечная энергия. В комплексе модуля с HC-SR04 также входят ресивер и трансмиттер.

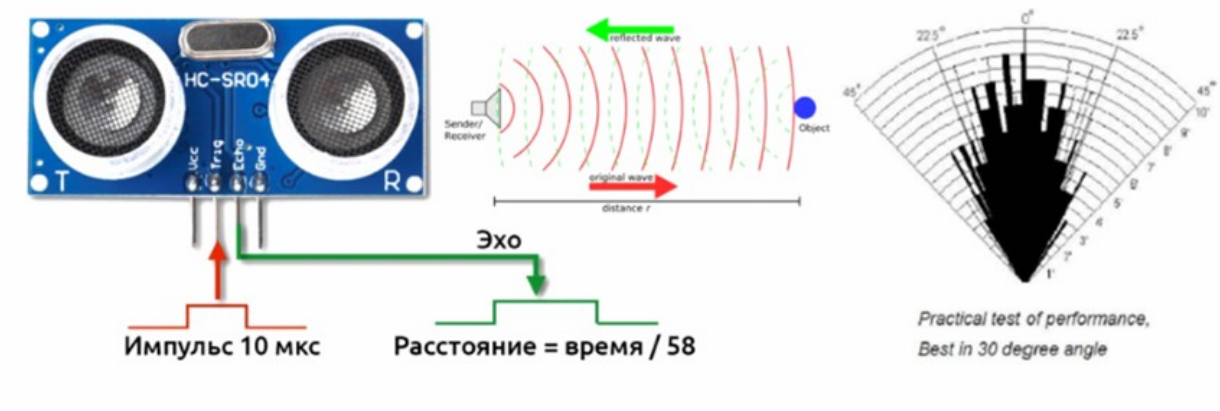


Рисунок 43 – ультразвуковой сонар.

Ультразвуковой дальномер HC-SR04 имеет следующие технические характеристики:
— Питающее напряжение 5В;
— Рабочий параметр силы т ока – 15 мА;
— Сила тока в пассивном состоянии <2 мА;
— Обзорный угол – 15°;
— Сенсорное разрешение – 0,3 см;
— Измерительный угол – 30°;
— Ширина импульса – 10-6 с.
— Ультразвуковой датчик расстояния HC-SR04
— Датчик оснащен четырьмя выводами (стандарт 2, 54 мм):
— Контакт питания положительного типа – +5В;
— Trig (T) – выход сигнала входа;
— Echo (R) – вывод сигнала выхода;
— GND – вывод «Земля».
Точность датчика подчиняется некоторым факторам:
— температуры и влажности воздуха;
— расстояния до объекта;

— расположения относительно датчика (согласно диаграммы излучения);

— качества исполнения элементов модуля датчика.

В основу положения процесса любого ультразвукового датчика закладывается явление отражения акустических волн, распространяющихся в воздухе. Но, стоит отметить, из курса физики знакомо, что скорость распространения звука в воздухе зависит от свойств этого самого воздуха (в главную очередь от температуры). Датчик же, пуская волны и измеряя время до их возврата, не знает в какой из сред они будут распространяться, поэтому часто устанавливают для расчетов определенную среднюю величину (как константу). Таким образом, из-за фактора температуры воздуха HC-SR04 может ошибаться в «трубке» значений от 1-5 мм до 3-5 см.

Фактор расстояния до объекта имеет важное значение, т.к. повышается вероятность отражения от соседних предметов, при том и сам сигнал затухает с расстоянием. Для того, чтобы звук прекрасно отражался от барьеров, датчик пускает ультразвук с частотой 40 кГц. Для данного действия в датчике имеется пьезокерамический излучатель, способный генерировать звук такой высокой частоты. Во-вторых, излучатель устроен, что звук распространяется не во все стороны (как это бывает у обычных динамиков), а в узком направлении.

Стоит отметить, что достоинствами применения ультразвуковых датчиков являются:

— Точность

Наилучшая точность измерений – одно из главных преимуществ приборов данного типа. Паспортная погрешность не превышает 2%. В жизни же она, как правило, снижается до значения, менее 1%. Стоит заметить, что для достижения подобных исходов нужна тщательная настройка и калибровка прибора, требующая участия грамотного специалиста.

— Отсутствие движущихся механических частей.

— Широкий размах измеряемых величин

Широкий размах измеряемых величин, причем неточность измерений в таком широком спектре гарантировано не покинет пределы заявленной, а вероятно будет значительно ниже.

— Малая цена по сравнению с иными типами бесконтактных датчиков;

— Бесконтактный метод измерения (право работать с агрессивными и вязкими средами);

— Надежность измерения, которое не зависит от свойств продукта: химических, физических;

— У части моделей предоставлена возможность автономной работы;

— Маленькие габариты;

— Многофункциональность прибора (определение сопутствующих величин, удаленное информирование GPS, SMS, сигнализация).

Из главных недостатков следует отметить – это блокировка выполнения основной программы микроконтроллера на время измерения, например на трёх метрах это будет 17 мс, а использование нескольких датчиков для последовательного измерения нескольких пространственных величин или обеспечения точности измерения создают весьма серьезную проблему для высокоскоростной обработки данных.

Использование ультразвуковых датчиков уровня – это экономически выгодный выход для не сложных условий эксплуатации, где воздействие негативных факторов окружающей среды на достоверность измерения минимально.

Таким образом, данный датчик наиболее подходит для изучения принципов сбора аналоговой информации, тем более, что миллиметровую точность, с помощью этого датчика, получить абсолютно не проблема, однако необходимо для этого применять методы калибровки на условия окружающей среды и методы фильтрации.

Калибровка:

скорость звука зависит от температуры: при +20°C это 343 м/с, а при -20°C – 318 м/с! А ведь это целых 318/343=7%, что на расстоянии в 1 метр даст погрешность 7 сантиметров.

Таким образом, для нахождения более точного расстояния с поправкой на температуру:

S = code * (0.609 * t + 330.75) / 2000

Данная зависимость является примером калибровочной характеристики датчика при переводе кода значения (code) в физическую величину. Графически эту зависимость можно представить на рисунке 44

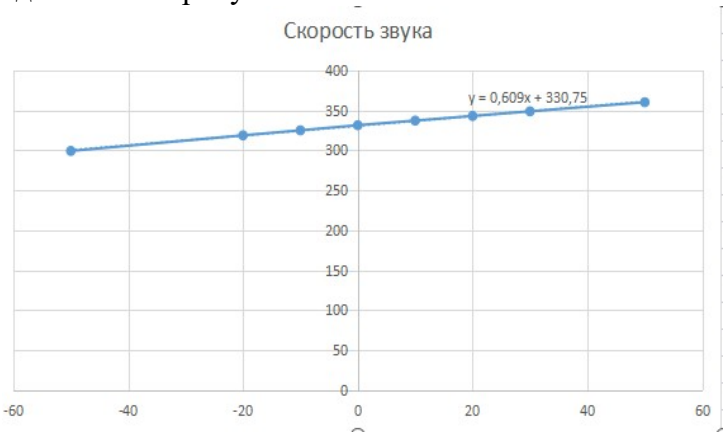


Рисунок 44 – зависимость скорости звука в воздухе от температуры. *Примечание*: следует помнить, что эта зависимость может быть не линейной из-за физических особенностей датчика, поэтому на заводе – изготовителе высокоточные датчики калибруют и заносят эти характеристики (калибровочные характеристики) в паспорт.

Цифровая фильтрация:

Цифровые (программные) фильтры позволяют отфильтровать различные шумы. В следующих примерах будут показаны некоторые популярные фильтры.

1) **Среднее арифметическое** вычисляется как сумма значений, делённая на их количество. Работает следующим образом: в цикле суммируем выборку из нескольких последовательных измерений в какую-нибудь переменную, потом делим на количество измерений.

Особенности использования:

- Отлично усредняет шум любого характера и величины
- Для целочисленных значений количество измерений есть смысл брать из степеней двойки (2, 4, 8, 16, 32…) тогда компилятор оптимизирует деление в сдвиг, который выполняется в сотню раз быстрее. Это если вы совсем гонитесь за оптимизацией выполнения кода
- “Сила” фильтра настраивается размером выборки последовательности измерений
- Делает несколько измерений за один раз, что может приводить к большим затратам времени

Рекомендуется использовать там, где время одного измерения ничтожно мало, или измерения в принципе делаются редко, что явно не подходит для ультразвукового сонара.

2) **Бегущее среднее арифметическое** - данный алгоритм работает по принципу буфера, в котором хранятся несколько последних измерений для усреднения. При каждом вызове фильтра буфер сдвигается, в него добавляется новое значение и убирается самое старое, далее буфер усредняется по среднему арифметическому.

Особенности использования:

- Усредняет последние N измерений, за счёт чего значение запаздывает. Нуждается в тонкой настройке частоты опроса и размера выборки

- Для целочисленных значений количество измерений есть смысл брать из степеней двойки (2, 4, 8, 16, 32…) тогда компилятор оптимизирует деление в сдвиг, который выполняется в сотню раз быстрее. Это если вы совсем гонитесь за оптимизацией выполнения кода

- “Сила” фильтра настраивается размером выборки (NUM_READS)

- Делает только одно измерение за раз, не блокирует код на длительный период

```
float runMiddleArifm(float newVal) { // функция принимает новое значение
  static byte idx = 0; // индекс хранится в функции
  static float valArray[NUM_READ]; // массив хранится в функции
  valArray[idx] = newVal; // пишем каждый раз в новую ячейку
  if (++idx >= NUM_READ) idx = 0; // организация счетчика индекса массива со сбросом в 0 при достижении NUM_READ
  float average = 0; // обнуляем среднее при каждом входе в функцию
  for (int i = 0; i < NUM_READ; i++) {
   average += valArray[i]; // суммируем массив
  }
  return (float)average / NUM_READ; // функция возвращает текущее среднее
}
```

3) **Бегущее среднее э**кспоненциальное **(Running Average)** – самый простой и эффективный фильтр значений, по эффекту аналогичен предыдущему, но гораздо оптимальнее в плане реализации:

```
//фильтрованное знач += (новое знач – фильтрованное знач) * коэффициент
К=0.2  //резкость фильтра 1…0
distFilt += (dist - distFilt) * K; // distFilt - глобальная переменная
```

Особенности использования:

- Самый лёгкий, быстрый и простой для вычисления алгоритм! В то же время очень эффективный

- “Сила” фильтра настраивается коэффициентом (0.0 – 1.0). Чем он меньше, тем плавнее фильтр

- Делает только одно измерение за раз, не блокирует код на длительный период

- Чем чаще измерения, тем лучше работает

- При маленьких значениях коэффициента работает очень плавно, что также можно использовать в своих целях

4)**Адаптивное бегущее среднее**. Чтобы бегущее среднее корректно работало с резко изменяющимися сигналами, коэффициент можно сделать адаптивным, чтобы он подстраивался под резкие изменения значения, например так: если фильтрованное значение “далеко” от реального – коэффициент резко увеличивается, позволяя быстро сократить “разрыв” между величинами. Если значение “близко” – коэффициент ставится маленьким, чтобы хорошо фильтровать шум:

```
// бегущее среднее Экспоненциальное  с адаптивным коэфициентом
// резкость фильтра зависит от модуля разности значений
if (abs(dist - distFilt) > 1.5) k = 0.9;
else k = 0.03;
distFilt += (dist - distFilt) * k;
```

6. ЗАДАНИЯ ПРАКТИЧЕСКИХ РАБОТ

Практическая работа состоит из трех частей:

1) домашняя подготовка «Домашнее задание», его необходимо выполнить перед практической работой;

2) выполнение практической работы «На занятии» производится на лабораторном стенде, в состав которого входят:

- отладочная плата Arduino Uno с кабелем USB для программирования микроконтроллера;
- среда разработки IDE Arduino, установленная на ЭВМ;
- набор элементов схем в зависимости от темы практической работы.

3) защита практической работы «Для защиты» производится после оформления результатов в тетради для практических работ в соответствии с разделом 7.

6.1. Практическая работа №1 «Быстрый старт»

Домашнее задание: Изучить материал разделов 3.1-3.4.

Ответить (в тетради) на контрольные вопросы:

1. Структура программы для микроконтроллера в среде разработки IDE Arduino;
2. Как написать комментарии к программе для микроконтроллера в среде разработки IDE Arduino?
3. Как подключить библиотеку?
4. Как объявить переменную?
5. Основные математические операторы.
6. Какая библиотека отвечает за работу с последовательным портом?
7. Для чего используется монитор порта и плоттер?

На занятии: Необходимо выполнить работу представленную в разделе 4 (Быстрый старт).

Для защиты: продемонстрировать подключение отладочной платы и вывод по последовательному порту данных на монитор, на скорости указанной преподавателем.

6.2. Практическая работа №2 «Основы программирования МК»

Домашнее задание: Изучить материал разделов 3.5-3.10.

Ответить (в тетради) на контрольные вопросы:

1. Какие условные операторы ветвления вы знаете?
2. Какие бывают циклы?
3. Как выйти из цикла?
4. Как пропустить ход в цикле?
5. Для чего нужны функции задержки?
6. Какие бывают режимы работы портов ввода/вывода?
7. Как выбрать источник опорного напряжения для АЦП?
8. Как ограничить диапазон измеряемой величины?
9. Как получить максимально случайную последовательность чисел?

На занятии: Необходимо выполнить работу представленную в разделе 4 (Быстрый старт) модифицировав программу используя раздел «Ветвления и циклы в программе» и «Генератор случайных чисел» в соответствии с таблицей вариантов:

Номер бригады	Ветвления и циклы в программе	Генератор случайных чисел
1	Анализировать переменную tik и выводить: - Serial.println(rnd), если tik четный, - Serial.println(tik) , если tik нечетный	rnd = Генерация случайного числа в диапазоне от 0 до 500
2	Анализировать переменную tik и выводить: - Serial.println(rnd), если tik кратен 10, - Serial.println(tik), если tik нечетный	rnd = Генерация случайного числа в диапазоне от 10 до 300
3	Анализировать переменную tik и выводить: - Serial.println(rnd), если tik четный, - Serial.println(100), если tik = 5, - Serial.println(200), если tik = 7, - Serial.println(300), если tik = 9, - Serial.println(400), если tik = 11, - Serial.println(500), если tik = 13	rnd = Генерация случайного числа в диапазоне от 30 до 900
4	Анализировать переменную tik и выводить: - Serial.println(rnd), если tik нечетный, - Serial.println(100), если tik = 4, - Serial.println(200), если tik = 6, - Serial.println(300), если tik = 8, - Serial.println(400), если tik = 10, - Serial.println(500), если tik = 12	rnd = Генерация случайного числа в диапазоне от 60 до 357
5	Анализировать переменную tik и выводить: - Serial.println(rnd), если tik нечетный, - Serial.println(tik) , если tik кратен 6	rnd = Генерация случайного числа в диапазоне от 10 до 600
6	Анализировать переменную tik и выводить: - Serial.println(rnd), если tik кратен 3, - Serial.println(tik) , если tik кратен 4	rnd = Генерация случайного числа в диапазоне от 90 до 700

Для защиты: продемонстрировать подключение отладочной платы и вывод по последовательному порту данных на монитор.

6.3. Практическая работа №3 «Матричная клавиатура»

Домашнее задание: Изучить материал раздела 5.2, 5.3.1.

Ответить (в тетради) на контрольные вопросы:

1. Как выглядит на блок-схеме алгоритма блок ветвления?
2. Как выглядит на блок-схеме алгоритма блок начала цикла?
3. Как выглядит на блок-схеме алгоритма блок конца цикла?
4. Как выглядит на блок-схеме алгоритма блок цикла со счетчиком?
5. Как выглядит на блок-схеме алгоритма блок вызова процедуры?
6. Что из себя представляет матричная клавиатура?
7. Что такое динамическое сканирование?

На занятии:

1) Собрать схему представленную на рисунке 45 с учетом значений переменных rowPins и colPins (см. текст программы №2)

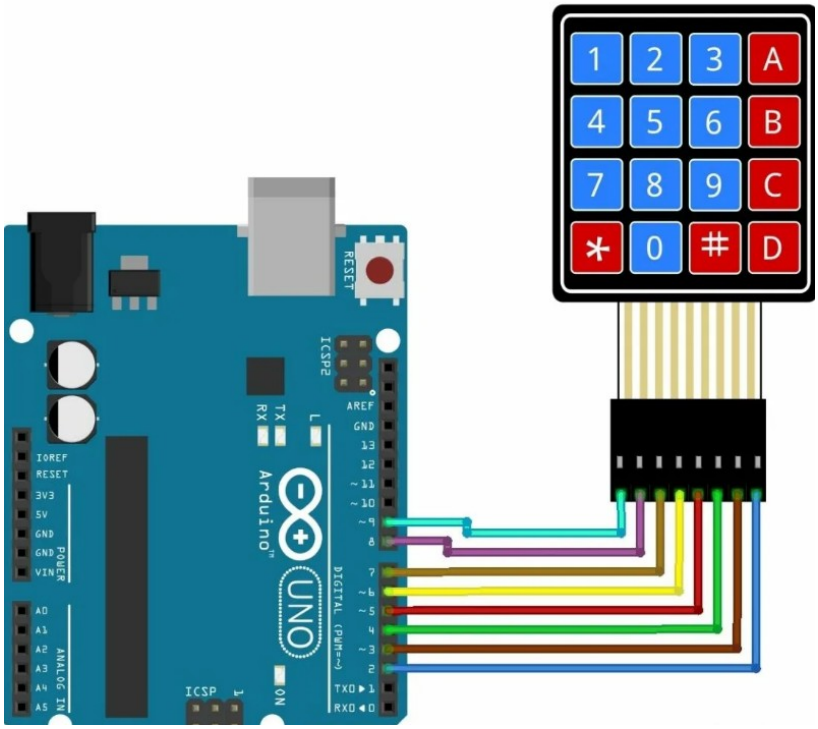


Рисунок 45 – схема подключения матричной клавиатуры.

2) Ввести и проанализировать текст программы 2 для микроконтроллера:

3) Провести проверку и загрузку программы в соответствии с разделом 4.

4) Составить блок-схему одного из алгоритмов, а также модифицировать программу для вызова print_ch(keys()) по прерыванию двух столбцов матричной клавиатуры в соответствии с вариантом

Номер бригады	Алгоритм функций	Создать 2 прерывания для столбцов
1,4	print_ch	1 и 2
2,5	get_keys_click	3 и 4
3,6	keys	1 и 4

5) Проанализировать два способа обработки нажатия клавиш (программно-управляемый и по прерыванию) и сделать заключение в отчете

Для защиты: продемонстрировать подключение отладочной платы и вывод по последовательному порту данных на монитор.

6.4. Практическая работа №4 «Семисегментный индикатор»

Домашнее задание: Изучить материал раздела 5.3.2, 5.3.3.

Ответить (в тетради) на контрольные вопросы:

1. Что такое UART?
2. Что такое SPI?
3. Что такое I2C?
4. Преимущества светодиодных индикаторов.

5. Недостатки светодиодных индикаторов.
6. Как оптимизировать использование цифровых портов (контактов) микроконтроллеров для подключения светодиодных индикаторов?
7. Какой из цифровых интерфейсов самый скоростной?

На занятии:

1) Собрать схему представленную на рисунке

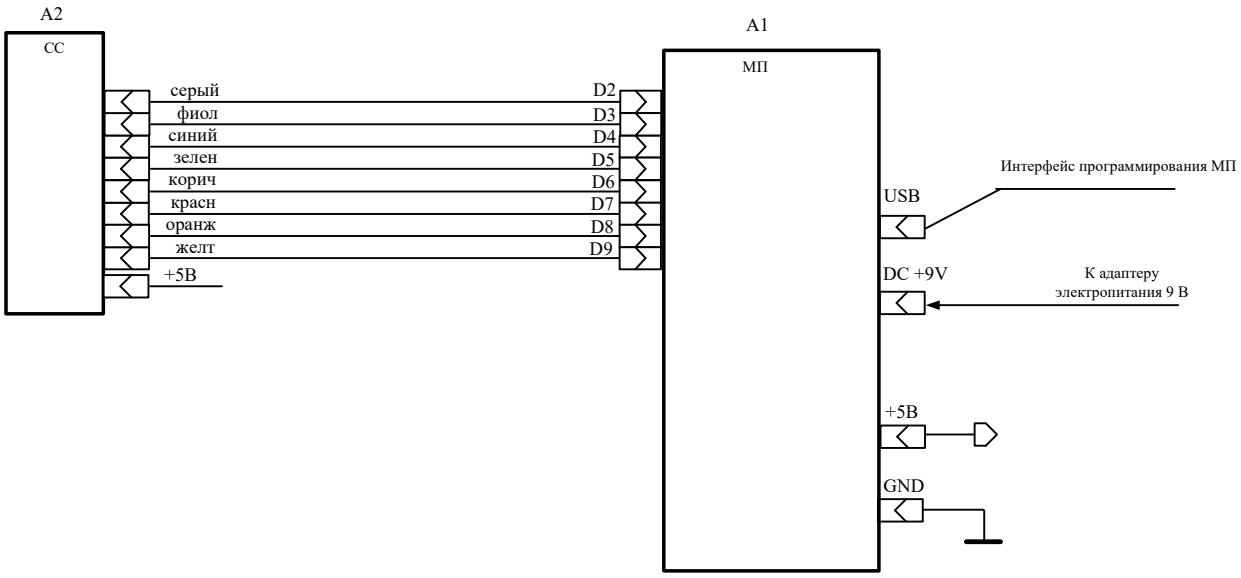


Таблица 1

Поз. обозначение	Наименование	Кол.
A1	Макетная плата Arduino UNO	1
A2	Макетная плата с семисегментным индикатором	1

Рисунок 46 – схема электрическая подключения семисегментного индикатора.

2) Ввести и проанализировать текст программы 4 для микроконтроллера.

3) Провести проверку и загрузку программы в соответствии с разделом 4.

4) Доработать программу в соответствии с заданием

Номер бригады	Задание
1	Вывести цифру «4» и через 2 секунды цифру «0» на 0,5 сек.
2	Вывести цифру «5» и через 3 секунды цифру «2» на 0,5 сек
3	Вывести символ “E” и мерцающую точку с частотой 2Гц
4	Вывести символ “F” и мерцающую точку с частотой 5Гц
5	Вывести цифру «8» и через 3 секунды цифру «1» на 0,5 сек
6	Вывести цифру“9” и мерцающую точку с частотой 1Гц

Для защиты: продемонстрировать подключение отладочной платы и выполнение действий указанных в варианте задания бригады.

6.5. Практическая работа №5 «Светодиодный индикатор»

Домашнее задание: Изучить материал раздела 5.1.

Ответить (в тетради) на контрольные вопросы:

1. Что такое схема?
2. Какая из всех типов схем применяется для макетирования и почему?
3. Основные части электрической схемы.
4. Как на электрической схеме обозначается микроконтроллер?
5. Как на электрической схеме обозначаются разъем?
6. Как на электрической схеме обозначить заземление?

На занятии:

1) Собрать схему представленную на рисунке 47.

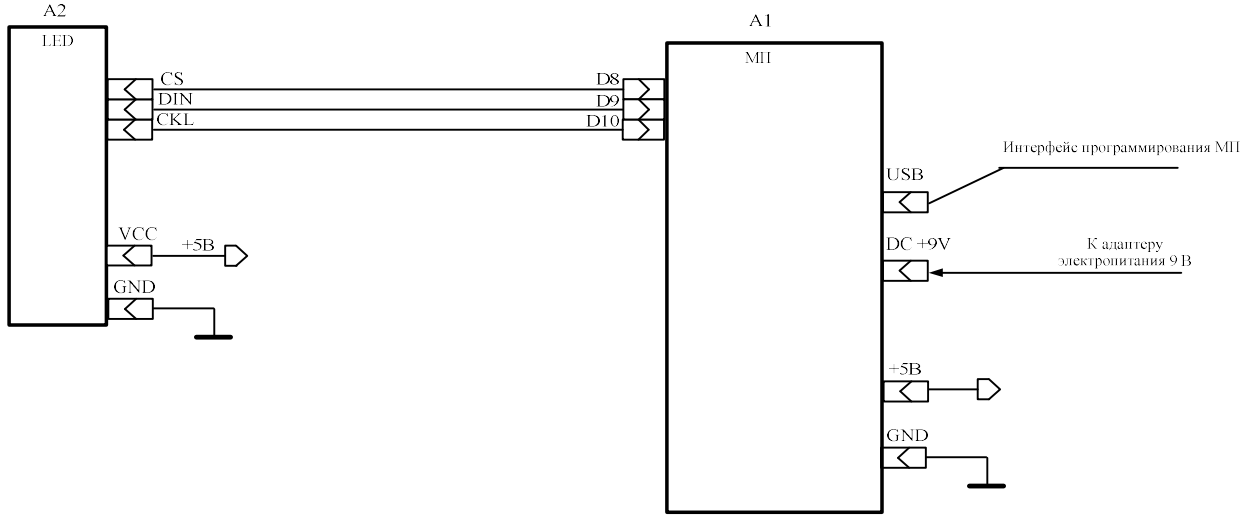


Таблица 1

Поз. обозна-чение	Наименование	Кол.
A1	Макетная плата Arduino UNO	1
A2	Макетная плата с LED 8x8	1

Рисунок 47 – схема электрическая подключения светодиодного индикатора.

2) Ввести и проанализировать текст программы 3 для микроконтроллера.

3) Провести проверку и загрузку программы в соответствии с разделом 4.

4) Доработать программу в соответствии с заданием

Номер бригады	Задание
1	Последовательно символы «A», «B», «C» с интервалом 1 сек
2	Последовательно символы «1», «2», «3» с интервалом 1 сек
3	Последовательно символы «D», «E», «F» с интервалом 1 сек
4	Последовательно символы «9», «8», «7» с интервалом 1 сек
5	Последовательно символы «0», «2», «5» с интервалом 1 сек
6	Последовательно символы «X», «Y», «Z» с интервалом 1 сек

5) Составить блок – схему передачи информации по протоколу SPI (процедуры Write_Max7219 и Write_Max7219_byte).

Для защиты: продемонстрировать подключение отладочной платы и выполнение действий указанных в варианте задания бригады.

6.6. Практическая работа №6 «Жидкокристаллический дисплей»

Домашнее задание: Изучить материал раздела 2, 5.3.4.

Ответить (в тетради) на контрольные вопросы:

1. Для чего нужны отладочные платы?
2. Что представляет собой пиксель в жидкокристаллическом дисплее?
3. Для чего в жидкокристаллическом дисплее используются поляризационные фильтры?
4. Для чего простые жидкокристаллические дисплеи подключают к микроконтроллеру по последовательному порту?
5. Какая функция инициализирует ЖК-дисплей?
6. Какая функция включает подсветку ЖК-дисплея?
7. Какая функция устанавливает позицию на ЖК-дисплее?
8. Какая функция позволяет напечатать текст на ЖК-дисплее?

На занятии:

1) Собрать схему представленную на рисунке

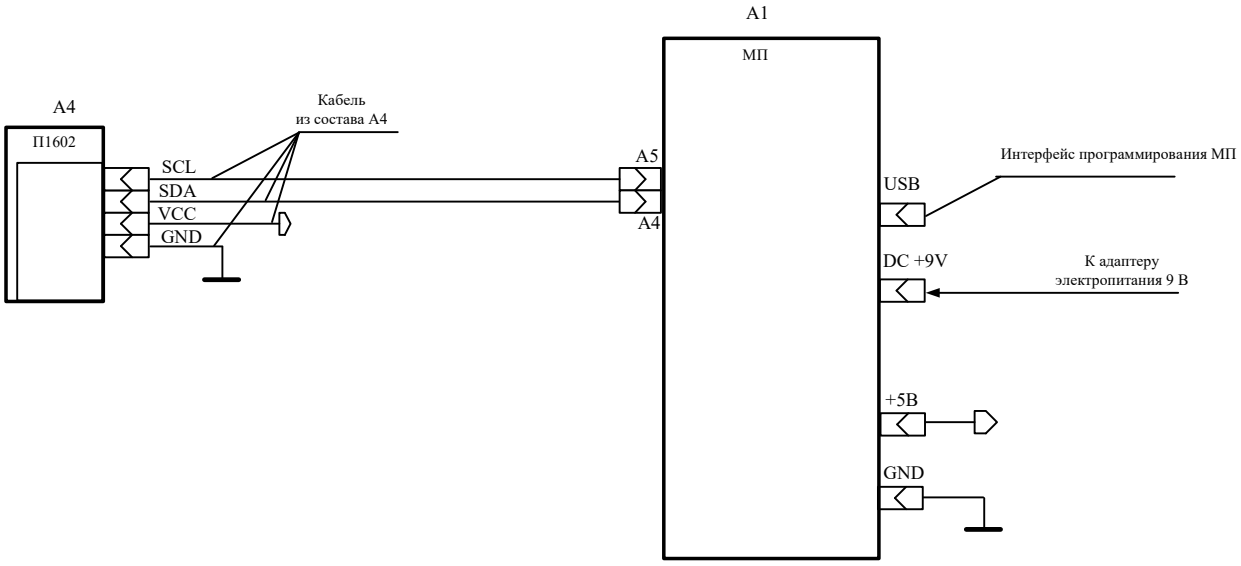


Таблица 1

Поз. обозначение	Наименование	Кол.
A1	Макетная плата Arduino UNO	1
A2	Панель 1602	1

Рисунок 48 – схема электрическая подключения ЖКД.

2) Разработать самостоятельно текст программы вывода текущего времени на экран жидкокристаллического дисплея 1602 с использованием драйвера LiquidCrystal_PCF8574.

Начало программы:

```
#include < LiquidCrystal_PCF8574.h>
/* Устанавливаем ЖК-дисплей по адресу 0x27, 16 символов и 2 строки */
 LiquidCrystal_PCF8574 lcd(0x27);
// Назначение порта 0x27 PCF8574 для переменной lcd
```

Библиотека обеспечивает следующие функции:

Функция	**Назначение**
lcd.init()	Инициализация ЖК-дисплея.
lcd.backlight()/lcd.noBacklight()	Включение/выкл. подсветки дисплея.
lcd.setCursor(uint8_t, uint8_t)	Установка позиции курсора.
lcd.clear()	Возвращает курсор в начало экрана
lcd.print(string)	Печать строки или переменных
lcd.home()	Возвращает курсор в начало экрана и удаляет все, что было на экране до этого
lcd.write(uint8_t)	Позволяет вывести одиночный символ на экран
lcd.cursor()	Показывает курсор на экране.
lcd.noCursor()	Скрывает курсор на экране
lcd.blink()	Курсор мигает (если до этого было включено его отображение)
lcd.noBlink()	Курсор не мигает (если до этого было включено его отображение)
lcd.display()	Позволяет включить отображение
lcd.noDisplay()	Позволяет отключить отображение
lcd.scrollDisplayLeft()	Прокручивает экран на один знак влево
lcd.scrollDisplayRight()	Прокручивает экран на один знак вправо
lcd.autoscroll()	Позволяет включить режим автопрокручивания. В этом режиме каждый новый символ записывается в одном и том же месте, вытесняя ранее написанное на экране
lcd.noAutoscroll()	Позволяет выключить режим автопрокручивания. В этом режиме каждый новый символ записывается в одном и том же месте, вытесняя ранее написанное на экране
lcd.leftToRight()	Установка направление выводимого текста — слева направо
lcd.rightToLeft()	Установка направление выводимого текста — справа налево
lcd.createChar(uint8_t, uint8_t[])	Создает символ. Первый параметр — это номер (код) символа от 0 до 7, а второй — массив 8 битовых масок для создания черных и белых точек

3) Провести проверку и загрузку программы в соответствии с разделом 4.

Для защиты: продемонстрировать подключение отладочной платы и выполнение действий указанных в задании.

6.7. Практическая работа №7 «Измерение физических величин»

Домашнее задание: Изучить материал раздела 5.3.5.

Ответить (в тетради) на контрольные вопросы:

1. Что представляет собой ультразвуковой сонар?
2. Как произвести перевод значений датчика в физические величины?
3. Как ограничить измеряемый диапазон физических величин?
4. Для чего нужна цифровая фильтрация?
5. Принцип работы фильтрации средним арифметическим.
6. Принцип работы фильтрации «Бегущим средним».
7. Принцип адаптивной фильтрации.

На занятии:

1) Собрать схему, представленную на рисунке 49.

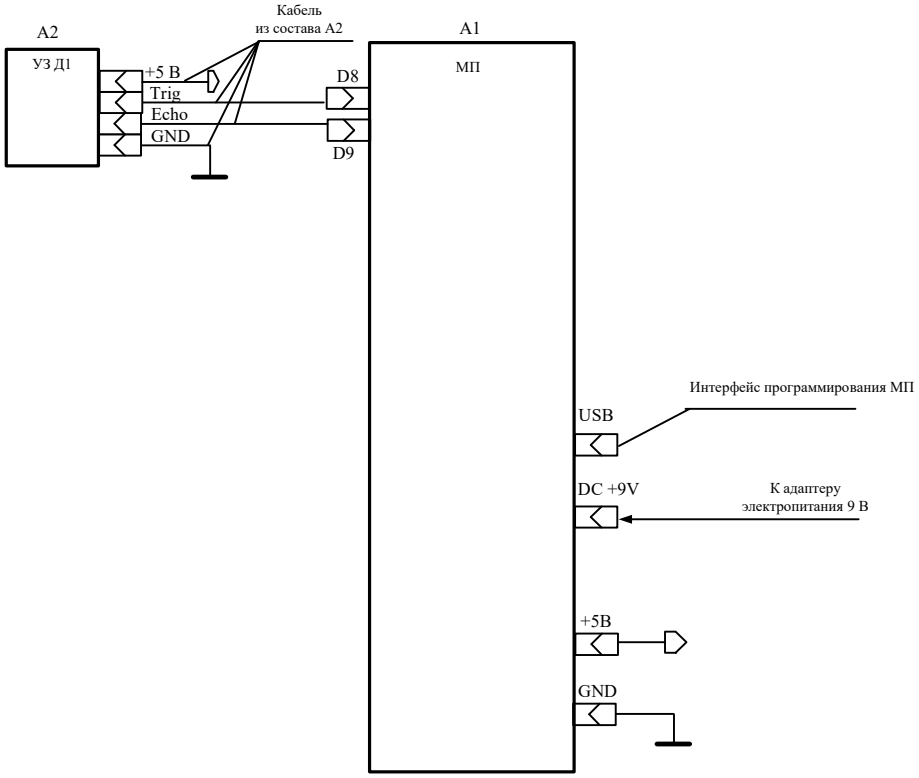


Таблица 1

Поз. обозна-чение	Наименование	Кол.
A1	Макетная плата Arduino UNO	1
A2	Ультразвуковой датчик HC-SR04	1

Рисунок 49 – схема электрическая подключения ультразвукового датчика.

2) Ввести и проанализировать текст программы 5 для микроконтроллера.

3) Провести проверку и загрузку программы в соответствии с разделом 4.

4) Доработать программу добавив в нее три функции фильтрации данных, получаемых от датчика и вывести значение на трех параллельных графиках плоттера. Значения на

графиках зафиксировать по 10 точкам и перенести в отчет. Сделать выводы по фильтрации параметров.

Для защиты: продемонстрировать подключение отладочной платы и продемонстрировать три варианта фильтрации измеряемой величины.

6.8. Практическая работа №8 «Проектирование электронного устройства»

Домашнее задание: Повторить весь материал методического пособия.

Разработать:

1. Схему электронного устройства в соответствии таблицей задания.
2. Разработать блок-схему алгоритма программы микроконитроллера.
3. Разработать программу в среде IDE Arduino, используя облачный сервис [5]:

Задание:

Номер бригады	Схема	Функциональные требования
1	К микроконтроллеру на отладочной плате Arduino UNO подключить матричную клавиатуру 4x4 (по параллельному интерфейсу) и ЖК дисплей 1602 (по интерфейсу I2C)	Вывести на ЖК дисплей вводимое с клавиатура время в формате «ЧЧ:ММ:СС». После завершения ввода запустить отображение часов реального времени.
2	К микроконтроллеру на отладочной плате Arduino UNO подключить матричную клавиатуру 4x4 (по параллельному интерфейсу) и семисегментный индикатор (по параллельному интерфейсу)	Вывести на семисегментный индикатор число с клавиатуры, после ввода запустить обратный отчет секунд до 0. Реализовать повторный ввод нового числа с запуском обратного отчета.
3	К микроконтроллеру на отладочной плате Arduino UNO подключить датчик HC-SR04 (по интерфейсу SPI) и ЖК дисплей 1602 (по интерфейсу I2C)	Измерить расстояние датчиком и вывести его значение в миллиметрах на дисплей. Измеряемый диапазон величин от 1 см до 1 м. При измерении применять фильтрацию значений и не допускать недостоверные выбросы.
4	К микроконтроллеру на отладочной плате Arduino UNO подключить матричную клавиатуру 4x4 (по параллельному интерфейсу) и матричный светодиодный модуль 8x8 (по интерфейсу SPI)	Вывести на матричный светодиодный модуль 8x8 число с клавиатуры, после ввода запустить обратный отчет секунд до 0. Реализовать повторный ввод нового числа с запуском обратного отчета.
5	К микроконтроллеру на отладочной плате Arduino UNO подключить датчик HC-SR04 (по интерфейсу SPI) и матричный светодиодный модуль 8x8 (по интерфейсу SPI)	Измерить расстояние датчиком и вывести его значение на гистограмму в измеряемом диапазоне от 10 см до 1 м. То есть при расстоянии более 1 м - все светодиоды д.б. включены, менее 10 см – выключены, промежуточное состояние светодиодов табло линейно зависит от измеренного расстояния, для этого нужно варьировать или яркостью или количеством одновремен-

Номер бригады	Схема	Функциональные требования
		но включенных светодиодов
6	К микроконтроллеру на отладочной плате Arduino UNO подключить матричную клавиатуру 4x4 (по параллельному интерфейсу) и ЖК дисплей 1602 (по интерфейсу I2C)	Вывести на ЖК дисплей вводимое с клавиатура время в формате «СС.МИЛЛИСЕК». После завершения ввода запустить обратный отчет времени до 0. При достижении 0 вывести сообщение «СТОП».

На занятии:

1) Сдать на проверку тетрадь по практическим работам, в которой проведена разработка схем и алгоритмов. Получить допуск к работе.

2) Собрать разработанную схему.

3) Ввести в микроконтроллер программу и отладить ее работу.

Для защиты: продемонстрировать подключение отладочной платы и продемонстрировать реализованную функциональность.

7. СОДЕРЖАНИЕ ОТЧЁТА

Отчёты по практическим работам оформляются в отдельной тетради и должны содержать:

- цель и задачи работы;
- ответы на контрольные вопросы;
- структурную схему устройства;
- текст программы на языке С;
- выводы по работе.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. . Белов А.В. Микроконтроллеры AVR. От азов программирования до создания практических устройств. 2016, Москва, Наука и техника. 544с.

2. https://arduino.ru/Hardware

3. http://www.gaw.ru.

4. ГОСТ 19.701-90 «Схемы алгоритмов программ, данных и систем»

5. https://wokwi.com/arduino/new?template=arduino-uno (бесплатный облачный сервис для разработки электрических схем и программ микроконтроллеров)

ПРИЛОЖЕНИЕ

Примеры программ для выполнения практических работ.

ПРОГРАММА 1. – «Hello Word!»

```
//объявление глобальных переменных
int led = 13;
int tik=0;

void setup() {

  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);
  Serial.begin(9600);  // Открываем порт на скорости 9600 бод.
  Serial.println("Hello Word!");

}

void loop() {
  digitalWrite(led, HIGH);
  delay(500);  // задержка между вкл/выкл
  digitalWrite(led, LOW);
  tik++;
  Serial.println(tik);  // Считываем значение и выводим в порт
  delay(500);  // задержка между циклами
}
```

ПРОГРАММА 2. – Матричная клавиатура.

```
//определение глобальных переменных
bool flag_key_press=false; //флаг признака нажатия клавиши
char mem=0;
char key_press;
//инициализация переменных для определения клавиш
const byte ROWS = 4; //число строк у нашей клавиатуры
const byte COLS = 4; //число столбцов у нашей клавиатуры

char hexaKeys[ROWS][COLS] = {// здесь мы располагаем названия наших клавиш, как на клавиатуре,для удобства пользования
{'1','4','7','*'},
{'2','5','8','0'},
{'3','6','9','#'},
{'A','B','C','D'}
};
byte rowPins[ROWS] = {7, 6, 5, 4}; //к каким выводам подключаем управление строками
byte colPins[COLS] = {11, 10, 9, 8}; //к каким выводам подключаем сканирование столбцов
```

```
void setup() {
  Serial.begin(9600);  // Открываем порт на скорости 9600 бод.
  Serial.print("Load normal. \n>");
  pinMode(rowPins[0], OUTPUT);
  pinMode(rowPins[1], OUTPUT);
  pinMode(rowPins[2], OUTPUT);
  pinMode(rowPins[3], OUTPUT);
  pinMode(colPins[0], INPUT_PULLUP);
  pinMode(colPins[1], INPUT_PULLUP);
  pinMode(colPins[2], INPUT_PULLUP);
  pinMode(colPins[3], INPUT_PULLUP);
}

void loop() {
 print_ch(get_keys_click(keys()));
}

// функция определения клавиша
char keys(void)
{
 char ch=0;
 for (int i=0;i< ROWS;i++)
{
 digitalWrite(rowPins[i], LOW);
 for (int j=0;j< COLS;j++)
  {
   if (digitalRead(colPins[j])==LOW)
   {
    ch=hexaKeys[i][j];
   }
  }

 digitalWrite(rowPins[i], HIGH);
}
 return ch;
}
// функция определения клика клавиши
char get_keys_click(char ch_in)
{
 char ch_out=0;

 if (ch_in==0){
  if (flag_key_press){
    ch_out=mem;
  }
  flag_key_press=false;
 }
 else{
  flag_key_press=true;
  mem=ch_in;
```

```
  }

  return ch_out;
}

// функция посимвольной печати
void print_ch(char ch)
{
 if (ch!=0)
 {
  if (ch=='#'){
  Serial.print("\n>");  // приглашение ввода новой строки

  }
  else{
   Serial.print(ch);  // Считываем значение и выводим в порт
  }
 }
}
```

ПРОГРАММА 3. – Светодиодная матрица.

```
unsigned char i;
unsigned char j;
/*Port Definitions*/
int Max7219_pinCLK = 10;
int Max7219_pinCS = 8;
int Max7219_pinDIN = 9;

void Write_Max7219_byte(unsigned char DATA)
{
 unsigned char i;
 digitalWrite(Max7219_pinCS,LOW);
 for(i=8;i>=1;i--)
 {
  digitalWrite(Max7219_pinCLK,LOW);
  digitalWrite(Max7219_pinDIN,DATA&0x80);// Extracting a bit data
  DATA = DATA<<1;
  digitalWrite(Max7219_pinCLK,HIGH);
 }
}
void Write_Max7219(unsigned char address,unsigned char dat)
{
 digitalWrite(Max7219_pinCS,LOW);
 Write_Max7219_byte(address); //address，code of LED
 Write_Max7219_byte(dat); //data，figure on LED
 digitalWrite(Max7219_pinCS,HIGH);
}

void Clear_Max7219(void)
{
  Write_Max7219(1,0b00000000);
  Write_Max7219(2,0b00000000);
  Write_Max7219(3,0b00000000);
  Write_Max7219(4,0b00000000);
  Write_Max7219(5,0b00000000);
  Write_Max7219(6,0b00000000);
  Write_Max7219(7,0b00000000);
  Write_Max7219(8,0b00000000);
}

void Init_MAX7219(void)
{
 Write_Max7219(0x09, 0x00); //decoding ：BCD
 Write_Max7219(0x0a, 0x00); //brightness
 Write_Max7219(0x0b, 0x07); //scanlimit；8 LEDs
 Write_Max7219(0x0c, 0x01); //power-down mode：0，normal mode：1
 Write_Max7219(0x0f, 0x00); //test display：1；EOT，display：0
}
```

```
void setup()
{
  pinMode(Max7219_pinCLK,OUTPUT);
  pinMode(Max7219_pinCS,OUTPUT);
  pinMode(Max7219_pinDIN,OUTPUT);
  delay(50);
  Init_MAX7219();
  delay(500);
  Clear_Max7219();
}
void loop()
{
Write_Max7219(1,0b00000000);
Write_Max7219(2,0b01100110);
Write_Max7219(3,0b01100110);
Write_Max7219(4,0b00000000);
Write_Max7219(5,0b10011001);
Write_Max7219(6,0b10000001);
Write_Max7219(7,0b01000010);
Write_Max7219(8,0b00111100);
for(int i=0;i<0x0F; i++)
{
   delay(100);
   Write_Max7219(0x0a, i); //brightness
}
}
```

ПРОГРАММА 4. – Семисегментный индикатор.

```
int PIN[8]={2,3,5,6,7,9,8,4};

void setup() {
  // начальная установка индикатора
  for (int _i=0; _i<9; _i++)
  {
   pinMode(PIN[_i],OUTPUT);
   digitalWrite(PIN[_i], HIGH);
  }
}
void loop() {

  for (int i=0; i<9; i++)
  {
    digitalWrite(PIN[i], LOW);//зажечь i-й сегмент
    delay(500);              //пауза 0,5 сек
    digitalWrite(PIN[i], HIGH);//погасить i-й сегмент
    delay(50);              //пауза 0,05 сек
  }
}
```

ПРОГРАММА 5. – Измерение расстояния.

```
// описание ПИН подключения датчика
#define HC_TRIG 3
#define HC_ECHO 2
void setup() {
  Serial.begin(9600); // для связи
  pinMode(HC_TRIG, OUTPUT); // trig выход
  pinMode(HC_ECHO, INPUT); // echo вход
}
void loop() {
  float dist = getDist(); // получаем расстояние
  Serial.println(dist); // выводим
  delay(50);
}
// функция получения расстояния
float getDist() {
// импульс 10 мкс
  digitalWrite(HC_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(HC_TRIG, LOW);
  // измеряем время ответного импульса
  uint32_t us = pulseIn(HC_ECHO, HIGH);
  // считаем расстояние и возвращаем
return (us / 58.2);
}
```

Оператор ЭДО ООО "Компания "Тензор"
ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
ПОДПИСАНО ЗАВЕДУЮЩИМ КАФЕДРЫ — **ФГБОУ ВО "РГРТУ", РГРТУ,** Корячко Вячеслав Петрович, Заведующий кафедрой САПР — **26.10.26** 14:37 (MSK) — Простая подпись
ПОДПИСАНО ЗАВЕДУЮЩИМ ВЫПУСКАЮЩЕЙ КАФЕДРЫ — **ФГБОУ ВО "РГРТУ", РГРТУ,** Корячко Вячеслав Петрович, Заведующий кафедрой САПР — **26.10.26** 14:37 (MSK) — Простая подпись