МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
«Национальный исследовательский ядерный университет «МИФИ»
Саровский физико-технический институт-филиал НИЯУ МИФИ

Физико-технический факультет

Кафедра квантовой электроники

Г.С. Рогожников

Прикладные физико- технические и компьютерные методы исследований: автоматизированные системы управления оптическими и оптико-механическими устройствами.

для студентов, обучающихся по направлению 03.03.01 «Прикладные математика и физика»

Саров
2016

Рогожников Г.С.

Прикладные физико-технические и компьютерные методы исследований: автоматизированные системы управления оптическими и оптико-механическими устройствами. Пособие для высших учебных заведений. – Саров, 2016. – 4,3 п/л.

Пособие содержит материалы, необходимые для успешного освоения курса «Прикладные физико-технические и компьютерные методы исследований». Информация приведена в контексте работы с оптическими и оптомеханическими устройствами, что может быть полезным для студентов кафедр квантовой электроники, экспериментальной физики, а также ядерной и радиационной физики в процессе изучения спецкурсов и НИРС.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Развитие современной науки и техники происходит во всё возрастающем темпе. Если еще в конце 20-го века ученый имел достаточно времени на скрупулезный анализ и перепроверку аналитически или вручную полученных результатов, то сегодня малейшее промедление чревато существенным отставанием от достижений научных сообществ, работающих в этом же направлении. Вопрос оптимизации процесса исследовательской работы привел к повсеместному внедрению компьютерных технологий на всех этапах. В расчетно-теоретических исследованиях это вызвало появление суперкомпьютерных вычислений, в экспериментальных – создание автоматизированных систем управления. Первоначально автоматизированные системы управления не имели успеха из-за скудности компонентной базы и сложности в использовании. Однако, вскоре массивные стойки плат сбора данных и обработки информации заменили так называемые микропроцессорные устройства или «микроконтроллеры» – аппараты, схожие по функционалу с персональными компьютерами, но предназначенные для выполнения специфических задач. Типичный микроконтроллер, в отличие от ПК, кроме интерфейсов ввода-вывода информации, ПЗУ, ОЗУ и микропроцессора, содержит усилители сигналов, цифро-аналоговые и аналого-цифровые преобразователи и прочие элементы, позволяющие задействовать микроконтроллер в экспериментальной физике. Возросшая производительность и быстродействие микропроцессоров стали основанием для использования микроконтроллеров в исследованиях, включающих в себя управление быстропротекающими процессами. Подобные исследования часто встречаются в физике высоких плотностей энергии и направленных потоков излучения, в частности – в области физики лазеров и лазерной техники. В данном методическом пособии рассматривается применение микроконтроллеров в разрезе автоматизации лазерных, оптических и оптомеханических устройств для ускорения процесса проведения исследований. В качестве примера для изучения микроконтроллеров был выбран класс устройств под названием Arduino – аппаратов начального уровня, предоставляющих широкий функционал и не требующих длительной специальной подготовки перед практическим использованием. Платформа Arduino в настоящий момент используется во всем мире для обучения студентов основам автоматизированных систем управления, а также для практического применения как в простых, так и достаточно сложных экспериментальных стендах и установках. Знания, полученные в процессе данного курса, являются базовыми для

последующего изучения дисциплины «Компьютерные технологии в науке и производстве».

ПЛАТФОРМА ARDUINO

Arduino – аппаратно-программная платформа с открытой архитектурой и исходным кодом, основанная на специально разработанной плате ввода-вывода и обработки информации, и среды разработки. Arduino может применяться как для использования в автоматических системах управления, так и в автоматизированных, для чего может быть связана с внешними управляющими устройствами или программой на персональном компьютере.

Arduino отличается от других представленных на рынке платформ следующими возможностями:

- Кроссплатформенность: Arduino может работать под управлением как Windows, так и Macintosh и Linux.
- IDE (среда разработки) Arduino основана на использовании языка Processing (или Wiring), похожего по синтаксису на известный большинству студентов CPP или C#
- Микроконтроллер программируется через интерфейс USB, а не через последовательный порт, что важно в свете исчезновения интерфейсов RS-232 в современных персональных компьютерах .
- Arduino использует открытую архитектуру и открытое программное обеспечение, что позволяет сторонним производителям выпускать дополнения и расширения существующих вариантов микроконтроллера.

Изначально проект Arduino развивался как образовательный и поэтому он отлично подходит начинающим для быстрого обучения.

Arduino состоит из двух основных частей - платы Arduino, которая является частью аппаратного обеспечения, и среды разработки (IDE) Arduino - программного обеспечения, которое запускается на персональном компьютере. IDE используется для создания скетчей (специальных компьютерных программ), которые выгружаются на плату Arduino.

!	Скетч – программа для микроконтроллера Arduino

Плата Arduino – миниатюрная плата микроконтроллера, содержащая помимо обрабатывающего устройства, интерфейсы ввода-вывода информации и преобразователи сигналов. По производительности микроконтроллер существенно уступает персональному компьютеру, но стоит значительно дешевле и приспособлен для разработки целого ряда устройств. В основе микроконтроллера Arduino как правило лежит процессор ATmega168 или ATmega328 (рис.1).

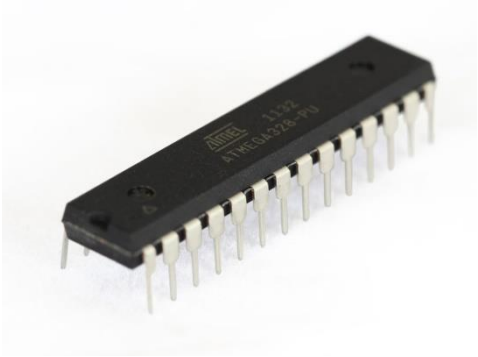
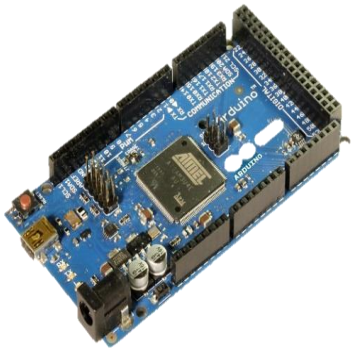
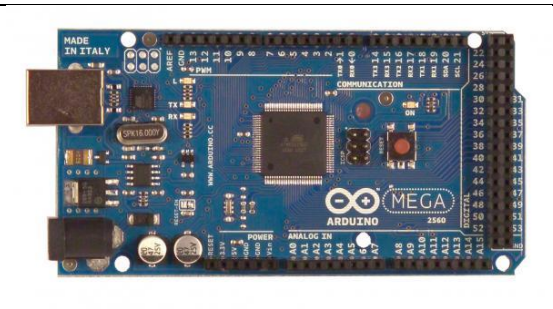
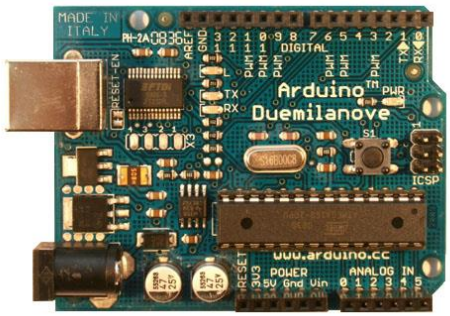


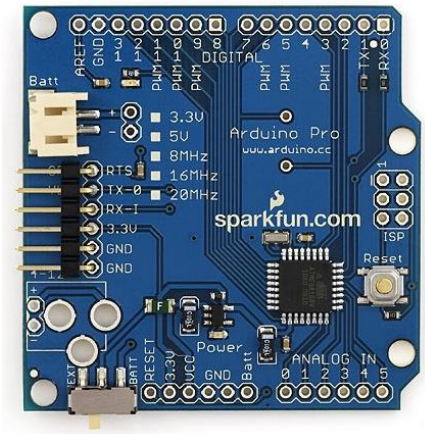
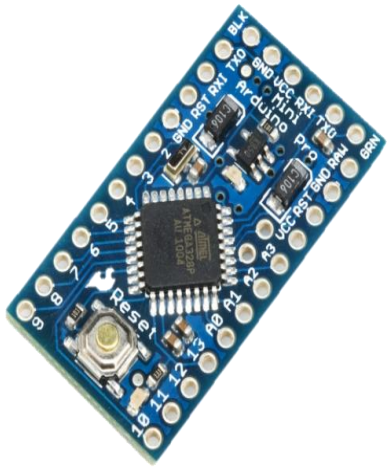
Рисунок 1. Процессор ATmega328

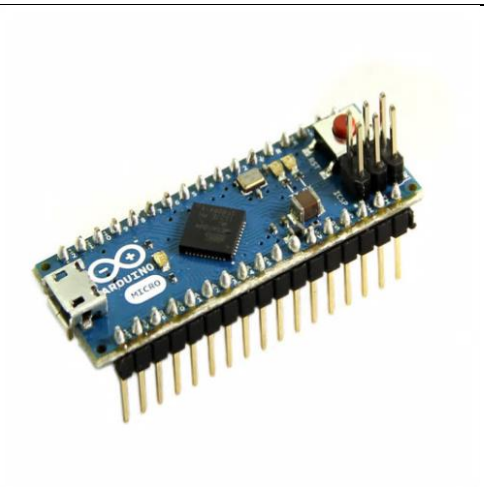
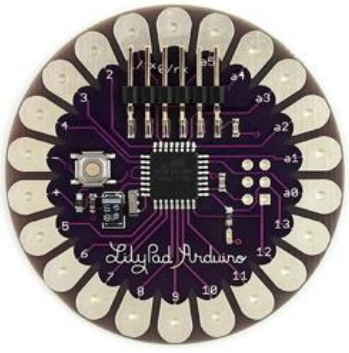
Существует много версий Arduino – Uno, Leonardo, Mini, Nano, Duemilanove, Mega, Diecimila, - основные характеристики некоторых устройств приведены в таблице 1.

Таблица 1. Внешний вид и основные характеристики некоторых устройств Arduino

	**Arduino UNO** Микроконтроллер на базе процессора ATmega328 . 14 цифровых входов/выходов, 6 из которых могут быть использованы как ШИМ выходы, 6 аналоговых входов, тактовая частота 16 МГц, подключение по шине USB, питание как от USB, так и внешнее.
	**Arduino DUE** Микроконтроллер на базе процессора Atmel SAM3X8E ARM Cortex-M3 CPU (32 бит). 54 цифровых входов/выходов, 12 из которых могут быть использованы как ШИМ выходы, 12 аналоговых входов, 4 последовательных порта (UART), тактовая частота 84 МГц.

	**Arduino Mega 2560** 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. Arduino Mega 2560 совместима со всеми платами расширения, разработанными для платформ Uno или Duemilanove.
	**Arduino Duemilanove** Плата на базе одного из микроконтроллеров: ATmega168 или ATmega328. Платформа содержит 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.
	**Arduino Diecimila** построена на микроконтроллере ATmega168. Платформа содержит 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.
	**Arduino Leonardo** Микроконтроллер на базе процессора ATmega32u4. 20 цифровых входов/выходов, 7 из которых могут быть использованы как ШИМ выходы, 12 аналоговых входов, тактовая частота 16 МГц, подключение по шине microUSB, питание как по USB, так и внешнее.

	**Arduino YUN** Микроконтроллер на базе процессора ATmega32u4. 20 цифровых входов/выходов, 7 из которых могут быть использованы как ШИМ выходы, 12 аналоговых входов, тактовая частота 16 МГц, + микропроцессор Atheros AR9331 с архитектурой MIPS @400 MHz, 64 Mb DDR2 ОЗУ, слот для карт памяти microSD, модуль Ethernet IEEE 802.3 10/100 Mbit/s, модуль WiFi IEEE 802.11 b/g/n
	**Arduino Pro** Построена на одном из микроконтроллеров: ATmega168 или ATmega328. Pro производится в обоих исполнениях 3.3 В / 8 МГц и 5 В / 16 МГц. Платформа содержит 14 цифровых входов и выходов (6 из которых могут использоваться как выходы ШИМ), 6 аналоговых входов, силовой разъем батареи, силовой выключатель, кнопку перезагрузки, отверстия для монтажа силового разъема, блок ICSP и блоки выводов. Шестипиновый блок может подключаться к кабелю FTDI или плате-конвертеру Sparkfun для обеспечения питания и связи через USB. Arduino Pro предназначена для непостоянной установки в объекты или экспонаты. Платформа поставляется без установленных выводов, что позволяет пользователям применять собственные выводы и разъемы. Расположение выводов совместимо с платами расширения Arduino
	**Arduino Mini** Построена на микроконтроллере ATmega168 и предназначена для использования в лабораторных работах и проектах, где пространство является критическим параметром. Платформа содержит 14 цифровых входов и выходов (6 из которых могут использоваться как выходы ШИМ), 8 аналоговых входов и кварцевый генератор 16 МГц. Программируется при помощи адаптера Mini USB или любого преобразователя USB или RS232 в TTL.

	**Arduino Micro** Микроконтроллер на базе ATmega32u4. Плата имеет 20 цифровых вход/выходов (из них 7 могут использоваться в качестве выходов ШИМ и 12 - как аналоговые входы), кварцевый генератор частотой 16 МГц, гнездо микро-USB, разъем ICSP и кнопка reset. Форм-фактор контроллера позволяет легко разместить его на макетной плате. Имеет встроенную поддержку USB-соединения, благодаря чему не требуется вспомогательный процессор. Это позволяет Micro появляться на подключенном компьютере в качестве мыши или клавиатуры в дополнение к виртуальному (CDC) последовательному порту (COM).
	**Arduino Nano** Платформа Nano, построенная 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.
	**Arduino LilyPad** Платформа Arduino LilyPad разработана с целью использования как часть одежды. Она может быть зашита в ткань со встроенными источниками питания, датчиками и приводами с проводкой. Платформа построена на микроконтроллере ATmega168V (маломощная версия с ATmega168)) или ATmega328V.

Так как каждая из разновидностей имеет свои существенные отличия, сосредоточимся на одной из них – Arduino UNO, наиболее подходящей для образовательного процесса ввиду оптимального сочетания функций.

ARDUINO UNO

Arduino UNO имеет 14 контактов цифрового ввода-вывода, которые могут быть как входами, так и выходами, что определяется скетчем. Также на плате находятся 6 контактов аналогового входа, ведущие к аналогово-цифровому преобразователю (АЦП). Присутствуют контакты питания, интерфейсы подключения к USB и RS-232 и прочие выводы.

Рисунок 2. Внешний вид Adruino UNO.

Плата может быть запитана от USB-порта компьютера, большинства USB-зарядных устройств, или от AC-адаптера с рекомендованным напряжением 9 вольт через стандартный разъём 2,1мм (плюс в центре). Если в разъём питания не подключён источник, плата получает питания от USB-разъёма.

Таблица 2. Характеристики Arduino UNO.

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 широтно-импульсной модуляции – 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

ВВОДЫ-ВЫВОДЫ ПЛАТЫ ARDUINO UNO

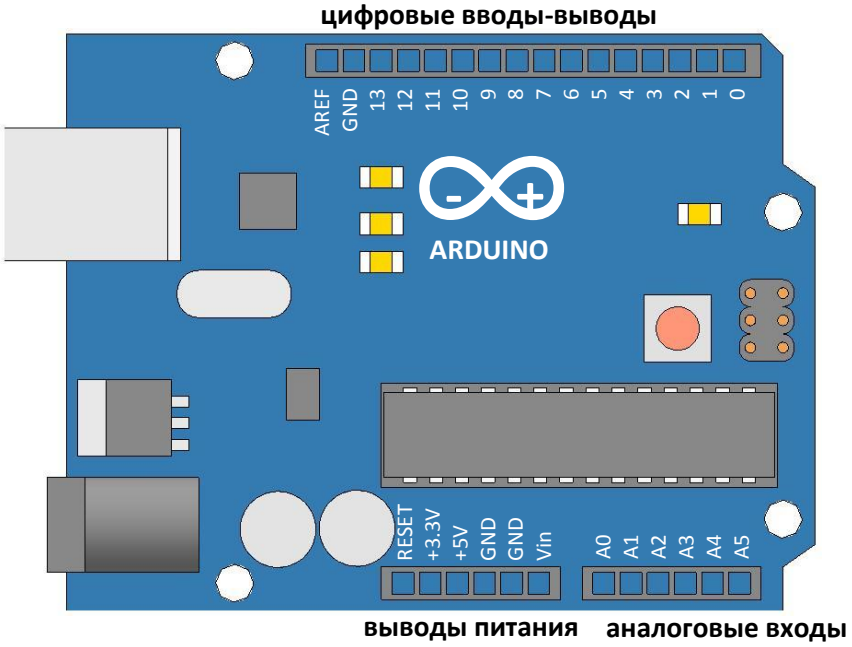


Рисунок 3. Маркировка выводов платы Arduino UNO

Выводы питания:

- **V_{IN}**. Вход используется для подачи питания от внешнего источника (в отсутствие 5 В от разъема USB или другого регулируемого источника питания). Подача напряжения питания происходит через данный вывод.
- **5V**. Регулируемый источник напряжения, используемый для питания микроконтроллера и компонентов на плате. Питание может подаваться от вывода VIN через регулятор напряжения, или от разъема USB, или другого регулируемого источника напряжения 5 В.
- **3.3V**. Напряжение на выводе 3.3 В генерируемое встроенным регулятором на плате. Максимальное потребление тока 50 мА.
- **GND**. Выводы заземления.

Цифровые вводы-выводы:

Каждый из 14 цифровых выводов Arduino Uno может быть настроен как вход или выход . Выводы работают при напряжении 5 В. Каждый вывод имеет нагрузочный резистор (по умолчанию отключен) 20-50 кОм и может пропускать до 40 мА. Некоторые выводы имеют особые функции:

- **Последовательная шина: 0 (RX) и 1 (TX)**. Выводы используются для получения (RX) и передачи (TX) данных через последовательный порт.

Данные выводы подключены к соответствующим выводам микросхемы ATmega (USB-to-TTL).

- **Внешнее прерывание: 2 и 3**. Данные выводы могут быть сконфигурированы на вызов прерывания либо на младшем значении, либо на переднем или заднем фронте, или при изменении значения.
- **ШИМ: 3, 5, 6, 9, 10, и 11.** Любой из выводов обеспечивает широтно-импульсную модуляцию с разрешением 8 бит.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** Посредством данных выводов осуществляется связь SPI.
- **LED: 13.** Встроенный светодиод, подключенный к цифровому выводу 13. Если значение на выводе имеет высокий потенциал, то светодиод горит.

!	**SPI** (*Serial Peripheral Interface)* — последовательный периферийный интерфейс — последовательный синхронный стандарт передачи данных в режиме полного дуплекса, предназначенный для обеспечения простого сопряжения микроконтроллеров и периферии. В отличие от стандартного последовательного порта SPI является синхронным интерфейсом, в котором любая передача синхронизирована с общим тактовым сигналом, генерируемым ведущим устройством (процессором). К одному последовательному периферийному интерфейсу ведущего устройства-микросхемы может присоединяться несколько микросхем.

!	**ШИМ** (PWM — pulse-width modulation) — управление средним значением напряжения на нагрузке путём изменения скважности импульсов

Широтно-импульсная модуляция, или ШИМ, это операция получения изменяющегося аналогового значения посредством цифровых устройств. Устройства – в нашем случае Arduino - используются для получения прямоугольных импульсов - сигнала, который постоянно переключается между максимальным и минимальным значениями. Данный сигнал моделирует напряжение между максимальным значением (5 В) и минимальным (0 В), изменяя при этом длительность времени включения 5 В относительно включения 0 В. Длительность включения максимального значения называется шириной импульса. Отношение периода следования

импульсов к их длительности называется скважностью. Величина, обратная скважности – коэффициент заполнения (duty cycle). Для получения различных аналоговых величин изменяется ширина импульса.

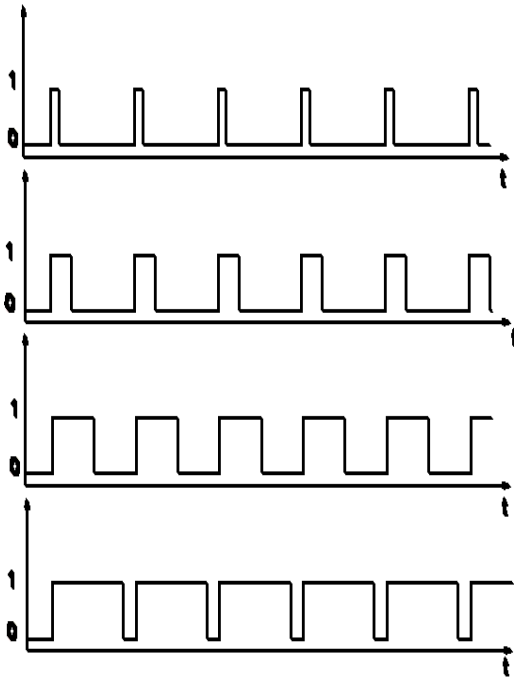


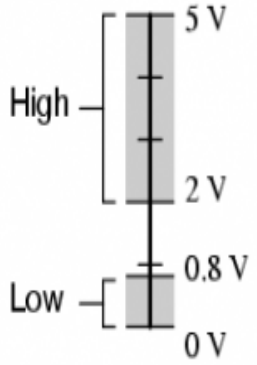
Рисунок 4. Пример широтно-импульсной модуляции сигнала

Управление ШИМ в Arduino производится путем записи значения 0..255 в цифровой вывод 3, 5, 6, 9, 10, 11. Значение «0» соответствует постоянному напряжению 0В на цифровом выводе, «255» - постоянному напряжению 5В. Значение «127» соответствует 50% заполнению, когда ширина импульса равна половине его периода.

Так, например, можно управлять яркостью свечения диода, подключенного непосредственно к выводу ШИМ.

Цифровые входы/выходы в своей работе используют общепринятые TTL-уровни сигналов, разграничивающие переключение из состояния LOW «0» в HIGH «1» и наоборот (рис.5).

TTL-уровни для входа:



TTL-уровни для выхода:

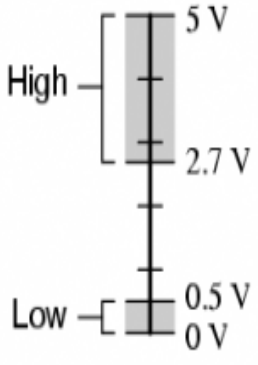


Рисунок 5. TTL-уровни для входа и выхода

Если к цифровому выводу ничего не подключено, то значения на нем могут принимать случайные величины, наводимые электрическими помехами или емкостной взаимосвязью с соседним выводом. Если на порт ввода не поступает сигнал, то в данном случае рекомендуется задать порту известное состояние. Это делается добавлением подтягивающих резисторов 10 кОм, подключающих вход либо к +5 В (подтягивающие к питанию резисторы), либо к земле (подтягивающие к земле резисторы).

Микроконтроллер Atmega имеет программируемые встроенные подтягивающие к питанию резисторы 20 кОм, поключающиеся программно.

Выводы, сконфигурированные как порты вывода, находятся в низкоимпедансном состоянии. Данные выводы могут пропускать через себя достаточно большой ток. Выводы микросхемы Atmega могут быть источником (положительный) или приемником (отрицательный) тока до 40 мА для других устройств. Такого значения тока достаточно чтобы подключить светодиод (обязателен последовательно включенный резистор), датчики, но недостаточно для большинства реле, соленоидов и двигателей.

Короткие замыкания выводов Arduino или попытки подключить энергоемкие устройства могут повредить выходные транзисторы вывода или весь микроконтроллер Atmega. В большинстве случаев данные действия приведут к отключению вывода на микроконтроллере, но остальная часть схемы будет работать согласно программе. Рекомендуется к выходам платформы подключать устройства через резисторы 470 Ом или 1 кОм, если устройству не требуется больший ток для работы.

Аналоговые входы:

На платформе Uno установлены 6 аналоговых входов (обозначенных как A0 .. A5), каждый разрешением 10 бит (т.е. может принимать 1024 различных значения – от 0 до 1023). Стандартно выводы имеют диапазон измерения до 5 В относительно земли. Некоторые выводы имеют дополнительные функции:

- **I2C: 4 (SDA) и 5 (SCL).** Посредством выводов осуществляется связь I2C (TWI).

Выводы Arduino, соответствующие аналоговым входам, имеют номера от 14 до 19. Это относится только к выводам Arduino, а не к физическим номерам выводов микроконтроллера Atmega. Аналоговые входы могут использоваться как цифровые выводы портов ввода/вывода: они имеют подтягивающие резисторы, включающиеся программно.

!	**I2C** – последовательная шина данных для связи интегральных схем, использующая две двунаправленные линии связи (SDA и SCL). Используется для соединения низкоскоростных устройств. Название представляет собой аббревиатуру слов **Inter-Integrated Circuit**. Данные передаются по двум проводам — проводу данных и проводу тактов.

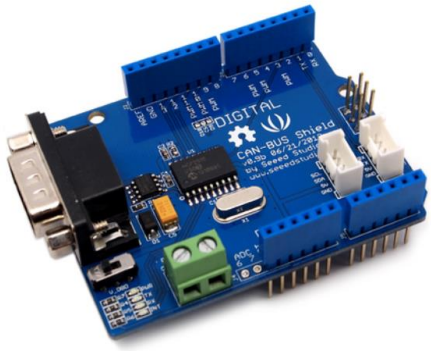
Дополнительные выводы платформы:

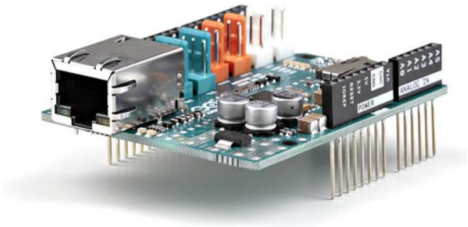
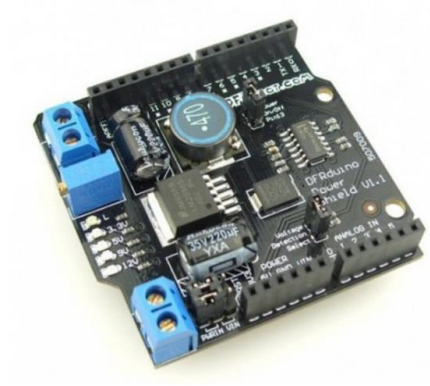
- **AREF.** Опорное напряжение для аналоговых входов.
- **Reset.** Низкий уровень сигнала на выводе перезагружает микроконтроллер. Обычно применяется для подключения кнопки перезагрузки на плате расширения, закрывающей доступ к кнопке на самой плате Arduino.

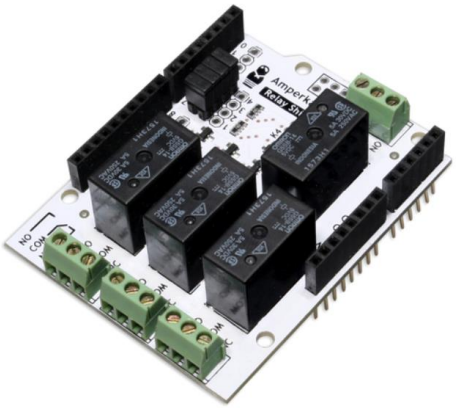
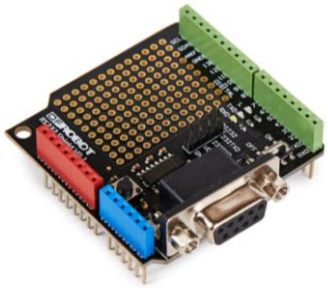
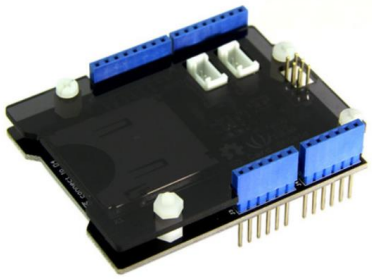
ПЛАТЫ РАСШИРЕНИЯ, ДАТЧИКИ И ИСПОЛНИТЕЛЬНЫЕ УСТРОЙСТВА

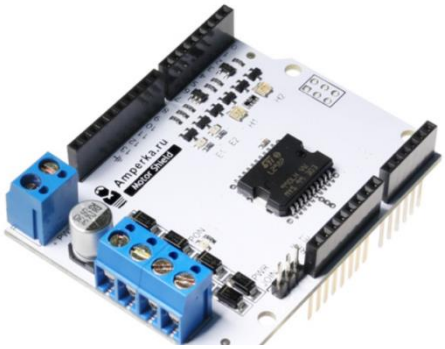
Для Arduino выпускается большое количество разнообразных плат расширения, позволяющих нарастить функционал в соответствии с поставленной задачей (табл.3). Платы расширения чаще всего устанавливаются «вторым этажом» над платой Arduino в случае совпадения форм-фактора и разъемов. В продаже так же есть «пустые» платы форм-фактора Arduino для прототипирования собственных плат расширения.

Таблица 3. Платы расширения Arduino

	**CAN-BUS Shield** Плата передачи данных по пром.стандарту шины CAN • Поддержка CAN версии 2.0B на скорости до 1 Мб/с • Управление по интерфейсу SPI на частоте до 10 МГц • Поддержка стандартного (11-битного) и расширенного (29-битного) протоколов передачи данных • 2 буфера приёма данных • Стандартный разъём D-sub на 9 выводов • Индикаторы приёма и передачи данных

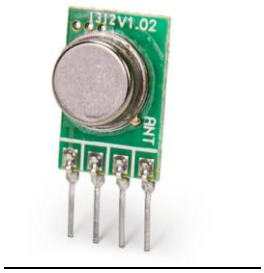
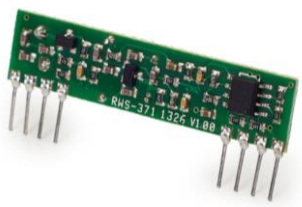
	**Ethernet Shield** Плата подключения к сети Ethernet • Работает на базе чипа Wiznet W5 500 • Поддержка TCP и UDP протоколов • Рабочее напряжение 5 В • Размер буфера 32 Кб • Скорость соединения 10/100 Мбит.
	**GPRS Shield** Плата для передачи данных по GSM сетям • Количество полос — 2: 900, 1800 МГц • Класс GSM — 2+ • Класс GPRS — «B»: одновременно работает либо GPRS (данные), либо GSM (SMS, голос). На время работы GSM, активность GPRS приостанавливается • Класс мультислотов — 8/10: до 85 Кбит/с на скачивание и до 42 Кбит/с на загрузку • Выходная мощность модуля — Класс 4 (2 Вт) на 900 МГц. Класс 1 (1 Вт) на 1800 • Потребление тока в спящем режиме: 1 мА
	**Multiservo Shield** Плата одновременного управления сервоприводами • Максимальный постоянный ток на сервоприводы: 10 А • Интерфейс управления: I²C • Количество сервоприводов: 18 по I²C, 6 через Arduino • Потребляемый микроконтроллером ток: 15 мА • Диапазон рабочих температур: −40…+85 °C
	**Power Shield** Плата управления двумя раздельными контурами питания • Входное напряжение: 4,5 – 35 В • Выходное напряжение: 1,25 – 12 В (не выше входного) • Максимальный ток на выходе: 2 А (3 А в пике) • Максимальная мощность: 15 Вт (3 А при 5 В) • КПД: 90%

	**Relay Shield** Плата коммутации устройств, подключенных к сети 220В • Ток обмотки: 80 мА • Максимальное коммутируемое напряжение: 30 В постоянного тока; 250 В переменного тока • Максимальный коммутируемый ток: 5 А (NO), 3 А (NC) • Рекомендованная частота переключения: до 1 Гц • Время жизни: не менее 50 000 переключений
	**RS-232 Shield** Плата передачи данных через интерфейс RS-232 • Максимальное расстояние связи: 15 м • Напряжение питания: 5 В • Стандартный разъем DB9(F) • Связь с Arduino через пины 0 (RX) и 1 (TX)
	SD Card Shield Плата записи и чтения информации с SD, SDHC и microSD карт • Объём карт памяти - до 16 Гб • Поддержка файловых систем FAT16 или FAT32
	**Wi-Fi Shield** Плата беспроводного соединения по стандарту 802.11 b/g (Wi-Fi) для общения с другими устройствами или выхода в интернет • Поддерживается шифрование WEP и WPA2 • Возможность использовать протоколы TCP и UDP • Есть слот для флеш-карт microSD объёмом до 2 Гб

	**Motor Shield** Плата управления двигателями постоянного тока и шаговыми двигателями • На базе чипа L298P • 2 независимых канала • Позволяет управлять моторами с напряжением 5–24 В в режиме раздельного питания и 7–12 В в режиме объединённого питания • Можно подключить на выбор: пару маломощных двигателей постоянного тока (DC-моторов), один биполярный шаговый двигатель, один DC-мотор с током до 4 А - если объединить каналы

Датчики, исполнительные устройства, и устройства ввода-вывода, разработанные специально для Arduino, представлены большим количеством вариантов. Некоторые, наиболее популярные у разработчиков АСУ на основе Arduino, приведены в таблицах 4-6:

Таблица 4. Датчики для платформы Arduino

	**Датчик температуры** Модуль выполнен на основе микросхемы TMP36. Он работает в диапазоне температур от −40 °C до +125 °C. Выходным результатом работы сенсора является аналоговый сигнал с уровнем напряжения прямо пропорциональным температуре, с шагом 10 мВ/°C. Точность: ±1 °C при температуре 25 °C, ±3 °C на всём диапазоне измерения.
	**Беспроводной передатчик на 433 МГц** Модуль выполняет функцию только передатчика. Для приёма его сигнала существует модуль-приёмник на 433 МГц. Модуль имеет всего 4 контакта: питание, земля, цифровой вход и антенна. Модуль просто передаёт восходящие и нисходящие фронты, поступающие на вывод «Data in». Это позволяет подключать к модулю напрямую даже такие простые источники сигнала, как кнопка или геркон.
	**Беспроводной приемник на 433 МГц** Модуль просто принимает переданные модулем-передатчиком фронты, и выдаёт их на ножку «Data out». Это позволяет подключать к модулю напрямую даже такие простые элементы, как светодиод, пьезодинамик или реле. Нужно лишь только усилить сигнал с помощью простого транзистора.

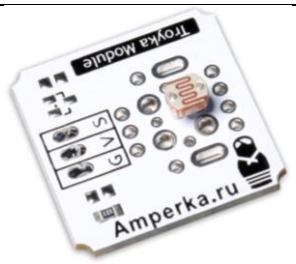
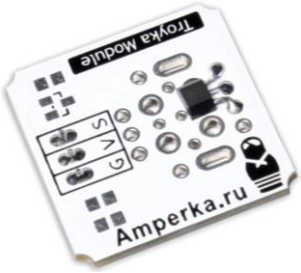
	**Датчик наклона** Датчик наклона — это капсула с металлическим шариком внутри. Шарик перекатывается в капсуле и замыкает или размыкает цепь. Таким образом датчик выдаёт простой цифровой сигнал: логический ноль или единицу в зависимости от того, в какую сторону наклонена капсула.
	**Датчик освещенности** Выходным результатом работы сенсора является аналоговый сигнал. Выходное напряжение датчика обратно пропорционально интенсивности падающего света. Датчик подключается к управляющей электронике через 3 провода.
	**Датчик Холла** Датчик магнитного поля. Датчик Холла выполнен на основе микросхемы SS49E. Выходным результатом работы сенсора является аналоговый сигнал. В отсутствии магнитного поля датчик выдаёт половину напряжения питания. При появлении магнитного поля значение отклоняется к нулю или напряжению питания в зависимости от полярности магнитного поля и пропорционально его интенсивности.
	**Датчик шума** Микрофон с предусилителем. Сенсор выдаёт аналоговый сигнал в диапазоне 0–5 В. Выходное напряжение пропорционально средней шумности за последние несколько сотен миллисекунд. Для регулировки чувствительности на модуле предусмотрен триммер.
	**ИК-приемник** Средство для дистанционного управления Arduino при помощи обычного пульта дистанционного управления от бытовой техники. Если использовать этот приёмник вместе с инфракрасным светодиодом, то можно организовать беспроводную двустороннюю полудуплексную связь между двумя устройствами через UART. В модуле с ИК-приёмником используется микросхема TSOP22. Выходным результатом работы сенсора является цифровой сигнал.

Таблица 5. Исполнительные устройства для платформы Arduino

	**Лазерный модуль** Лазерный модуль HLM1230 — простой источник яркого точечного света. Длина волны: 650 нм. Рабочая температура: −15…+35 °C. Размер пятна: менее 2 мм на расстоянии 3 м
	**Двигатель постоянного тока AMP-F010 с редуктором** Рабочее напряжение: 3 – 9 В, номинальное напряжение: 6 В Ток без нагрузки: 80 мА, ток при блокировке: 1600 мА Диаметр выходного вала: 3 мм, длина вала: 10 мм Ширина × Высота: 12×10 мм Материал шестерней: металл
	**Шаговый биполярный двигатель 28STH45-0674A** Шаг: 1,8°±5% (200 на оборот) Номинальное напряжение питания: 4,4 В, номинальный ток фазы: 670 мА Крутящий момент: не менее 0,9 кг×см Максимальная скорость старта: 1000 шагов/сек Диаметр вала: 5 мм, длина вала: 20 мм Габариты корпуса: 28×28×45 мм

Таблица 6. Устройства ввода/вывода для платформы Arduino

	**Зуммер** Пьезодинамик, воспроизводящий звук.
	**Кнопка** Кнопка с уменьшенным дребезгом контактов для организации управляющих сигналов.

	**Потенциометр** Выходным результатом работы сенсора является аналоговый сигнал, с уровнем напряжения прямо пропорциональным углу поворота ручки потенциометра.
	**Светодиод** Светодиод является источником широкополосного излучения с известной центральной длиной волны. Бывают диоды с несколькими излучателями, позволяющие генерировать излучение одновременно или попеременно на разных длинах волн.
	**Сегментный индикатор** Позволяет генерировать набор знаков для отображения цифр, некоторых букв и элементов псевдографики.
	**Матричный индикатор** Обладает высокой информативностью (количество и качество отображаемой информации зависит только от размеров матрицы и размеров одного пикселя). Если нет встроенного, то требуется внешний энкодер.

ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ (IDE)

IDE (интегрированная среда разработки) - это специальная программа, работающая на персональном копьютере, которая позволяет писать скетчи для платы Arduino на простом языке подбном C++ и C#. Когда происходит выгрузка скетча на плату, написанный код будет транслирован в язык C, и будет передан компилятору avr-gcc, который и произведёт финальную трансляцию в язык, понятный микроконтроллеру. Последний шаг очень важен, так как Arduino упрощает процесс создания управляющих программ, скрывая все возможные сложности программирования микроконтроллеров.

Цикл программирования Arduino упрощённо выглядит так:

- Подключить плату к USB-порту персонального компьютера.
- Написать скетч (программу)
- Выгрузить скетч на плату через USB-соединение и подождать несколько секунд для перезапуска платы
- Плата выполнит написанный скетч.

Среда разработки Arduino состоит из встроенного текстового редактора программного кода, области сообщений, окна вывода текста(консоли), панели инструментов с кнопками часто используемых команд и нескольких меню. Для загрузки программ и связи среда разработки подключается к аппаратной части Arduino.

Скетч пишется в текстовом редакторе, имеющем инструменты вырезки/вставки, поиска/замены текста. Во время сохранения и экспорта проекта в области сообщений появляются пояснения, также могут отображаться возникшие ошибки. Окно вывода текста(консоль) показывает сообщения Arduino, включающие полные отчеты об ошибках и другую информацию. Кнопки панели инструментов позволяют проверить и записать программу, создать, открыть и сохранить скетч, открыть мониторинг последовательной шины.

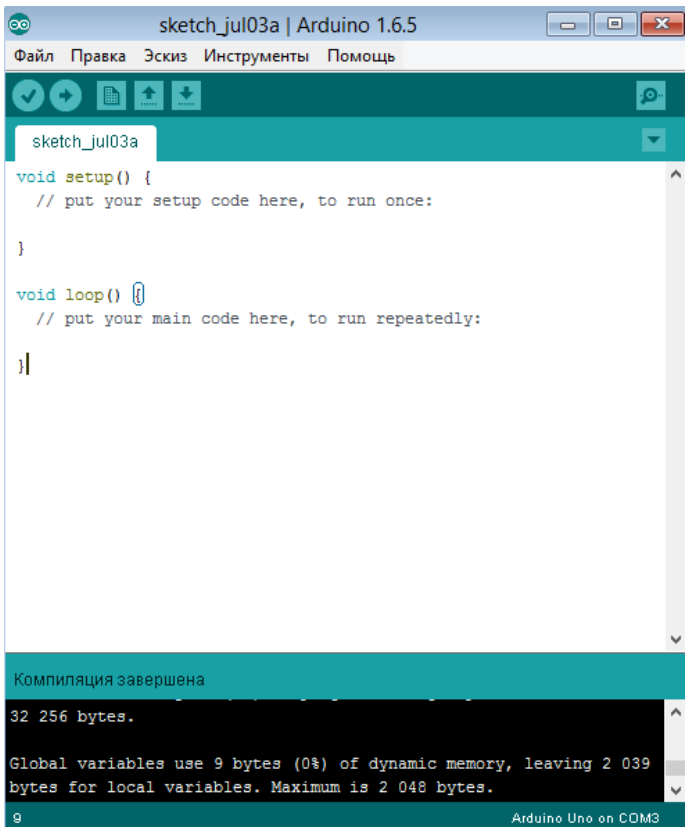


Рисунок 6. Внешний вид интегрированной среды разработки Arduino

Интерфейс IDE

Основные элементы управления IDE Arduino вынесены в панель инструментов, располагающуюся в верхней части окна IDE ниже строки меню (твбл.7).

Таблица 7. Элементы панели инструментов IDE Arduino

 *Verify/Compile (Эскиз-> Проверить/Скомпилировать)*	Проверка программного кода на ошибки, компиляция.
 *New (Файл->Создать)*	Создание нового скетча.
 *Open (Файл->Открыть)*	Открытие меню доступа ко всем скетчам в проводнике.
 *Save (Файл->Сохранить)*	Сохранение скетча.
 *Upload to I/O Board (Эскиз-> Выгрузить)*	Компилирует программный код и загружает его в устройство Arduino.
 *Serial Monitor (Инструменты -> Монитор последовательного порта)*	Открытие мониторинга последовательного порта (Serial monitor).

Дополнительные команды сгруппированы в пять выпадающих меню: File (Файл), Edit (Правка), Sketch (Эскиз), Tools (Инструменты), Help (Помощь). Доступность меню определяется работой, выполняемой в данный момент.

Edit (Правка)

- *Copy for Discourse (Копировать для форума)*
  Копирует в буфер обмена подходящий для размещения на форуме код скетча с выделением синтаксиса.

- *Copy as HTML (Копировать в HTML)*
  Копирует код скетча в буфер обмена как HTML код, для размещения на веб-страницах.

Sketch (Эскиз)

- *Verify/Compile (Проверить/скомпилировать)*
  Проверка скетча на ошибки.

- *Import Library(Импортировать библиотеку)*
  Добавляет библиотеку в текущий скетч, вставляя директиву #include в код скетча.

- *Show Sketch Folder (Показать папку эскиза)*
  Открывает папку, содержащую файл скетча, на рабочем столе.

- *Add File...(Добавить файл)*
  Добавляет файл в скетч (файл будет скопирован из текущего места расположения). Новый файл появляется в новой закладке в окне скетча. Файл может быть удален из скетча при помощи меню закладок.

Tools (Инструменты)

- *Auto Format(Автоформатирование)*
  Данная опция оптимизирует код, например, выстраивает в одну линию по вертикали открывающую и закрывающую скобки и помещает между ними утверждение.

- *Board (Плата)*
  Выбор используемой платформы.

- Serial Port (Порт)
  Меню содержит список последовательных устройств передачи данных (реальных и виртуальных) на компьютере. Список обновляется автоматически каждый раз при открытии меню Tools.

- *Burn Bootloader (Записать загрузчик)*
  Пункты данного меню позволяют записать Загрузчик (Bootloader) в микроконтроллер на платформе Arduino. Данное действие не требуется в текущей работе с Arduino, но пригодится, если имеется новый ATmega (без загрузчика). Перед записью рекомендуется проверить правильность выбора платформы из меню. При использовании AVR ISP необходимо выбрать соответствующий программатору порт из меню Serial Port.

Средой Arduino используется принцип блокнота: стандартное место для хранения программ (скетчей). Скетчи из блокнота открываются через меню File > Sketchbook (Файл->Папка со скетчами) или кнопкой Open (Открыть) на панели инструментов. При первом запуске программы Arduino автоматически создается директория для блокнота. Расположение блокнота меняется через диалоговое окно Preferences (Файл->Настройки).

Среда позволяет работать с несколькими файлами скетчей (каждый открывается в отдельной закладке). Файлы кода могут быть стандартными Arduino (без расширения), файлами C (расширение *.c), файлами C++ (*.cpp) или файлами заголовков (.h).

Перед работой с платой Arduino требуется задать необходимые параметры в меню **Tools > Board** (Инструменты-> Плата) и **Tools > Serial Port** (Инструменты -> Порт), т.е. выбрать из списка используемую плату и порт, к которому она подключена**.** В Windows последовательные порты обозначаются как COM1…COM N.

Монитор последовательного порта (Serial Monitor) отображает данные посылаемые в/из платформы Arduino. Для отправки данных необходимо ввести текст и нажать кнопку Send или Enter. Затем выбирается скорость передачи из выпадающего списка, соответствующая значению Serial.begin в скетче.

ПРОГРАММИРОВАНИЕ ARDUINO

Переменные

Переменная – это место хранения данных. Она имеет имя, значение и тип. В скетче Arduino имена переменных могут состоять только из латинских литер, цифр и знака подчеркивания. Имя переменной не может начинаться с цифры. Переменные могут быть глобальными и локальными. Глобальные переменные объявляются в теле скетча, их значения доступны для всех конструкций скетча. Локальные переменные создаются внутри какой-либо конструкции скетча, например, функции, и из значения доступны только внутри данной конструкции. Значение переменной зависит от ее типа, т.е. способности хранить тот или иной тип данных.

Типы данных

Переменные и функции скетча Arduino могут принимать один из перечисленных в таблице 8 типов данных. Указание типа при объявлении переменной или функции является обязательным. При обращении к существующим переменным и функциям указывать их тип не требуется.

Таблица 8. Типы данных

Тип	**Хранит**	**Диапазон значений**	**Размер, байт**
boolean	логический (булевый) тип данных	true или false	1
char	алфавитно-цифровой символ (литера). При объявление литеры используются одиночные кавычки: 'A'.	литера	1
byte	8-ми битное беззнаковое целое число,	0..255	1
int	знаковое целое число от -2^{15} до $2^{15}-1$	-32 768.. 32 767	2
unsigned int	беззнаковое целое число. В отличие от int, тип unsigned int может хранить только положительные целые числа в диапазоне от 0 до $(2^{16}-1)$	0.. 65535	2
word	16-битное, не содержащее знака, число от 0 до 65535. Тоже самое, что unsigned int	0.. 65535	2
long	целое число в расширенном диапазоне от - 2,147,483,648 до 2,147,483,647.	- 2,147,483,648.. 2,147,483,647	4
unsigned long	беззнаковое целое число в диапазоне от 0 до $(2^{32} - 1)$	0.. 4,294,967,295	4
float	число с плавающей запятой. Этот тип часто используется для операций с данными, считываемыми с аналоговых входов.	-3.4028235E+38 ..3.4028235E+38	4
double	число с плавающей запятой	-3.4028235E+38 ..3.4028235E+38	4
void	ничего. Используется при объявлении функций, если функция не возвращает никакого значение при ее вызове	-	-

Функции

Функции позволяют разбивать код на сегменты, которые выполняют определенные задания. После выполнения происходит возврат в место, откуда была вызвана функция. Причиной создания функции является необходимость выполнять одинаковое действие несколько раз.

Синтаксис функции можно рассмотреть на следующем примере:

`int myFunction(int a, int b)`
`{`
`  return (a+b);`
`}`
	Функция возвращает целочисленное значение типа int. Имя функции – myFunction. В функцию передаются два целочисленных параметра a и b.
	Тело функции заключено в фигурные скобки {}. Оператор return позволяет вернуть результат выполнения функции.

Пример вызова функции myFunction:

`int c=myFunction(1,2);`	В целочисленную переменную *c* будет записан результат выполнения функции myFunction, т.е. число 3.

Если функция не должна возвращать никаких значений, она объявляется как void:

`void myFunction(int a, int b)`
`{`
`  a=b;`
`}`
	Функция не возвращает никакого значения, все результаты пропадают после завершения работы функции, если не были задействованы глобальные переменные. Оператор возврата return отсутствует.

Скетч Arduino должен по умолчанию содержать две основные функции, реализующие взаимодействие программы с мкроконтроллером:

Функция setup():

Функция setup() вызывается, когда стартует скетч. Используется для инициализации переменных, определения режимов работы выводов, запуска используемых библиотек и т.д. Функция setup запускает только один раз, после каждой подачи питания или сброса платы Arduino.

Функция loop():

После вызова функции setup(), которая инициализирует и устанавливает первоначальные значения, функция loop() повторяется в цикле, позволяя программе совершать вычисления и реагировать на них. Данную функцию необходимо использовать для активного управления платой Arduino

Данные функции имеют пустой тип void, так как не возвращают никаких значений.

Структура скетча Arduino

Скетч Arduino имеет следующую простую структуру:

- Объявление глобальных переменных;
- Функция setup();
- Функция loop();
- Пользовательские функции().

Порядок размещения функций в скетче не влияет на очередность их выполнения. Первой всегда вызывается функция setup(), второй – функция loop(). Пользовательские функции могут быть вызваны из этих двух перечисленных функций.

`int c=0;`	Объявление глобальной переменной *c*.

`void setup()`
`{`
`}`
	Вызывается пустая функция setup.

`void loop()`
`{`
`c=myFunction(1,2);`
`}`
	Вызывается функция loop, в которой в бесконечном цикле переменной *c* присваивается результат выполнения пользовательской функции myFunction.

`int myFunction(int a, int b)`
`{`
`  return (a+b);`
`}`
	Объявление и описание пользовательской функции myFunction.

Управляющие операторы

Управляющие операторы, перечисленные в таблице 9, во многом совпадают с операторами языка С.

Таблица 9. Управляющие операторы

<table>
<tr><td>if</td><td>оператор сравнения, - проверяет, достигнута ли истинность условия. Например:

<table><tr><td><pre>if (a > b)
{
c=a;
}</pre></td><td>Если значение переменной a больше значения переменной b, в переменную c записывается значение переменной a.</td></tr></table>
То есть, если выражение в круглых скобках истинно, выполняются операторы внутри фигурных скобок. Если нет, программа пропускает этот код.</td></tr>
<tr><td>if..else</td><td>оператор сравнения способный осуществлять несколько проверок, объединенных вместе. Например:

<table><tr><td><pre>if (a > b)
{
c=a;
}
else
{
c=b;
}</pre></td><td>Если значение переменной a больше значения переменной b, в целочисленную переменную c записывается значение переменной a. Иначе в переменную c записывается значение переменной b.</td></tr></table></td></tr>
<tr><td>for</td><td>оператор повторения части кода, заключенной в фигурные скобки – т.н. «цикл». Количество повторений реализуется через счетчик – специальную переменную. Заголовок цикла for состоит из трех частей:

for (счетчик; условие; приращение)
{
операторы
}

здесь «счетчик» - инициализация переменной, выполняющей роль счетчика, «условие» - проверка счетчика на достижение определенного значения, «приращение» - на сколько изменяется значение счетчика за один цикл.
Каждый раз в цикле проверяется условие, и если оно верно, выполняется блок операторов и приращение, затем условие проверяется вновь. Когда логическое значение условия становится ложным, цикл завершается.</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>

```
for (int i=0;i<=2;i++)
{
c=c+1;
}
```

</td><td>В качестве счетчика инициализируем переменную i с начальным значением 0. В соответствии с условием цикл будет продолжаться пока счетчик не достигнет значения 2. Приращение равно единице.</td></tr>
</table>
</td></tr>
<tr><td>switch..case</td><td>оператор сравнения способный осуществлять несколько проверок, объединенных вместе. Синтаксис:

switch(переменная)
{
case значение1:
операторы;
break;
…
case значениеN:
операторы;
break;
default:
операторы;
}

Оператор switch сравнивает значение переменной со значением, определенном в операторах case. Когда найден оператор case, значение которого равно значению переменной, выполняется программный код в этом операторе. Ключевое слово break является командой выхода из оператора case и обычно используется в конце каждого case. Без оператора break оператор switch будет продолжать вычислять следующие выражения, пока не достигнет break или конец оператора switch. Код после оператора default выполняется, если не выбрана ни одна альтернатива. Оператор default является необязательным. Пример:
<table>
<tr><td>

```
switch(a){
case 1:
     b=10;
     break;
case 2:
     b=100;
     break;
default:
     b=0;
}
```

</td><td>Если значение переменной a равно 1, присваиваем переменной b значение 10 и прекращаем процедуру сравнения.
Если значение переменной a равно 2, присваиваем переменной b значение 100 и прекращаем процедуру сравнения.
Если значение переменной a не равно ни 1 ни 2, присваиваем переменной b значение 0.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td>while</td>
<td>Оператор выполнения куска кода в цикле непрерывно и бесконечно до тех пор, пока выражение в круглых скобках, () не станет равно логическому ЛОЖНО. Что-то должно изменять значение проверяемой переменной, иначе выход из цикла while никогда не будет достигнут. Синтаксис:

while(выражение)
{
операторы;
}

Пример:
<table>
<tr>
<td><pre>int a=1;

while(a<10)
{
      b=b+a;
      a++;
}</pre></td>
<td>Инициализируем переменную a с начальным значением 1.
Пока значение переменной a меньше 10, прибавляем к значению переменной b значение переменной a. Увеличиваем значение переменной a на единицу.</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>do..while</td>
<td>оператор do..while работает так же, как и цикл while, за исключением того, что условие проверяется в конце цикла, таким образом, цикл do будет всегда выполняться хотя бы раз.
Синтаксис:

do
{
операторы;
} while (условие);

Пример:
<table>
<tr>
<td><pre>do
{
      c=c+1;
} while (c<100);</pre></td>
<td>Прибавляем к переменной c единицу до тех пор, пока не достигнем значения 100.</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>break</td>
<td>оператор для принудительного выхода из циклов do, for или while, не дожидаясь завершения цикла по условию. Пример:
<table>
<tr>
<td><pre>while(1)
{
c=c+1;
if (c>10){ break; }
}</pre></td>
<td>В бесконечном цикле while увеличиваем значение переменной c на единицу. Когда значение переменной c превысит 10, цикл прервется.</td>
</tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td>continue</td><td>оператор continue пропускает оставшиеся операторы в текущем шаге цикла. Вместо них выполняется проверка условного выражения цикла, которая происходит при каждой следующей итерации. Пример:

<table>
<tr><td><pre>
for (int i=1;i<=10;i++)
{
if(i>=5 && x<=7){
continue;
}
c=c+1;
}
</pre></td><td>В цикле, который должен исполнить код (приращение значение переменной c на единицу) 10 раз, пропускаем с 5 по 7-ю итерацию.</td></tr>
</table>
</td></tr>
<tr><td>return</td><td>оператор прекращает вычисления в функции и возвращает значение из прерванной функции в вызывающую, если это нужно. Синтаксис:

return; или return значение;

Пример:

<table>
<tr><td><pre>
int f(int a, int b)
{
if (a > b)
{
      return a;
}
else
{
      return b;
}
}
</pre></td><td>Если значение переменной a больше значения переменной b, функция вернет значение переменной a, иначе – значение переменной b.</td></tr>
</table>
</td></tr>
</table>

Математические функции

В языке программирования Arduino существует стандартный набор математических функций, позволяющих осуществлять бо́льшую часть элементарных вычислений (таблица 10). Некоторые функции приспособлены для работы с АЦП и ШИМ.

Таблица 10. Математические функции.

min (x,y)	возвращает наименьшее из двух значений
max (x,y)	возвращает наибольшее из двух значений
abs(x)	возвращает модуль числа
constrain(x,a,b)	проверяет попадание числа *x* в диапазон [a,b]. Возвращает: **x**: если **x** входит в область допустимых значений [**a**..**b**] **a**: если **x** меньше **a** **b**: если **x** больше **b**
map(value, fromLow, fromHigh, toLow, toHigh)	пропорционально переносит значение (**value**) из текущего диапазона значений (**fromLow** .. **fromHigh**) в новый диапазон (**toLow** .. **toHigh**), заданный параметрами. Возвращает значение в новом диапазоне.
pow(x,y)	вычисляет и возвращает значение *x* возведенное в заданную степень *y*. **pow()** может возводить в дробную степень
sq(x)	возвращает квадрат числа, заданного параметром
sqrt(x)	вычисляет и возвращает квадратный корень числа, заданного параметром
sin(rad)	возвращает синус угла, заданного в радианах. Результат функции всегда в диапазоне -1..1.
cos(rad)	возвращает косинус угла, заданного в радианах. Результат функции всегда в диапазоне -1..1.
tan(rad)	возвращает тангенс угла, заданного в радианах.
randomSeed(seed)	инициализирует генератор псевдослучайных чисел. Генерируемая последовательность случайных чисел очень длинная, и всегда одна и та же. Точка в этой последовательности, с которой начинается генерация чисел, зависит от параметра *seed*.
random(min,max)	возвращает псевдослучайное число находящееся в диапазоне [min, max).

Цифровой ввод/вывод

Функции цифрового ввода-вывода (работают для цифровых входов/выходов платы Arduino D1-D13.

Функция pinMode

Описание	Устанавливает режим работы заданного вход/выхода(pin) как входа или как выхода.
Синтаксис	pinMode(pin, mode)
Параметры	pin: номер входа/выхода(pin), который Вы хотите установить mode: режим одно из двух значение - INPUT или OUTPUT, устанавливает на вход или выход соответственно.
Возвращаемое значение	нет

Функция digitalWrite

Описание	Подает HIGH («1») или LOW («0») значение на цифровой вход/выход (pin). Если вход/выход (pin) был установлен в режим выход (OUTPUT) функцией pinMode(), то для значение HIGH напряжение на соответствующем вход/выходе (pin) будет 5В (3.3В для 3.3V плат), и 0В(земля) для LOW. Если вход/выход (pin) был установлен в режим вход (INPUT), то функция digitalWrite со значением HIGH будет активировать внутренний 20K нагрузочный резистор. Подача LOW в свою очередь отключает этот резистор.
Синтаксис	digitalWrite(pin, value)
Параметры	pin: номер вход/выхода(pin) value: значение HIGH или LOW
Возвращаемое значение	нет

Функция digitalRead

Описание	Функция считывает значение с заданного входа - HIGH или LOW
Синтаксис	digitalRead(pin)
Параметры	pin: номер вход/выхода(pin) который Вы хотите считать
Возвращаемое значение	HIGH или LOW

Пример использования функций управления цифровыми вводами/выводами: скетч, управляющий работой светодиода, подключенного к выход/выходу 13 при помощи кнопки, подключенной к вход/выходу 7 (рис.7).

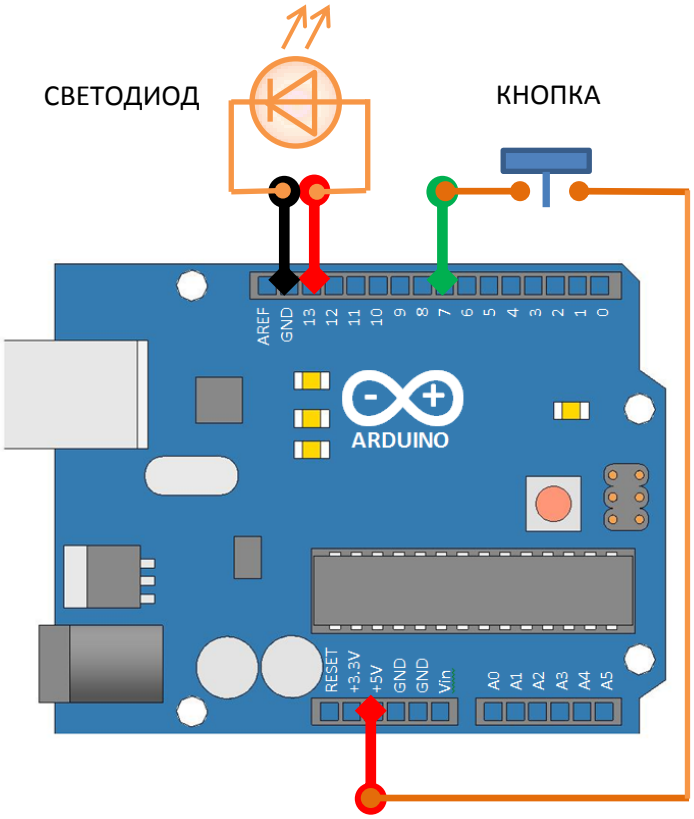


Рисунок 7. К объяснению использования функций управления цифровыми вводами/выводами

`int led = 13;`
`int button = 7;`
`int val = 0;`
 	
`void setup()`
`{`
`  pinMode(led, OUTPUT);`
`  pinMode(button, INPUT);`
`}`
 	
`void loop()`
`{`
`  val=digitalRead(button);`
`  digitalWrite (led, val);`
`}`
	Led - переменная, хранящая номер вывода, к которому подключен светодиод;
	button - переменная, хранящая номер вывода, к которому подключена кнопка;
	
	В процессе инициализации (функция setup) определяем D13 как выход, а D7 как вход.
	
	В бесконечном цикле функции loop считываем значение из D7 – либо HIGH, либо LOW.
	Сразу же передаем считанное значение на выход D13.

Как видно из скетча, пока кнопка будет нажатой, светодиод будет гореть, и потухнет при отпускании кнопки.

Аналоговый ввод/вывод

Функция analogRead

Описание	Функция считывает значение с указанного аналогового входа. Большинство плат Arduino имеют 6 каналов (8 каналов у платы Mini и Nano, 16 у Mega) с 10-битным аналого-цифровым преобразователем (АЦП). Напряжение поданное на аналоговый вход, обычно от 0 до 5 вольт будет преобразовано в значение от 0 до 1023, это 1024 шага с разрешением 0.0049 Вольт. Разброс напряжение и шаг может быть изменен функцией analogReference().Считывание значение с аналогового входа занимает примерно 100 микросекунд (0.0001 сек), т.е. максимальная частота считывания приблизительно 10,000 раз в секунду.
Синтаксис	analogRead(pin)
Параметры	pin: номер порта аналогового входа с которого будет производиться считывание (0..5 для большинства плат, 0..7 для Mini и Nano и 0..15 для Mega)
Возвращаемое значение	int (0 to 1023)

Функция analogWrite

Описание	Выдает аналоговую величину (ШИМ волну) на порт вход/выхода. Функция может быть полезна для управления яркостью подключенного светодиода или скоростью электродвигателя. После вызова analogWrite() на выходе будет генерироваться постоянная прямоугольная волна с заданной шириной импульса до следующего вызова analogWrite (или вызова digitalWrite или digitalRead на том же порту вход/выхода). Частота ШИМ сигнала приблизительно 490 Hz. На большинстве плат Arduino (на базе микроконтроллера ATmega168 или ATmega328) ШИМ поддерживают порты 3, 5, 6, 9, 10 и 11. Для вызова analogWrite() нет необходимости устанавливать тип вход/выхода функцией pinMode().
Синтаксис	analogWrite(pin, value)
Параметры	pin: порт вход/выхода на который подаем ШИМ сигнал. value: период рабочего цикла значение между 0 (полностью выключено) and 255 (сигнал подан постоянно).
Возвращаемое значение	нет

Функция analogReference

Описание	Функция определяет опорное напряжение относительно которого происходят аналоговые измерения. Функция analogRead() возвращает значение с разрешением 8 бит (1024) пропорционально входному напряжению на аналоговом входе и в зависимости от опорного напряжения. Возможные настройки: • DEFAULT: стандартное опорное напряжение 5 В (на платформах с напряжением питания 5 В) или 3.3 В (на платформах с напряжением питания 3.3 В) • INTERNAL: встроенное опорное напряжение 1.1 В на микроконтроллерах ATmega168 и ATmega328, и 2.56 В на ATmega8. • EXTERNAL: внешний источник опорного напряжения, подключенный к выводу AREF
Синтаксис	analogReference(type)
Параметры	type: определяет используемое опорное напряжение (DEFAULT, INTERNAL или EXTERNAL).
Возвращаемое значение	нет

Примером использования функций аналогового ввода-вывода может служить скетч, управляющий яркостью свечения светодиода при помощи значений напряжения, регистрируемых АЦП платы Arduino. Для этого можно соединить один из аналоговых входов Arduino с шиной питания +5V через подстроечное сопротивление, а к выходу ШИМ подключить светодиод (рисунок 8).

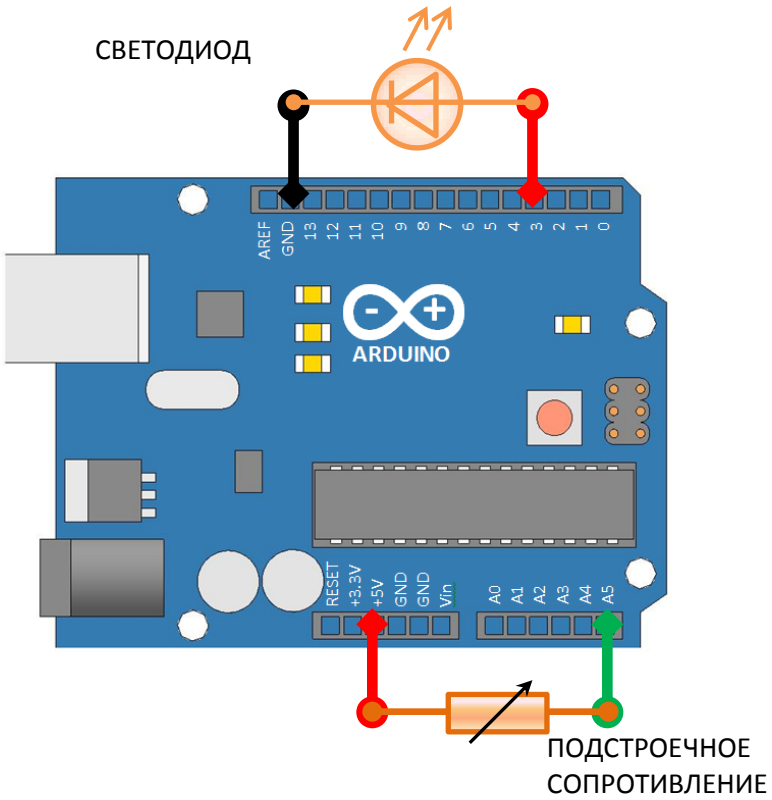


Рисунок 8. К объяснению использования функций управления аналоговыми вводами/выводами

```
int led = 3;
int control = 5;
int val = 0;

void setup()
{
  pinMode(led, OUTPUT);
}

void loop()
{
  val=analogRead(control);
  analogWrite(led, int(val/4));
}
```

Led - переменная, хранящая номер вывода, к которому подключен светодиод;
control - переменная, хранящая номер вывода, к которому подключено подстроечное сопротивление ;
val – переменная где будет храниться считываемое значение;

В процессе инициализации (функция setup) определяем D3 как выход.

В бесконечном цикле функции loop считываем значение из A5 – от 0 до 1023, преобразуем его в формат 0..255 и передаем считанное значение на ШИМ выход D3.

Как видно из скетча, изменяя при помощи подстроечного сопротивления на напряжения аналоговом входе от 0 до +5V мы добьемся плавного изменения яркости светодиода – от полного затухания до максимально возможной яркости.

Дополнительные функции ввода/вывода

Функция tone

Описание	Генерирует на порту вход/выхода сигнал — прямоугольную "волну", заданной частоты и с 50% рабочим циклом. Длительность может быть задана параметром, в противном случае сигнал генерируется пока не будет вызвана функция noTone(). К порту вход/выхода может быть подключен к пьезо или другой динамик для воспроизведения сигнала.
Синтаксис	tone(pin, frequency) tone(pin, frequency, duration)
Параметры	pin: номер порта вход/выхода, на котором будет генерироваться сигнал frequency: частота сигнала в Герцах duration: длительность сигнала в миллисекундах
Возвращаемое значение	нет

Функция noTone

Описание	Останавливает сигнал, генерируемый на порту вход/выхода вызывом функции tone(). Если сигнал не генерировался, то вызов noTone() ни к чему не приводит.
Синтаксис	noTone(pin)
Параметры	pin: номер порта вход/выхода, на котором прекращаем сигнал
Возвращаемое значение	нет

Функция pulseIn

Описание	Считывает длину сигнала на заданном порту (HIGH или LOW). Например, если задано считывание HIGH функцией pulseIn(), функция ожидает пока на заданном порту не появиться HIGH. Когда HIGH получен, включается таймер, который будет остановлен когда на порту вход/выхода будет LOW. Функция pulseIn() возвращает длину сигнала в микросекундах. Функция возвращает 0, если в течение заданного времени (таймаута) не был зафиксирован сигнал на порту.

	Возможны некоторые погрешности в измерение длинных сигналов. Функция может измерять сигналы длиной от 10 микросекунд до 3 минут.
Синтаксис	pulseIn(pin, value) pulseIn(pin, value, timeout)
Параметры	pin: номер порта вход/выхода, на котором будет ожидаться сигнал. (int) value: тип ожидаемого сигнала — HIGH или LOW. timeout (опционально): время ожидания сигнала (таймаут) в микросекундах; по умолчанию - одна секунда. (unsigned long)
Возвращаемое значение	Длина сигнала в микросекундах или 0, если сигнал не получен до истечения таймаута. (unsigned long)

Для примера ниже приведен скетч, в котором в переменную pulse_length в бесконечном цикле считывается длительность сигнала, подаваемого на 8 вход.

`int pin = 8;`
`unsigned long pulse_length;`
 	
`void setup()`
`{`
`  pinMode(pin, INPUT);`
`}`
 	
`void loop()`
`{`
`  pulse_length=pulseIn(pin, HIGH);`
`}`
	Pin - переменная, хранящая номер вывода, на который подается сигнал;
	pulse_length - переменная, хранящая длительность подаваемого сигнала;
	
	В процессе инициализации (функция setup) определяем D8 как вход.
	
	В бесконечном цикле функции loop, как только на D8 появляется уровень HIGH, начинаем считать длительность сигнала. После появления уровня LOW считывание прекращается и измеренная длительность сигнала записывается в переменную pulse_length.

Функции времени

millis

Описание	Возвращает количество миллисекунд с момента запуска скетча. Переполнение памяти, выделенной под хранение количества миллисекунд происходит приблизительно через 50 дней.
Синтаксис	millis()
Параметры	нет
Возвращаемое значение	количество миллисекунд (unsigned long)

micros

Описание	Возвращает количество микросекунд с момента запуска скетча. Переполнение памяти, выделенной под хранение количества миллисекунд происходит приблизительно через 70 часов. Минимальное разрешение для 16-битных платформ составляет 4 микросекунды.
Синтаксис	micros()
Параметры	Нет
Возвращаемое значение	количество микросекунд (unsigned long)

delay

Описание	Останавливает выполнение программы на заданное в параметре количество миллисекунд (1000 миллисекунд в 1 секунде).
Синтаксис	delay(ms)
Параметры	ms: количество миллисекунд, на которое приостанавливается выполнение программы. (unsigned long)
Возвращаемое значение	Нет

delayMicroseconds

Описание	Останавливает выполнение программы на заданное в параметре количество микросекунд (1 000 000 микросекунд в 1 секунде). Максимальное значение параметра, которое обеспечивает заданную точность (3 мкс) - 16383
Синтаксис	delayMicroseconds(micros)
Параметры	micros: количество микросекунд, на которое приостанавливается выполнение программы. (unsigned int)
Возвращаемое значение	Нет

Функции передачи данных

Для обмена данными в Arduino используется последовательный порт (UART, или Serial). Соединение происходит через выводы 0 (RX -прием) и 1 (TX-передача), так что не рекомендуется использовать эти выводы для других целей.

Serial.begin

Описание	Устанавливает соединение через последовательный порт с заданной скоростью. Скорость указывается в бодах (бит в секунду) и имеет фиксированный набор значений: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, или 115200. Проверка факта установки соединения происходит через условный оператор: if(Serial) или if(!Serial)
Синтаксис	Serial.begin(speed) В Arduino Mega есть три независимых последовательных порта, так что присутствует свой синтаксис: Serial1.begin(speed) Serial2.begin(speed) Serial3.begin(speed)
Параметры	speed: скорость соединения в бодах (long)
Возвращаемое значение	Нет

Serial.end

Описание	Разрывает соединение через последовательный порт.
Синтаксис	Serial.end() В случае Arduino Mega: Serial1.end() Serial2. end() Serial3. end()
Параметры	Нет
Возвращаемое значение	Нет

Serial.print

Описание	Передает в последовательный порт информацию в виде текста формата ASCII. Числа типа float по умолчанию искусственно ограничиваются двумя знаками после запятой. Для передачи в последовательный порт информации в любой другой кодировке, она должна быть помещена внутрь функции F(): Serial.print(F(вЂћHello WorldвЂќ)) Serial.print(78) выведет в порт "78" Serial.print(1.23456) - "1.23" Serial.print('N') - "N" Serial.print("Hello world.") - "Hello world."
Синтаксис	Serial.print(val) Serial.print(val, format)
Параметры	val – передаваемое значение (любой тип данных) format – формат вывода чисел: BIN – двоичный, OCT – восьмеричный, DEC – десятичный, HEX – шестнадцатеричный. Serial.print(78, BIN) выведет в порт "1001110" Serial.print(78, OCT) - "116" Serial.print(78, DEC) - "78" Serial.print(78, HEX) - "4E"
Возвращаемое значение	Количество переданных в порт байт (size_t (long))

Serial.println

Описание	Аналогично Serial.print передает в последовательный порт информацию в виде текста формата ASCII, только завершаемого символом конца строки.
Синтаксис	Serial.println(val) Serial.println(val, format)
Параметры	val – передаваемое значение (любой тип данных) format – формат вывода чисел: BIN – двоичный, OCT – восьмеричный, DEC – десятичный, HEX – шестнадцатеричный. Если вместо перечисленных констант указать целое положительное число, оно определит количество выводимых знаков после запятой.

	Serial.println(1.23456, 0) - "1" Serial.println(1.23456, 2) - "1.23" Serial.println(1.23456, 4) - "1.2346"
Возвращаемое значение	Количество переданных в порт байт (size_t (long))

Serial.write

Описание	Передает двоичные данные в последовательный порт.
Синтаксис	Serial.write(val) Serial.write(str) Serial.write(buf, len)
Параметры	val – передаваемое значение (1 байт) str – строка, передаваемая как набор байтов buf – массив, передаваемый как набор байтов len – длина передаваемого массива
Возвращаемое значение	Количество переданных в порт байт (byte)

Serial.read

Описание	Считывает данные из последовательного порта.
Синтаксис	Serial.read() Для Arduino Mega: Serial1.read() Serial2.read() Serial3.read()
Параметры	Нет
Возвращаемое значение	Первый доступный в порту байт или -1, если в порту данных нет (int).

Serial.readBytes

Описание	Считывает символы из последовательного порта в буфер.
Синтаксис	Serial.readBytes(buffer, length)
Параметры	buffer – буфер (массив) для хранения считанных символов (char[] или byte[]) length – число символов, которое надо считать (int)
Возвращаемое значение	Количество символов, считанных в буфер или 0, если таковых не нашлось (byte).

Serial.readBytesUntil

Описание	Считывает символы из последовательного порта в буфер до тех пор, пока не встретится терминальный символ, не будут считано нужное количество символов или не произойдет тайм-аут.
Синтаксис	Serial.readBytesUntil(character, buffer, length)
Параметры	character – символ, который остановит считывание данных (char) buffer – буфер (массив) для хранения считанных символов (char[] или byte[]) length – число символов, которое надо считать (int)
Возвращаемое значение	Количество символов, считанных в буфер или 0, если таковых не нашлось (byte).

Serial.setTimeout

Описание	Устанавливает время тайм-аута для функций Serial.read, Serial.readBytes и Serial.readBytesUntil. По умолчанию тайм-аут составляет 1000 миллисекунд.
Синтаксис	Serial.setTimeout(time)
Параметры	time – время тайм-аута (long)
Возвращаемое значение	нет

Serial.available

Описание	Возвращает число байт, доступных для считывания из буфера последовательного порта. Размер буфера составляет 64 байта.
Синтаксис	Serial.available() Для Arduino Mega: Serial1.available() Serial2.available() Serial3.available()
Параметры	Нет
Возвращаемое значение	Число доступных для считывания байт.

Serial.find

Описание	Ищет заданную строку в буфере последовательного порта.
Синтаксис	Serial.find(target)
Параметры	target – искомая строка (char)
Возвращаемое значение	boolean

Serial.findUntil

Описание	Ищет заданную строку в буфере последовательного порта до тех пор, пока не встретится терминальная строка.
Синтаксис	Serial.findUntil(target, terminal)
Параметры	target – искомая строка (char) terminal – терминальная строка (char)
Возвращаемое значение	boolean

Serial.parseInt

Описание	Ищет следующее целочисленное значение в буфере последовательного порта.
Синтаксис	Serial.parseInt()
Параметры	нет
Возвращаемое значение	int

Serial.parseFloat

Описание	Ищет следующее число с плавающей запятой в буфере последовательного порта. Любой символ, отличный от цифры, является признаком конце числа с плавающей запятой.
Синтаксис	Serial.parseFloat()
Параметры	нет
Возвращаемое значение	float

Ниже приведен пример использования функций передачи данных:

```
int incbyte = 0;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  if(Serial.available()>0){
incbyte = Serial.read();
Serial.print("Получено: ");
Serial.print(incbyte, DEC);
}
}
```

incbyte - переменная, хранящая данные, считанные из последовательного порта;

В процессе инициализации (функция setup) устанавливаем связь по последовательному порту со скоростью 9600 бод.

В бесконечном цикле функции loop проверяем доступность данных во входном буфере последовательного порта, и если буфер не пуст, считываем данные в переменную incbyte. Затем отправляем через последовательный порт строку «Получено:» и считанный ранее байт с последующим символом конца строки.

ИСПОЛЬЗОВАНИЕ ARDUINO В АВТОМАТИЗИРОВАННЫХ СИСТЕМАХ УПРАВЛЕНИЯ ОПТИЧЕСКИМИ И ОПТОМЕХАНИЧЕСКИМИ УСТРОЙСТВАМИ

Управление шаговыми двигателями оправ и трансляторов

Шаговые двигатели широко используются в оптомеханике для дистанционной прецизионной юстировки оптических элементов. Шаговый двигатель - это электромеханическое устройство, преобразующие сигнал управления в угловое (или линейное) перемещение ротора с фиксацией его в заданном положении. В отличие от двигателя постоянного тока у шагового двигателя отсутствуют щетки и коммутатор - для этого там несколько отдельных обмоток, которые коммутируются внешней электроникой (драйвером). Вращение ротора происходит за счет коммутации обмоток шаг за шагом, без обратной связи.

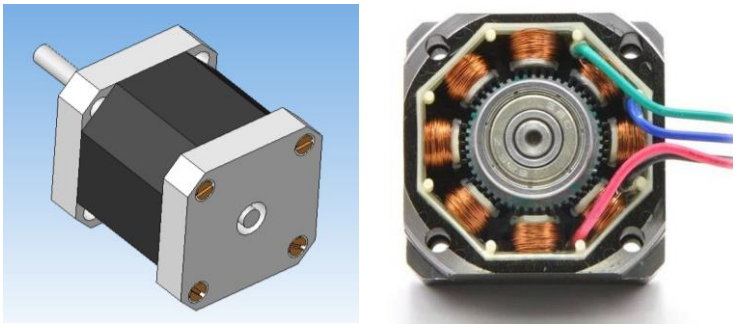


Рисунок 9. Шаговый двигатель

Здесь проявляется и один недостаток шаговых двигателей - в случае механической перегрузки, когда ротор не двигается, шаги начинают путаться и движение становится неточным. По виду обмоток, шаговые двигатели разделяются на два типа: **униполярные** и **биполярные** шаговые двигатели. По строению их делят еще на три вида:

- С переменным магнитным сопротивлением (высокая точность, низкий крутящий момент, низкая цена)
- С постоянным магнитом (низкая точность, высокий крутящий момент, низкая цена)
- Гибридный (высокая точность, высокий крутящий момент, высокая цена)

У шаговых двигателей с **переменным магнитным сопротивлением** зубчатые обмотки и зубчатый ротор из железа. Максимальная сила тяги возникает при перекрытии зубьев обоих сторон. В шаговых двигателях с **постоянным магнитом**, как следует из названия, есть постоянный магнит, который ориентируется в зависимости от полярности обмотки. В **гибридных** используются обе технологии.

Независимо от модели шагового двигателя для создания одного полного оборота вала (360 градусов) требуется сотня коммутационных шагов. Для обеспечения стабильного и плавного движения используют подходящую управляющую электронику, которая управляет двигателем в соответствии с его параметрами (инертность ротора, крутящий момент, резонанс и т.д.). Вдобавок в управляющей электронике можно применять различные методы коммутации. Коммутацию последовательно по одной обмотке называют полным шагом, но если коммутируется поочередно одна и две обмотки, то это называется полушагом. Используют так же синусоидальные микрошаги, что дает особую точность и плавность управления.

Униполярный шаговый двигатель имеет пять или шесть проводов. В соответствии со схемой привода запускается разом только одна четвертая обмоток. Линии *Vcc* (см.рис.10) обычно соединяются с положительным питающим напряжением двигателя. Концы обмоток 1a, 1b, 2a, и 2b соединяются при коммутации через транзисторы только с землей, в связи, с чем их управляющая электроника довольно простая.

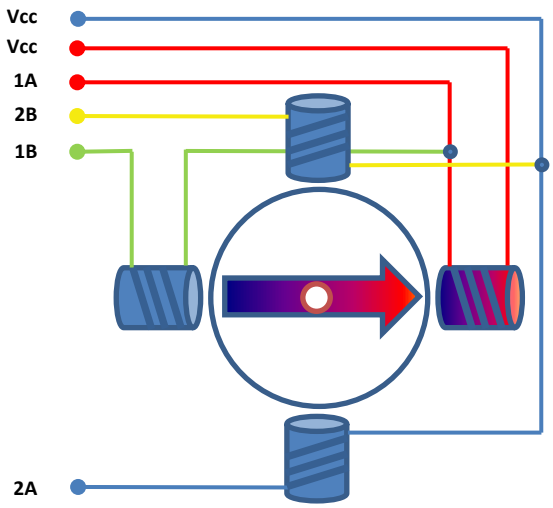


Рисунок 10. Схема обмоток униполярного шагового двигателя.

Биполярный шаговый двигатель отличается от униполярного шагового двигателя тем, что полярность обмоток изменяется во время коммутации. Разом активируется половина обмоток, что обеспечивает в сравнении с униполярными шаговыми двигателями большую эффективность. У биполярных шаговых двигателей четыре провода, которые все соединяются отдельно полумостом. При коммутации полумосты прикладывают к концам обмоток положительное или отрицательное напряжение. Униполярные шаговые двигатели можно запускать и с помощью биполярного драйвера: для этого нужно соединить только линии обмоток 1a, 1b, 2a и 2b (*Vcc* остается не соединенным).

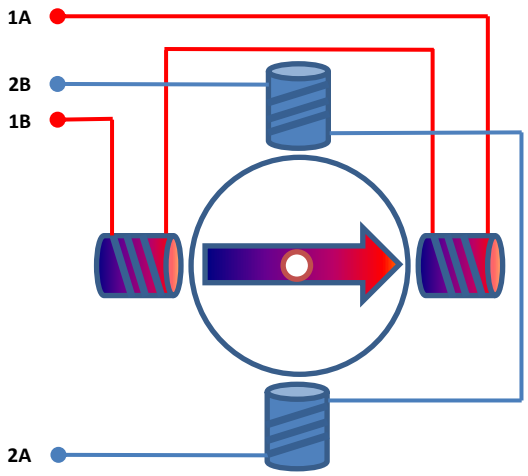


Рисунок 11. Схема обмоток биполярного шагового двигателя.

Необходимые коммутации полного шага и полушага шаговых двигателей с обоими видами обмоток отображает следующая таблица. «0» соответствует отрицательной полярности, «+» - положительной.

Таблица 11. Управление обмотками шагового двигателя (полный шаг)

Шаг	**1A**	**2A**	**1B**	**2B**
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Таблица 12. Управление обмотками шагового двигателя (полушаг)

Шаг	1A	2A	1B	2B
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Для управления шаговыми двигателями при помощи Arduino существуют специальные драйверы и платы расширения, т.к. выводы микроконтроллера являются слаботочными, поэтому ток мотора, при подключении его напрямую, выведет их из строя. Эти платы позволяют управлять скоростью и направлением вращения двигателя с помощью логических сигналов микроконтроллера. Среди всего разнообразия, представленного на рынке, мы рассмотрим платы MotorShield на основе микросхемы-драйвера L298P и MotorShield на основе микросхемы-драйвера L293D.

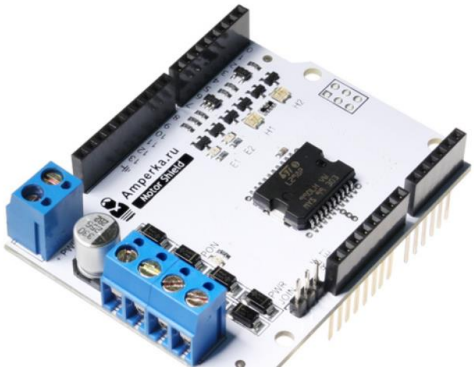


Рисунок 12. Внешний вид платы расширения MotorShield L298P

MotorShield L298P - плата расширения для Arduino на базе чипа L298P, позволяющая управлять моторами с напряжением 5–24 В в режиме раздельного питания и 7–12 В в режиме объединённого питания. Форм-фактор платы позволяет использовать MotorShield как надстройку для платы микроконтроллера Arduino UNO.

Плата имеет 2 независимых канала. Используя их, можно подключить на выбор: пару маломощных двигателей постоянного тока (DC-моторов), один биполярный шаговый двигатель, один DC-мотор с током до 4 А - если объединить каналы.

Для коммуникации с микроконтроллером используются цифровые контакты Arduino:

- D4 — направление, правый
- D5 — скорость (ШИМ), правый
- D6 — скорость (ШИМ), левый
- D7 — направление, левый

Рисунок 13. Внешний вид платы расширения MotorShield L293D

MotorShield L293D - плата расширения для Arduino на базе чипа L293D, по функционалу аналогичная MotorShield L298P. Форм-фактор платы позволяет использовать MotorShield как надстройку для платы микроконтроллера Arduino
Mega2560. При этом оказываются задействованы все 6 аналоговых входов, которые могут быть использованы как цифровые порты (с 14 по 19).
Входы, занятые только при использовании двигателей (DC или шагового):

- D11 — DC мотор #1 / шаговый #1
- D3 — DC мотор #2 / шаговый #1
- D5 — DC мотор #3 / шаговый #2
- D6 — DC мотор #4 / шаговый #2

Эти контакты заняты при использовании любого из шаговых двигателей:

- D4, D7, D8 и D12 для управления DC/Шаговыми моторами с помощью микросхемы сдвигового регистра 74HC595, расположенной на плате.

Схема коммутации MotorShield L298P приведена ниже.

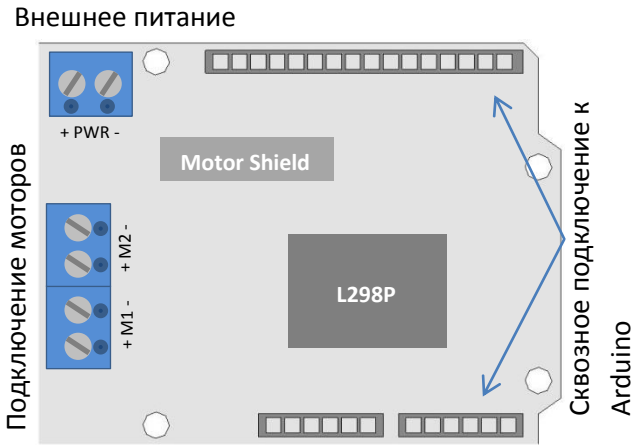


Рисунок 14. Схема коммутации MotorShield L298P

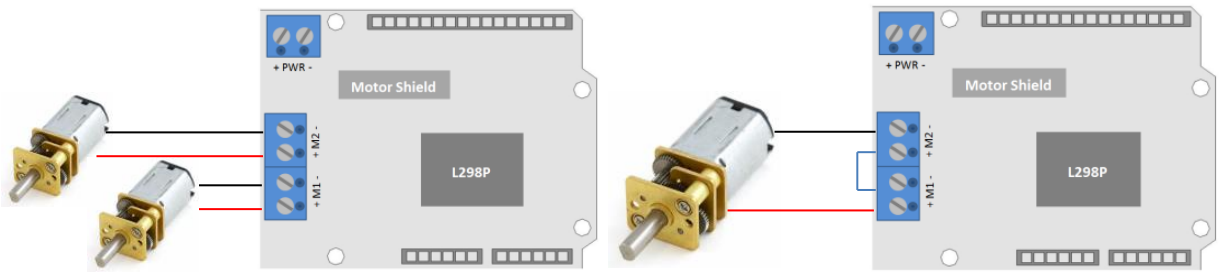


Рисунок 15. Подключение DC-моторов к MotorShield L298P.

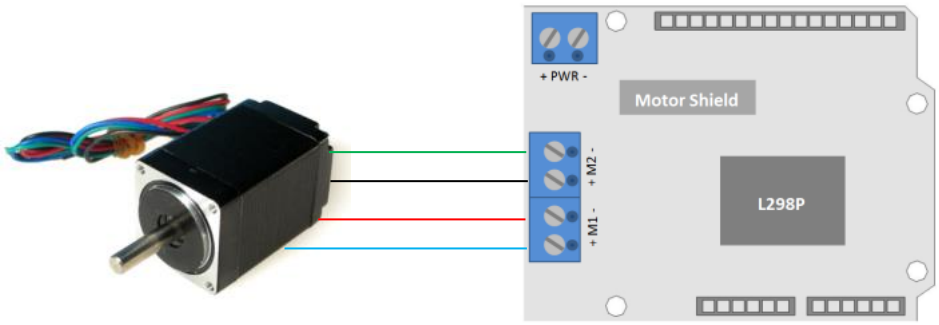


Рисунок 16. Подключение биполярного шагового двигателя к MotorShield L298P.

Управление DC-мотором, подключенным к M1:

Цифровой выход D4 задает направление вращения – HIGH соответствует вращению по часовой стрелке, а LOW - против часовой стрелки. Цифровой выход D5 задает скорость вращения путем изменения ШИМ от 0 до 255.

Аналогично управляется мотор, подключенный к M2, только за скорость вращения отвечает цифровой выход D6, а за направление вращения – цифровой выход D7.

Пример программы:

`void setup()`
`{`
`  pinMode(4, OUTPUT);`
`  pinMode(5, OUTPUT);`
`}`
	В процессе инициализации (функция setup) определяем D4 и D5 как выходы.

`void loop()`
`{`
`  digitalWrite(4, HIGH);`
`  analogWrite (5, 100);`
`}`
	Задаем направление вращения – по часовой стрелке (D4->HIGH) и скорость вращения 100 (D5->100)

Для программирования вращения шаговых двигателей или DC-моторов можно использовать библиотеку Stepper, входящую в состав IDE Arduino. Подключение библиотеки происходит посредством директивы include:

#include<Stepper.h>

Далее при помощи соответствующего конструктора создается необходимое количество экземпляров класса Stepper, которые содержат необходимые функции:

Stepper (steps, pin1, pin2, pin3, pin4)

Здесь **steps** – характеристика конкретного подключенного шагового двигателя – количество шагов в одном полном обороте, pin1-pin4 – номера цифровых входов, управляющих работой двигателя. Для MotorShield L298P – это 4, 5, 6, 7. Управляющих функций всего две. Первая,

setSpeed(rpm)

устанавливает скорость вращения двигателя при помощи параметра rpm (тип *long*) – количества оборотов в минуту. Вторая функция

step(n)

– заставляет двигатель совершать n шагов с установленной ранее скоростью rpm. Параметр **n** имеет тип *int* и его положительное значение означает движение по часовой стрелке, а отрицательное – против. Пример управления

шаговым двигателем, подключенным к Arduino посредством MotorShield L298P приведен ниже:

`#include<Stepper.h>`
 	
`Stepper motor=Stepper(200, 4,5,6,7);`
 	
`void setup()`
`{`
`  motor.setSpeed(30);`
`}`
 	
`void loop()`
`{`
`  motor.step(100);`
`}`
	Подключаем библиотеку Stepper
	
	Создаем экземпляр класса Stepper с именем motor. В конструкторе прописываем кол-во шагов на один полный оборот двигателя – 200, и номера управляющих выходов 4-7.
	
	Устанавливаем скорость движения – 30 оборотов в минуту.
	
	Делаем 100 шагов по часовой стрелке с заданной скоростью.

При помощи шаговых двигателей можно организовать управление некоторыми оптомеханическими элементами, такими как оптические оправы и трансляторы.

Оправы предназначены для фиксации оптических элементов и углового позиционирования в пространстве в зависимости от предусмотренных степеней свободы. Так, например, оправа, приведенная на рис.17, позволяет заклонять зажатый ее подвижную часть элемент в двух плоскостях путем вращения микрометрических винтов. Во многих серийно изготавливаемых оправах предусмотрена возможность замены ручек микрометрических винтов на шаговые двигатели (Рис.17).

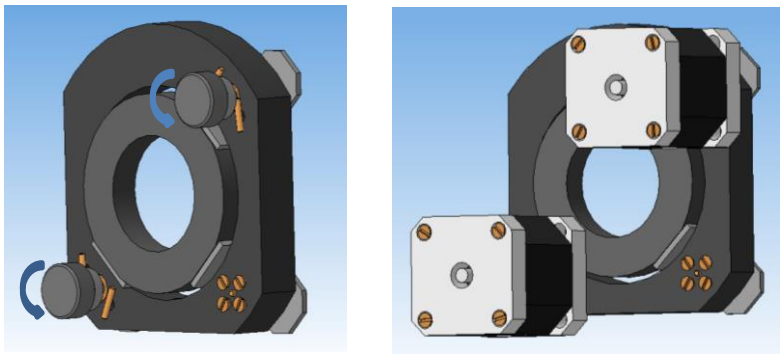


Рисунок 17. Оптические оправы с ручными и автоматическими регулировками.

Транслятор предназначен для перемещения закрепленного на нем объекта в одной или нескольких плоскостях. Транслятор, изображенный на рис.18, перемещает подвижную площадку в одном выделенном направлении путем осуществления червячной передачи от шагового двигателя.

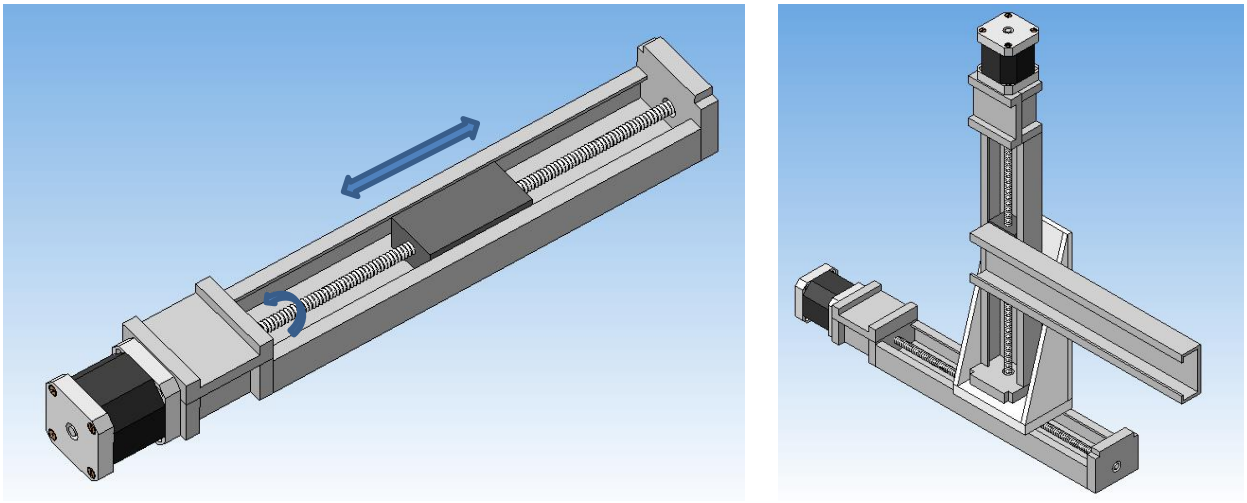


Рисунок 18. Транслятор и система двухкоординатного сканирования.

Один транслятор, закрепленный на каретке другого, позволяют создать систему двухкоординатного сканирования, которое часто используется в оптике для построения изображений и снятия пространственных распределений.

Определение крайних положений при передвижении может быть выполнено при помощи так называемых концевых выключателей – миниатюрных кнопок или размыкателей, установленных на пути перемещающегося объекта. Концевые выключатели могут быть как механическими, так и оптическими. В первом случае при воздействии происходит замыкание или размыкание электрических контактов, во втором появляется или исчезает преграда между светодиодом и фотодиодом.

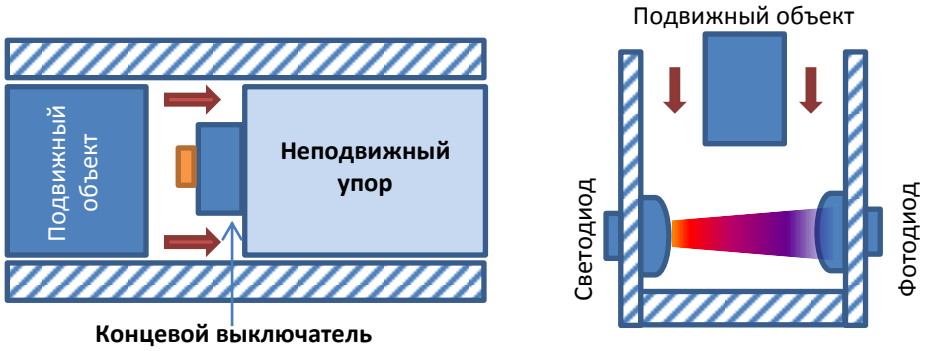


Рисунок 19. К принципу работы концевых выключателей.

Концевые выключатели также могут быть подсоединены к платформе Arduino через цифровые входы и шину питания +5V.

Реализуем систему двухкоординатного сканирования на базе микроконтроллера Arduino UNO с подключенными к нему двумя платами-драйверами шаговых двигателей и четырьмя концевыми выключателями.

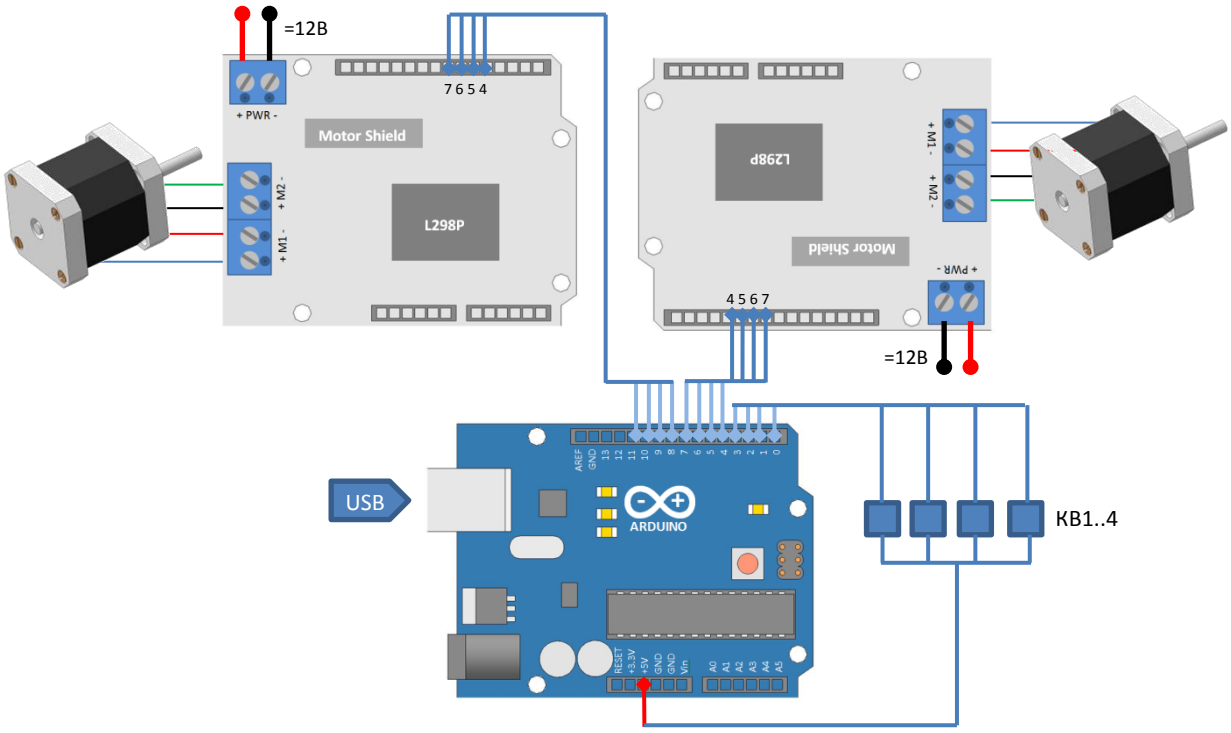
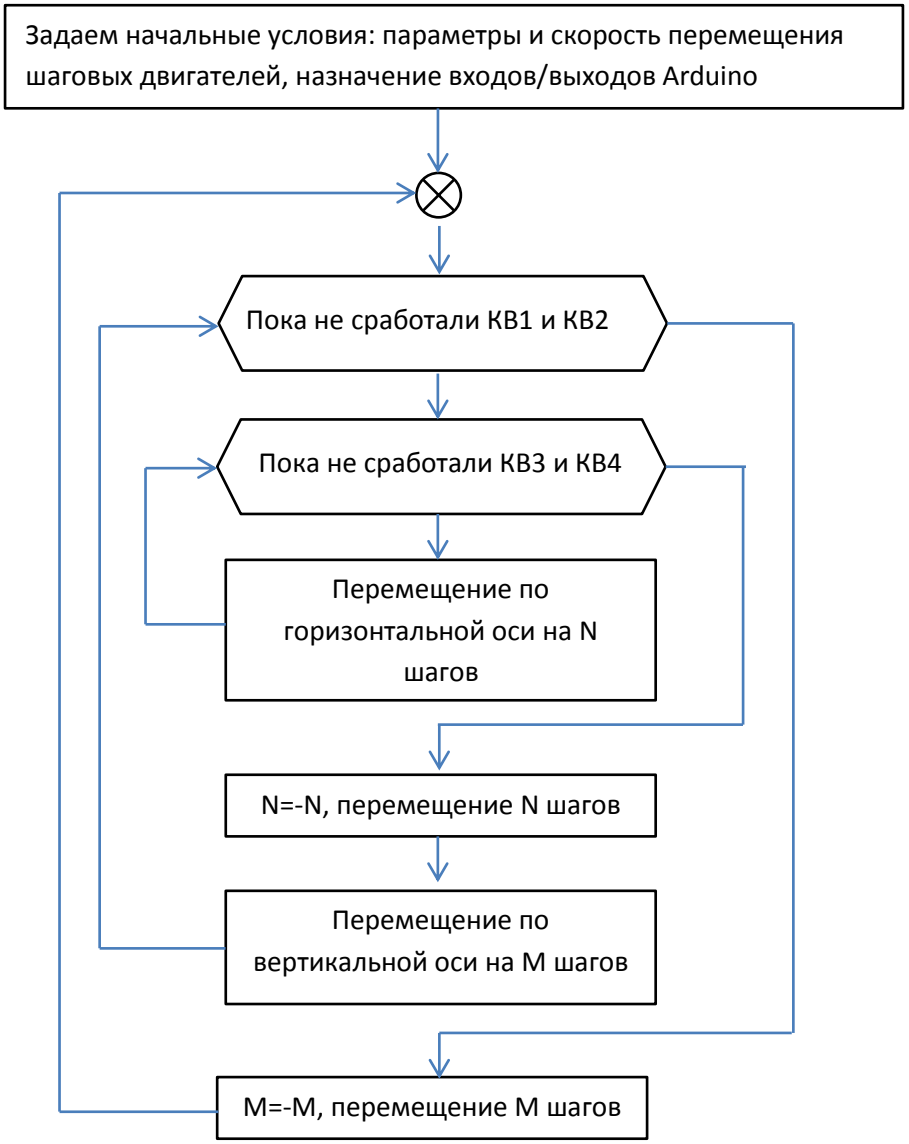


Рисунок 20. Схема управления системой двухкоординатного сканирования.

Шаговые двигатели входят в состав трансляторов, закрепленных так, как было показано на рис.18, образуя горизонтальную и вертикальную оси сканирования.

Алгоритм построчного сканирования, использующий сигналы от концевых выключателей, будет выглядеть следующим образом:



Следует обратить внимание, что после обработки действий, следующих за срабатыванием концевых выключателей, необходимо их «освободить», совершив движение в противоположную срабатыванию сторону.

?	Задание: напишите программу двухкоординатного сканирования, реализованного по представленному алгоритму.

Управление оборудованием и приборами через стандартные интерфейсы передачи данных

В настоящий момент большая часть оборудования и приборов, использующихся при проведении исследований в области лазерной физики и техники, имеет возможность дистанционного управления посредством подключения к управляющим элементам через стандартные интерфейсы передачи данных, такие как RS232, USB, Ethernet, и т.п. Протокол управляющих команд для такого исследовательского оборудования обычно является открытым и поставляется в комплекте или может быть найден на сайте производителя.

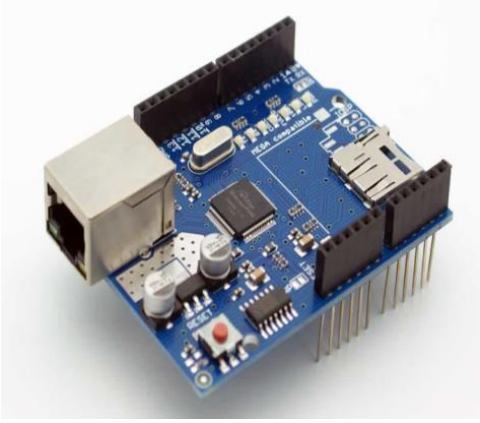


Рисунок 21. Плата расширение Ethernet Shield.

Подключение может быть реализовано через платы расширений Arduino, доступные на рынке. Например, связь через интерфейс Ethernet представляет Arduino Ethernet Shield. Популярность данного интерфейса среди производителей оборудования обусловлена его универсальностью и простотой использования. Плата основана на чипе Wiznet W5 500, который поддерживает TCP и UDP протоколы. Рабочее напряжение 5 В, размер буфера 32 Кб, скорость соединения 10/100 Мбит. При подключении к Arduino UNO занимает входы SPI (11,12,13), 10 и 4. Одновременно открытыми может быть открыто до 8 подключений. Ethernet Shield выступает в роли полноценного сетевого устройства, таким образом может быть осуществлено управление через интернет. Для соединения на плате имеется стандартный разъем 8P8C (RJ45). Также имеется слот для microSD карты объемом до 2 Гб. Для программирования сетевого взаимодействия используется библиотека Ethernet из стандартного комплекта IDE.

Подключение библиотеки происходит стандартным способом:

#include<Ethernet.h>

В состав библиотеки входят функции, обеспечивающие ее работу. Для инициализации Ethernet-соединения используется функция

Ethernet.begin(mac [, ip, dns, gateway, subnet);]

с параметрами **mac** – MAC-адрес устройства, **ip** – ip-адрес устройства, **dns** – адрес DNS сервера, **gateway** – ip-адрес шлюза, **subnet** – маска подсети. Обязательным параметром является только **mac**. MAC – адрес устройства обычно указан в документации на него или на стикере, приклеенном к плате. В случае старых версий платы Ethernet Shield можно самим задать MAC-адрес. Маска подсети по умолчанию – 255.255.255.0. Функция **begin** возвращает 1 в случае успешного подключения и 0 в противном случае. Ниже – пример подключения к Ethernet Shield:

`#include<Ethernet.h>`
 	
`byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };`
`byte ip[] = { 10, 0, 0, 177 };`
 	
`void setup()`
`{`
`  Ethernet.begin(mac, ip);`
`}`
 	
`void loop()`
`{`
 	
`}`
	Подключаем библиотеку Ethernet
	
	Прописываем MAC-адрес устройства.
	
	Прописываем ip-адрес устройства
	
	Устанавливаем связь с Ethernet Shield по MAC-адресу назначаем ip-адрес.

Для того, чтобы потом узнать ip-адрес устройства, можно использовать функцию

Ethernet.localIP()

Класс **EthernetServer** позволяет создать сервер на базе Ethernet Shield для приема и обработки информации, поступающей от клиентских приложений. Конструктор класса имеет один параметр – номер порта, который сервер будет «слушать».

EthernetServer(port)

У класса есть функции **begin**() для старта серверного приложения и **available**() для установления готовности чтения данных с клиентского приложения. Для передачи данных в клиентское приложение используются функции **write**(), **print**() и **println**().

Класс **EthernetClient** создает клиента на базе Ethernet Shield для подключения к заданному порту. В классе реализованы функции:

- **connect**(ip, port) – для соединения с ресурсом по его ip-адресу и порту;
- **connect**(URL,port) – для соединения с ресурсом по его URL и порту;
- **connected**() – для проверки, установлено ли соединение с ресурсом;
- **available**() – для установления готовности чтения данных с сервера;
- **read**() – для чтения данных с сервера;
- **write(), print(), println**() – для передачи данных на сервер;
- **stop**() – для разрыва соединения.

Пример использования классов EthernetServer и EthernetClient:

`#include<Ethernet.h>` `#include <SPI.h>`	Подключаем библиотеку Ethernet и SPI
`byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };`	Прописываем MAC-адрес устройства.

`byte ip[] = { 10, 0, 0, 177};`
`byte gateway[] = { 10, 0, 0, 1 };`
`byte subnet[] = { 255, 255, 0, 0};`
	Прописываем ip-адрес устройства
	Прописываем адрес шлюза
	Прописываем маску подсети

`EthernetServer server = EthernetServer(23);`	Создаем экземпляр класса EthernetServer, «слушающего» порт 23 (Telnet)

`void setup()`
`{`
`  Ethernet.begin(mac, ip, gateway, subnet);`
	Устанавливаем связь с Ethernet Shield по MAC-адресу, назначаем ip-адрес, расписываем маршрутизацию.

`  server.begin();` `}`	Даем команду серверу «слушать» порт 23.
`void loop()` `{`	
`  EthernetClient client = server.available();`	Создаем экземпляр класса EthernetClient, готовый передать серверу данные.

`if (client == true)`
`{`
`server.write(client.read());`
`}`
`}`
	Если есть данные для передачи данным клиентом, принимаем их и передаем их обратно.

?	Задание: напишите программу подключения к некоторому устройству через Ethernet для считывания данных. MAC-адрес – любой, ip-адрес 192.168.1.1. Формат пакета данных: 2 байта заголовок пакета (0xA0, 0xAD), 8 байт содержимое пакета, 2 байта – конец пакета (0xBB,0xBB)

Управление оборудованием и приборами через сигнальные цепи

Сигнальные цепи используются для приема и передачи простых управляющих низковольтных сигналов, например сигналов синхронизации, коммутации, и т.п.

Рассмотрим реализацию генератора задержек на базе микроконтроллера Arduino UNO. Генератор должен принимать на входе импульс TTL уровня, и с регулируемой задержкой отправлять по нескольким выходным каналам. Для простоты вначале разберемся с реализацией одного выходного канала.

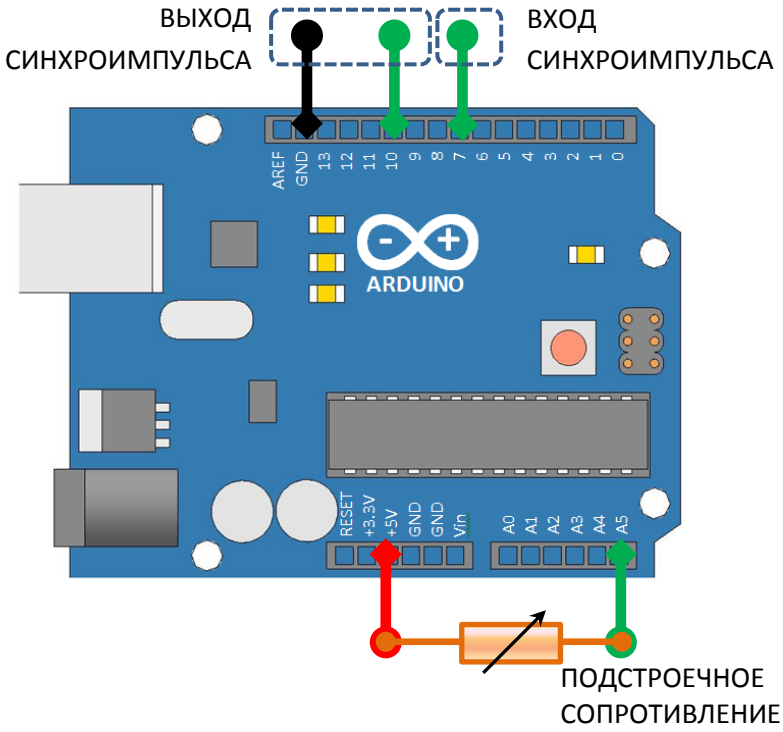
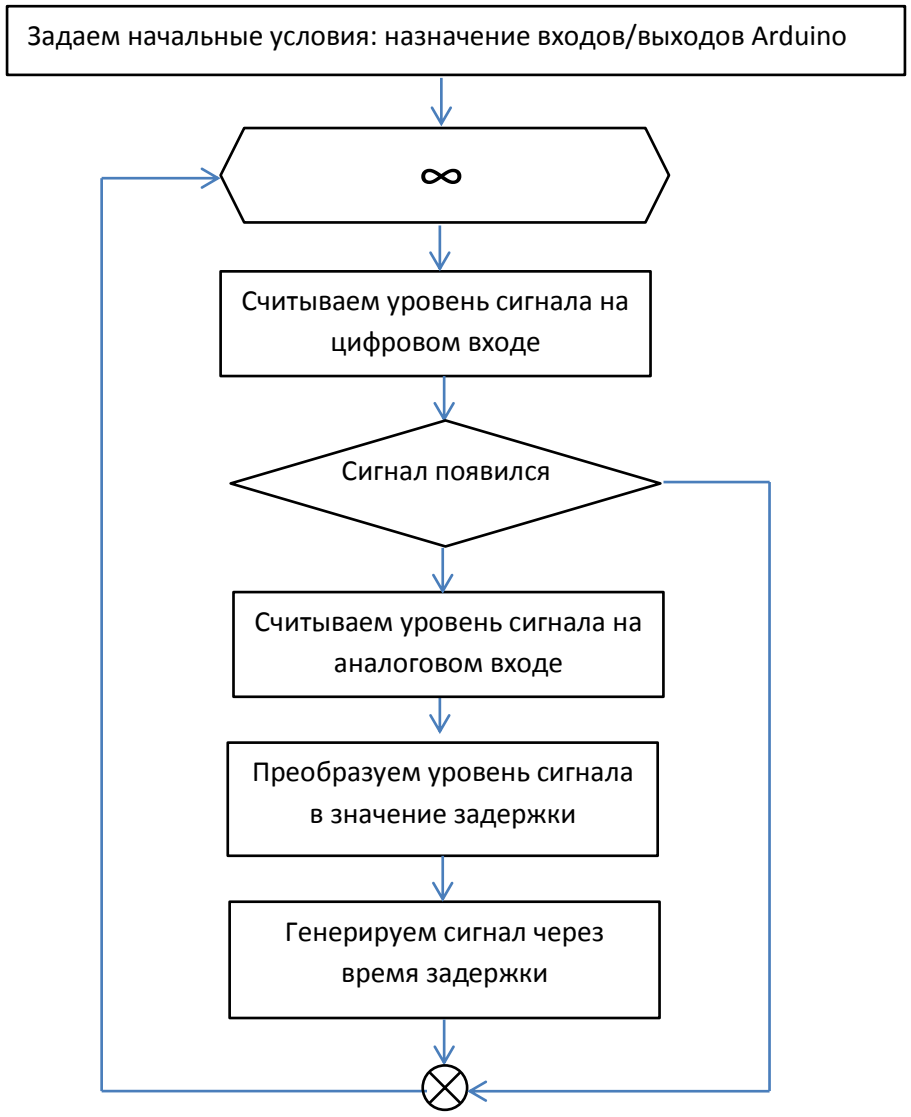


Рисунок 22. Схема генератора задержек.

Входом и выходом могут служить цифровые входы микроконтроллера Arduino. Регулировать задержку будем при помощи подстроечного сопротивления, подключенного к аналоговому входу контроллера через шину +5V.

Алгоритм функционирования генератора должен быть следующим:



?	Задание: напишите программу генерации задержек по представленному алгоритму.

Управление оборудованием в отсутствие интерфейсов и сигнальных цепей

Оборудованием, в котором не предусмотрена возможность внешнего управляющего воздействия, тем не менее, можно управлять. Так, характеристики излучения некоторых лазерных диодных модулей зависят от напряжения и тока, подаваемых через шину питания. Изменяя эти параметры можно добиться необходимой мощности излучения или осуществить его временную и амплитудную модуляцию. Например, можно подключить лазерный диодный модуль HLM1230 к одному из цифровых входов/выходов, работающих с ШИМ, как показано на рис.23. Регулировку уровня ШИМ легко осуществить путем подключения подстроечного сопротивления к аналоговому входу через шину питания +5V. Как известно, АЦП микроконтроллера преобразует уровень входного сигнала 0-5V в диапазон значений от 0 до 1023. Чтобы масштабировать этот диапазон до необходимых для управления ШИМ 0-255 можно использовать рассмотренную ранее функцию map(value, fromLow, fromHigh, toLow, toHigh), которая пропорционально переносит значение (value) из текущего диапазона значений (fromLow .. fromHigh) в новый диапазон (toLow .. toHigh), заданный параметрами. Функция возвращает значение в новом диапазоне.

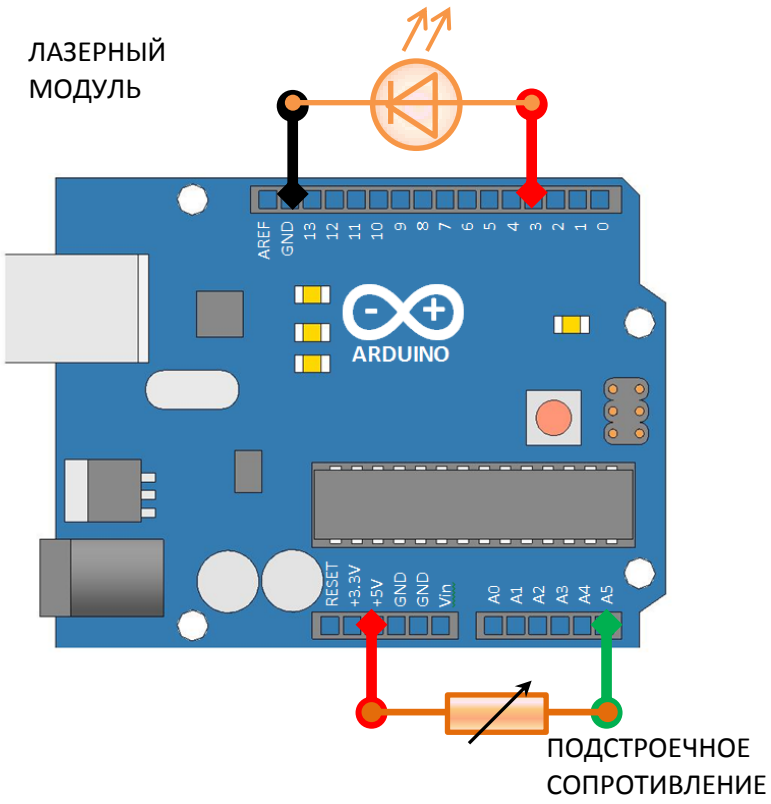


Рисунок 23. Схема управления лазерным диодным модулем HLM1230.

Для того, чтобы перевести значение, выданное АЦП в диапазон чисел, подходящих для управления ШИМ следует использовать функцию map следующим образом:

pwm=map(analogRead(pin), 0, 1023, 0, 255)

где pin – номер аналогового входа, к которому подключено подстроечное сопротивление, pwm – преобразованное значение ШИМ. Следует отметить, что рассмотренный способ управления ШИМ сильно зависит от характеристик подстроечного сопротивления и не сможет предоставить плавное и точное изменение значений. Управление ШИМ можно реализовать путем передачи нужных значений непосредственно в скетч через персональный компьютер при помощи класса Serial. Так как в Arduino UNO последовательный интерфейс и USB объединены, возможна передача данных в последовательный порт микроконтроллера по стандартному USB кабелю. Для реализации независимого подключения к последовательному порту необходимо использовать цифровые выводы 0 (RX – прием данных) и 1(TX – передача данных). В таком случае передаваемые данные должны иметь соответствующий формат:

Рисунок 24. Формат передачи данных через последовательный порт.

Последовательный порт также может пригодиться для связи нескольких микроконтроллеров между собой для расширения количества используемых выводов.

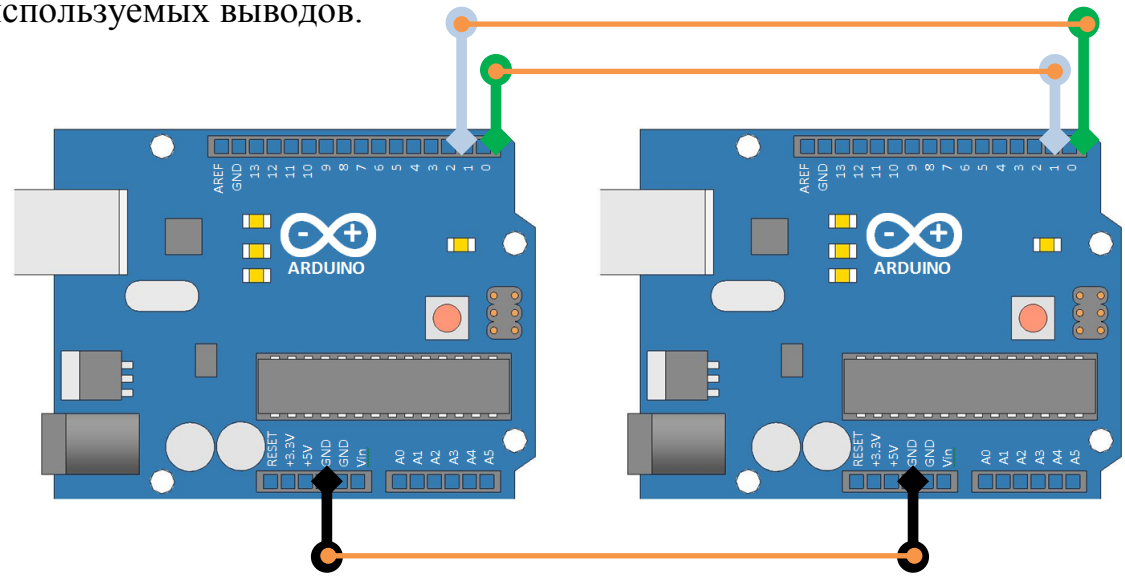


Рисунок 25. Схема связи двух плат Arduino через последовательный порт.

Для этого необходимо объединить устройства по шине «земли» (GND) и перекрестно соединить цифровые входы/выходы, отвечающие за передачу данных: D0 (RX1)+ D1(TX2), D1(TX1)+D0(RX2)

?	Задание: напишите программу обмена данными между двумя контроллерами Arduino UNO

Использование датчиков

Использование датчиков совместно с Arduino позволяет создавать полноценные автоматизированные системы управления с обратной связью. Классическая схема АСУ с обратной связью выглядит следующим образом:

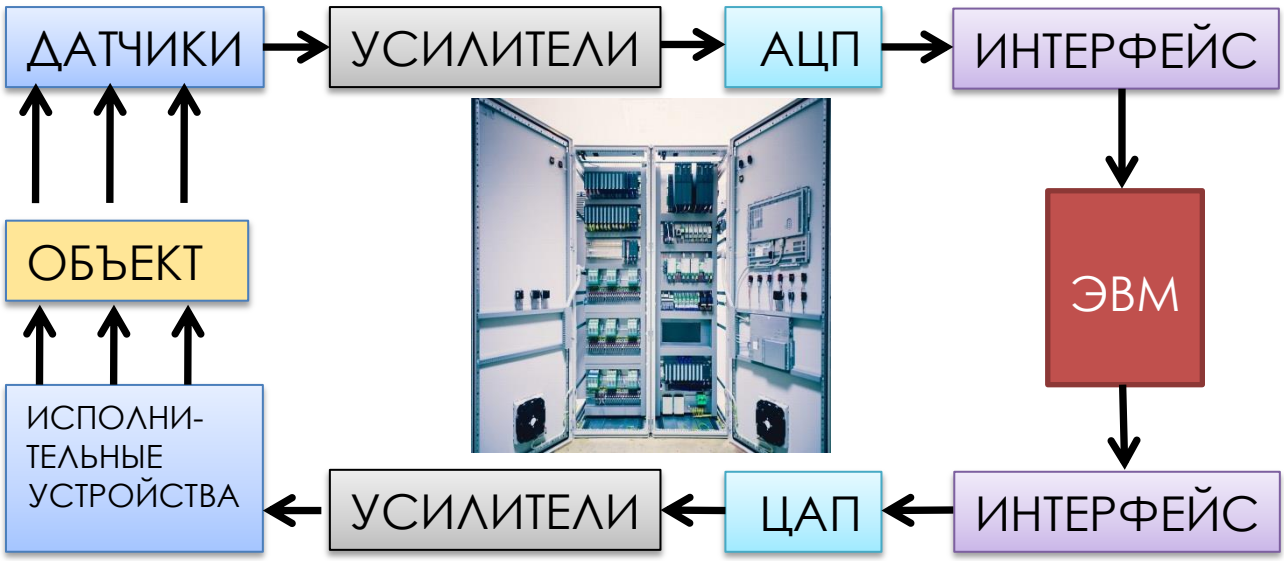


Рисунок 26. Схема АСУ с обратной связью.

Сигнал от датчиков должен быть преобразован по амплитуде, чтобы соответствовать характеристикам входного сигнала, предъявляемым аналогово-цифровым преобразователем (АЦП). АЦП посредством специального или стандартного интерфейса передает данные в ЭВМ для обработки. Обработанные данные могут быть отправлены на цифро-аналоговый преобразователь (ЦАП) для управления исполнительными устройствами.

Так как Arduino и имеющиеся платы расширения имеют достаточно большой функционал, схема АСУ с использованием Arduino выглядит немного иначе:

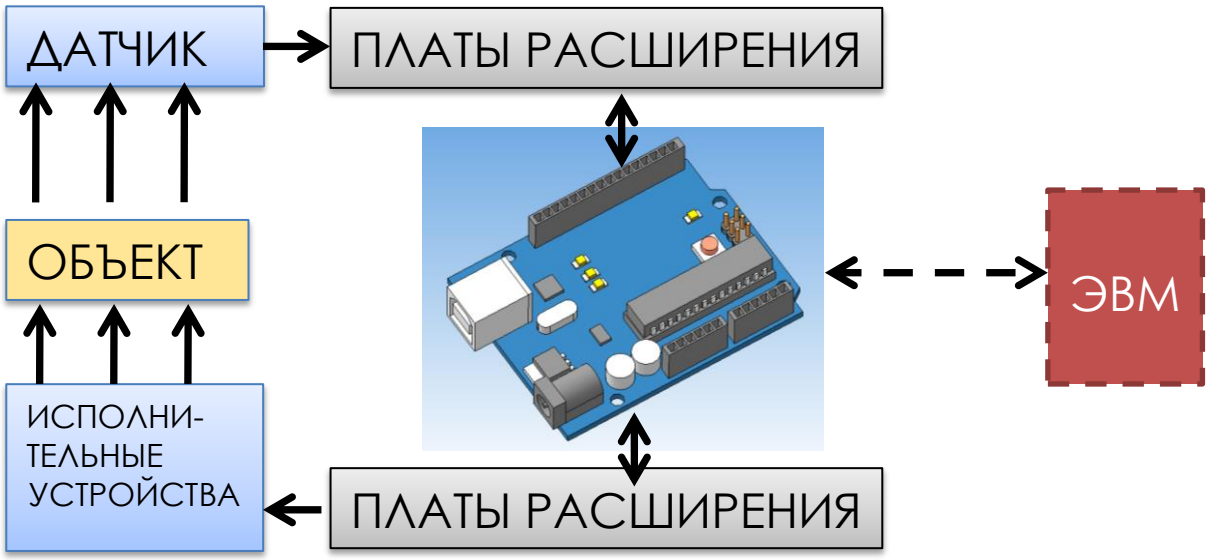


Рисунок 27. Схема АСУ с использованием Arduino.

Рассмотрим, как можно контролировать температуру оптического кристалла, входящего в состав задающего генератора лазерных импульсов. Измерение температуры на поверхности кристалла легко провести при помощи терморезистивного датчика или термопары, который подключается напрямую к Arduino.

Изменить температуру на поверхности кристалла способен элемент Пельтье, действующий следующим образом: при подаче напряжения на выводы элемента, одна его сторона начинает охлаждаться, за счет чего сильно нагревается другая сторона. Для питания элемента Пельтье необходимо иметь 12-24 В и ток до 2.5А, что подразумевает использование платы расширения PowerShield.

Термодатчик и элемент Пельтье закрепляются на противоположных гранях кристалла, не мешающих прохождению лазерного излучения. Программа, записанная в Arduino непрерывно считывает значения температуры, и по принципу отрицательной обратной связи корректирует её, усиливая или уменьшая охлаждение (рис.28).

?	Задание: напишите программу, поддерживающая температуру кристалла на уровне 35±1°C при помощи термодатчика и элемента Пельтье. «0В» на выходе термодатчика соответствует 5 °C, «5В» на выходе термодатчика соответствует 50 °C, точность измерения 0,1 В. Элемент Пельтье имеет температуру 10 °C при подаче на него 12В, и 40 °C при подаче 3.5В, точность установки напряжения 0.2 В. Уровень напряжения на выходе PowerShield контролируется по UART (байт «0» соответствует 0В, байт «255» – 24В)

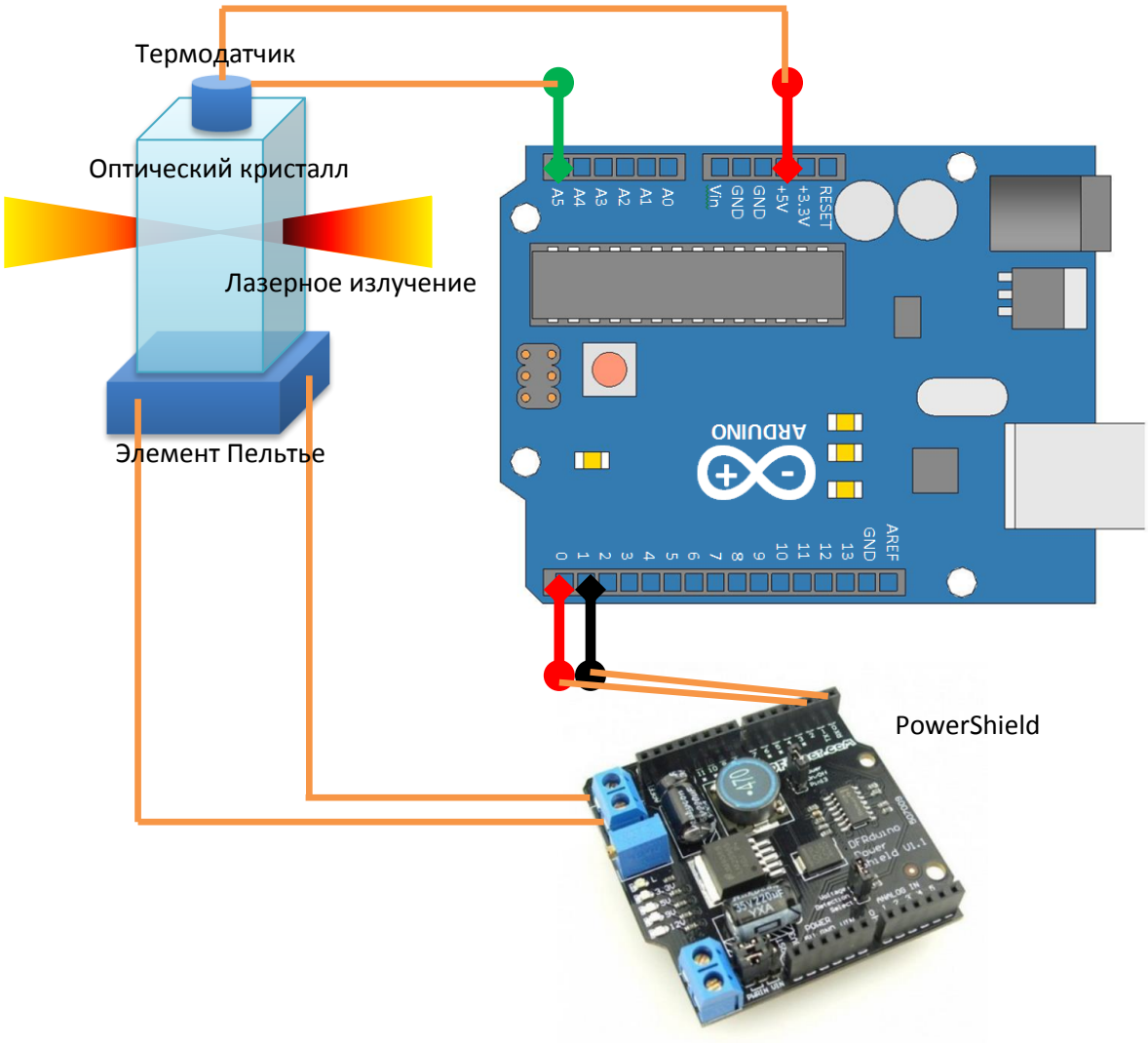


Рисунок 28. К схеме контроля температуры кристалла.

РЕСУРСЫ В СЕТИ ИНТЕРНЕТ

#	Адрес / URL	Описание
1	arduino.ru	Введение в программирование микроконтроллеров Arduino
2	arduino.cc	Официальный сайт платформы Arduino
3	amperka.ru	Полезная информация о микроконтроллерах Arduino, платах расширения, датчиках и т.п., интернет-магазин

Автор благодарит сайт amperka.ru за ряд фотографий некоторых устройств Arduino, плат расширения и датчиков.