Оглавление

[Основные понятия и характеристики архитектуры микропроцессоров 2](#_Toc106052688)

[Назначение микропроцессоров 3](#_Toc106052689)

[Классификация микропроцессоров 3](#_Toc106052690)

[4. Структура микропроцессорной системы 3](#_Toc106052691)

[5. Назначение и функции чипсета в микропроцессорной системе 4](#_Toc106052692)

[6. Основные направления развития архитектуры универсальных микропроцессоров 5](#_Toc106052693)

[Многопроцессорные и многомашинные вычислительные системы(30) 7](#_Toc106052694)

[Выбор микроконтроллера 8](#_Toc106052695)

[Различие между микроконтроллерами AVR, ARM, 8051 и PIC(65) 9](#_Toc106052696)

[Состав и назначение семейств PIC-контроллеров(66) 10](#_Toc106052697)

[Основные этапы разработки микропроцессорной системы на основе микроконтроллера(75) 12](#_Toc106052698)

[Методы и средства совместной отладки аппаратных и программных средств(76) 14](#_Toc106052699)

[13. Плата и распиновка Arduino Uno 15](#_Toc106052700)

[14. Широтно-импульсная модуляция 17](#_Toc106052701)

[15. Среда разработки Arduino IDE 18](#_Toc106052702)

[16.Программирование в Arduino. Цифровые выводы. Аналоговые входы. Широтно-импульсная модуляция 19](#_Toc106052703)

[17.Память в Arduino 21](#_Toc106052704)

[18.Структура программы Ардуино. 22](#_Toc106052705)

[19) Управляющие операторы. Оператор if (условие) и операторы сравнения ==, !=, < , >. 23](#_Toc106052706)

[20) Оператор for. Оператор while. Оператор do ... while 24](#_Toc106052707)

[Оператор while 24](#_Toc106052708)

[Оператор do ... while 24](#_Toc106052709)

[21) Оператор break. Оператор continue. Оператор return 24](#_Toc106052710)

[22) Синтаксис 24](#_Toc106052711)

[23) Типы данных 25](#_Toc106052712)

[24) Массивы 26](#_Toc106052713)

[25. Void. Константы. Переменные. Объявление переменных. Границы переменных 27](#_Toc106052714)

[26. Цифровой ввод/вывод. Функция pinMode. Функция digitalWrite(), Void setup(), Void loop(), Функция digitalRead() 29](#_Toc106052715)

[27. Аналоговый ввод/вывод. Функция analogRead(). Функция analogReference(), Функция analogWrite() 31](#_Toc106052716)

[28. Дополнительные фунции ввода/вывода. Функция tone(). Функция noTone(). Функция shiftOut(). Функция pulseIn() 34](#_Toc106052717)

[29. Работа со временем. Функция millis(). Функция micros(). Функция delay(). Функция delayMicroseconds() 37](#_Toc106052718)

[30. Математические функции: map() ; pow() ; sq(); sqrt (); min(); max(); abs(); constrain(); 41](#_Toc106052719)

[31. Генераторы случайных значений: randomSeed(); random() . 45](#_Toc106052720)

# Основные понятия и характеристики архитектуры микропроцессоров

Микропроцессор – программно-управляемое устройство, которое предназначено для обработки цифровой информации и управления процессом этой обработки, и выполнена в виде одной или нескольких больших интегральных схем.

БИС – функционально сложная интегральная схема.

МПС – функционально законченное изделие состоящее из одного или нескольких устройств, основу которой составляет микропроцессор.

Микропроцессор характеризуется большим количеством параметров и свойств, так как он является, с одной стороны – функционально сложным вычислительным устройством, а с другой – электронным прибором, изделием электронной промышленности.

Как средство вычислительной техники он характеризуется прежде всего: архитектурой, системой команд, типами и форматами обрабатываемых данных, режимами адресации, количеством и распределением регистров, принципами взаимодействия с оперативной памятью и внешними устройствами.

Основными характеристиками микропроцессоров являются – разрядность, тактовая частота, система команд, объем адресуемой памяти.

1. Разрядность – определяется максимальной разрядностью целочисленных данных, обрабатываемых за один такт, т.е. фактически разрядностью арифметико логического устройства. (обозначается m/n/k)
2. Тактовая частота – определяет максимальное время переключения элементов ЭВМ;
3. Виды и форматы обрабатываемых данных;
4. Система команд, режимы адресации операндов;
5. Ёмкость прямоадресуемой оперативной памяти - определяется разрядностью шиной адреса;
6. Частота внешней синхронизации (максимально возможная частота, минимально возможная частота)
7. Производительность – чем выше, тем лучше для пользователя (влияет по совокупности выше перечисленных хар-к)

# Назначение микропроцессоров

Микропроцессор выполняет функции:

1. Выборку команд программы из основной памяти;
2. Дешифрацию команд;
3. Выполнение арифметических, логических и др. операций закодированных команд;
4. Управление пересылкой информации между регистрами и основной памятью, между устройствами ввода/вывода;
5. Обработку сигналов от устройств ввода/вывода, в том числе реализацию прерывания с этих устройств;
6. Управление и координация основных узлов микропроцессора.

# Классификация микропроцессоров

Микропроцессоры могут быть:

* Универсальные (RISC, CISC);
* Однокристальные (с фиксированной разрядностью и фиксированной системой команд);
* Секционные (с наращиванием разрядности и микропрограммным управлением).

# 4. Структура микропроцессорной системы

Микропроцессорная система – сложная система, включающая в себе большое количество случайных устройств.

Основой её являются – микропроцессоры.

Связь устройств ЭВМ между собой выполняется с помощью сопряжений – интерфейсов.

*Интерфейс* – совокупность программы и аппаратных средств, предназначены для передачи информации между компонентами ЭВМ и включающие в себя электронные схемы, линии, шины и сигналы адресов, данных и управления, алгоритмы передачи сигнала и правила интерпретации сигнала устройствами.

Основным способом организации микропроцессорной системы является – магистрально-модульный принцип: все устройства, включая микропроцессор, представляются в виде модулей, которые соединяются между собой общей магистралью.

Обмен информации по магистрали удовлетворяет требования некоторого общего интерфейса, установленного для магистрали данного типа.

Каждый модуль подключается к магистрали путём специальных интерфейсных схем (они могут быть достаточно сложными. обычно они выполняются в виде специализированных микропроцессорных БИС).

Такие схемы называются – контроллерами.

*Контроллер* – обладает высокой степенью автономности, что позволяет обеспечить параллельную во времени работу периферийных устройств и выполнение программной обработки данных микропроцессоров.

Недостатки магистрально-модульного принципа – невозможность одновременного взаимодействия более 2х модулей -> ограничение производительности компьютера.

Поэтому существует 2 основных способа организации адресного пространства микропроцессорных систем:

1. С общим адресным пространством внешних устройств и оперативной памяти.
2. С независимым адресным пространством.

# 5. Назначение и функции чипсета в микропроцессорной системе

Чипсет – набор БИС (обычно от 1-3 микросхем), функционально эквивалентный микросхемам, входящим в стандартную конфигурацию микропроцессорной системы.

Чипсет интегрирует в себе функцию устройств:

* контроллера оперативной памяти
* контроллера кэш-памяти 2го и/или 3го уровня
* контроллера ПДП (прямой доступ памяти)
* контроллера приоритетных прерываний
* контроллера клавиатуры
* контроллера мыши PS/2
* контроллера инфракрасного порта
* таймера реального времени
* моста шины PCI

Обычно в составе чипсета определяют:

* Северный мост – системный контроллер, который входит в контроллер системной шины (AGP и PCI), ОЗУ и кэш-памяти.
* Южный мост – периферийный контроллер, включающий контроллеры EIDE, клавиатуры моста PCI-to-PCI, последовательных или параллельных портов, шины USB и др.

Выбор чипсета определяет конфигурацию микропроцессорной системы и её производительности. Если микропроцессор можно заменить и ёмкость ОЗУ увеличить, то замена чипсета однозначно связана с заменой системной платой.

Ограничение чипсета также однозначно ограничивает возможности замены других элементов микропроцессорной системы.

Чипсет накладывает ограничение на функциональные характеристики системы в целом.

# 6. Основные направления развития архитектуры универсальных микропроцессоров

Основными направлениями развития и повышения производительности универсальных микропроцессоров:

1. Повышение тактовой частоты;
2. Увеличение количества одновременно выполняемых команд.

*Повышение тактовой частоты,* которая в основном обеспечивается путём увеличения количества ступенек в конвейере, приводит к большим потерям времени при необходимости перезагрузки конвейера.

Такое увеличение имеет также и физические ограничения, связанные со схемотехникой кристалла БИС.

Ограничение определяется также влиянием накладных расходов при передаче частично обработанной команды на следующую ступень конвейера.

*Повышение производительности* за счёт увеличения числа конвейера в микропроцессоре можно оценить увеличением числа команд, выполняемых программами за один такт.

Увеличение этого числа команд ведёт к усложнению расположенного в БИС устройства управления*.*

Рост производительности микропроцессора не является прямо пропорциональным росту количества конвейеров (т.е. оно значительно ниже).

В настоящее время для повышения производительности микропроцессоров используются подходы:

* **CMP (Chip Multi ProcessINg)** - создание на одном кристалле системы из нескольких микропроцессоров (многоядерность);
* **SMT (Simultaneous MultiThreadINg)** - многонитевая архитектура;
* **EPIC (Explicitly Parallel INsTRuction ComputINg)** - вычисления с явным параллелизмом в командах.

1) **Направление CMP** обеспечивается возросшими технологическими возможностями, которые позволяют создать на одном кристалле несколько микропроцессоров и организовать их работу по принципу мультипроцессорных систем.

Производители чипов уже не гонятся за частотой, сместив акцент на многоядерную архитектуру, которая позволяет наращивать производительность, сохраняя в приемлемых границах энергопотребление и тепловыделение.

Многоядерные процессоры хорошо приспособлены для требовательных мультимедийных задач, таких как обработка видеозаписей, работы с большими базами данных, одновременное выполнение нескольких ресурсоемких заданий.

Так как задача повышения производительности работы отдельных приложений требует распараллеливания, то проблема перемещается с аппаратного на программный уровень.

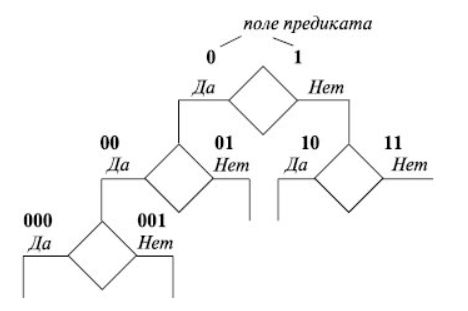
2) **Направление SMT** в развитии архитектуры микропроцессоров базируется на том, что одна задача не в состоянии полностью загрузить все возрастающие ресурсы микропроцессора. Поэтому на одном процессоре осуществляется запуск нескольких задач одновременно, при этом распараллеливание программ осуществляется аппаратными средствами МП.

3) **Направление EPIC** фактически использует известную технологию **VLIW (Very Large INsTRuction Word)** - очень длинного командного слова.

Распараллеливание алгоритма между исполнительными модулями производится компилятором на этапе создания машинного кода, когда команды объединяются в связки и не конкурируют между собой за ресурсы микропроцессора. При этом упрощается блок управления на кристалле.

Особенности архитектуры EPIC:

1. Явный параллелизм в машинном коде. Поиск зависимостей между командами проводит не процессор, а компилятор.
2. Большое количество регистров.
3. Масштабируемость архитектуры до большого количества функциональных устройств (*АЛУ*, *FPU*, *MMX*, *SSE* и т. п.).
4. Применение предикатов. Предикатный подход исходит из предпосылки, что возросшие мощности микропроцессоров позволяют запускать параллельно команды из разных ветвей условного ветвления вместо того, чтобы ожидать формирования истинных признаков для выбора правильного направления или полагаться на блок предсказания переходов, рискуя прийти к необходимости перезагрузки достаточно *длинных конвейеров* в случае неудачного предсказания. При этом каждая команда снабжается специальным полем условия (предикатом) ([рис. 11.1](https://intuit.ru/studies/courses/604/460/lecture/10341?page=1#image.11.1)). По мере определения истинных признаков ветвления те команды, предикаты которых указывали, что они выбраны из другой ветви, снимаются с обработки в конвейере. Результаты команд не записываются в прием ник до определения правильности направления перехода.



# Многопроцессорные и многомашинные вычислительные системы(30)

Существует несколько вариантов классификации систем параллельной обработки данных. По-видимому, самой ранней и наиболее известной является классификация архитектур вычислительных систем, предложенная в 1966 году М. Флинном. Классификация базируется на понятии потока, под которым понимается последовательность элементов, команд или данных, обрабатываемая процессором. На основе числа потоков команд и потоков данных выделяются четыре класса архитектур:

SISD, MISD, SIMD, MIMD.

SISD ( sINgle INsTRuction sTReam / sINgle data sTReam ) - одиночный поток команд и одиночный поток данных. К этому классу относятся прежде всего классические последовательные машины, или, иначе, машины фон-неймановского типа. В таких машинах есть только один поток команд, все команды обрабатываются последовательно друг за другом и каждая команда инициирует одну операцию с одним потоком данных. Не имеет значения тот факт, что для увеличения скорости обработки команд и скорости выполнения арифметических операций процессор может использовать конвейерную обработку. В таком понимании машины данного класса фактически не относятся к параллельным системам.

SIMD ( sINgle INsTRuction sTReam / multIPle data sTReam ) - одиночный поток команд и множественный поток данных. Применительно к одному микропроцессору этот подход реализован в MMX- и SSE- расширениях современных микропроцессоров. Микропроцессорные системы типа SIMD состоят из большого числа идентичных процессорных элементов, имеющих собственную память. Все процессорные элементы в такой машине выполняют одну и ту же программу. Это позволяет выполнять одну арифметическую операцию сразу над многими данными - элементами вектора. Очевидно, что такая система, составленная из большого числа процессоров, может обеспечить существенное повышение производительности только на тех задачах, при решении которых все процессоры могут делать одну и ту же работу.

MISD (multIPe INsTRuction sTReam / sINgle data sTReam ) - множественный поток команд и одиночный поток данных. Определение подразумевает наличие в архитектуре многих процессоров, обрабатывающих один и тот же поток данных. Ряд исследователей к данному классу относят конвейерные машины.

MIMD (multIPe INsTRuction sTReam / multIPle data sTReam ) - множественный поток команд и множественный поток данных. Базовой моделью вычислений в этом случае является совокупность независимых процессов, эпизодически обращающихся к разделяемым данным. В такой системе каждый процессорный элемент выполняет свою программу достаточно независимо от других процессорных элементов. Архитектура MIMD дает большую гибкость: при наличии адекватной поддержки со стороны аппаратных средств и программного обеспечения MIMD может работать как однопользовательская система, обеспечивая высокопроизводительную обработку данных для одной прикладной задачи, как многопрограммная машина, выполняющая множество задач параллельно, и как некоторая комбинация этих возможностей. К тому же архитектура MIMD может использовать все преимущества современной микропроцессорной технологии на основе строгого учета соотношения стоимость/производительность. В действительности практически все современные многопроцессорные системы строятся на тех же микропроцессорах, которые можно найти в персональных компьютерах, рабочих станциях и небольших однопроцессорных серверах.

Как и любая другая, приведенная выше классификация несовершенна: существуют машины, прямо в нее не попадающие, имеются также важные признаки, которые в этой классификации не учтены. Рассмотрим классификацию многопроцессорных и многомашинных систем на основе другого признака - степени разделения вычислительных ресурсов системы.

В этом случае выделяют следующие четыре класса систем:

- системы с симметричной мультипроцессорной обработкой (symmeTRic multIProcessINg), или SMP-системы;

- системы, построенные по технологии неоднородного доступа к памяти (non-un IForm memory access), или NUMA-системы;

- кластеры;

- системы вычислений с массовым параллелизмом (massively parallel processor), или MPP-системы.

# Выбор микроконтроллера

При проектировании цифровой системы необходимо осуществить правильный выбор микроконтроллера. Основная цель – выбрать наименее дорогой микроконтроллер (чтобы снизить общую стоимость системы), но в то же время удовлетворяющий спецификации системы, т. е. требованиям по производительности, надежности, условиям применения и т. д.

Основные критерии выбора микроконтроллера (в порядке значимости):

1) Пригодность для прикладной системы. Может ли она быть сделана на однокристальном микроконтроллере или ее можно реализовать на основе какой-либо специализированной микросхемы.

2) Имеет ли микроконтроллер требуемое число контактов, портов ввода-вывода, поскольку в случае их недостатка он не сможет выполнить работу, а в случае избытка цена будет слишком высокой.

3) Имеет ли микроконтроллер все требуемые периферийные устройства, такие как аналого-цифровой, цифро-аналоговый преобразователи, интерфейсы связи и т.д.

4) Имеет ли микроконтроллер другие периферийные устройства, которые не потребуются в системе (это зачастую увеличивает стоимость микроконтроллера).

5) Обеспечивает ли ядро микроконтроллера необходимую производительность, т. е. вычислительную мощность, позволяющую обрабатывать системные запросы в течение всей жизни системы на выбранном прикладном языке.

6) Выделено ли в бюджете проекта достаточно средств, чтобы позволить себе использовать данный микроконтроллер. Для ответа на этот вопрос, обычно требуются расценки поставщика. Если данный микроконтроллер не приемлем для проекта, все остальные вопросы становятся несущественными, и разработчик должен начать поиски другого микроконтроллера.

7) Доступность.

7.1. Существует ли устройство в достаточных количествах.

7.2. Производится ли оно сейчас.

7.3. Что ожидается в будущем.

7.4. Поддержка разработчика.

7.5. Ассемблеры.

7.6. Компиляторы.

7.7. Средства отладки.

7.8. Внутрисхемные эмуляторы.

7.9. Информационная поддержка

7.9.1. Примеры применения.

7.9.2. Сообщения об ошибках.

7.9.3. Утилиты, в том числе бесплатные ассемблеры.

7.9.4. Примеры исходных текстов.

7.9.5. Поддержка применений у поставщика.

7.9.6. Квалификация поддерживающего персонала, действительно ли он заинтересован в помощи при решении вашей проблемы.

7.9.7. Связь с поддерживающим профессионалом.

7.10. Надежность фирмы производителя.

7.10.1. Компетентность, подтвержденная разработками.

7.10.2. Надежность производства, т.е. качество продукции.

7.10.3. Время работы в этой области.

# Различие между микроконтроллерами AVR, ARM, 8051 и PIC(65)

**Микроконтроллер 8051**– это восьмибитное семейство микроконтроллеров, имеет 128 байт оперативной памяти, 4 Кбайт ПЗУ, два таймера, один последовательный порт и четыре порта на одном кристалле. Процессор может обрабатывать до восьми бит данных одновременно. Если данные больше восьми бит, то они должны быть разбиты на части, чтобы процессор мог легко их обрабатывать. Большинство микроконтроллеров серии 8051 различных производителей содержат 4 Кбайт ПЗУ, хотя объем ПЗУ может быть расширен до 64 Кбайт.

**Контроллер периферийного интерфейса (PIC)–** Микроконтроллер PIC быстрее и проще реализует программы, если сравнивать с другими микроконтроллерами, такими как 8051. Простота программирования и простота взаимодействия с другими периферийными устройствами делает PIC более успешным микроконтроллером.

Микроконтроллеры AVR доступны в трех основных подсемействах:

1) TinyAVR: меньше памяти, небольшой размер, подходит только для более простых приложений.

2) MegaAVR: это популярные микроконтроллеры, в основном имеющие относительно большое количество памяти (до 256 КБ), большее количество встроенных периферийных устройств и подходят для довольно сложных приложений.

3) XmegaAVR: используются в коммерческих приложениях для решения сложных задач, которым требуется большая память программ и высокая скорость.

Микроконтроллеры с ядром **ARM** также являются одним из семейств процессоров на базе архитектуры RISC.

Микроконтроллеры ARM основаны на тридцатидвух битных и 6четырехбитных многоядерных процессорах RISC. Процессоры RISC предназначены для выполнения меньшего количества инструкций, чтобы они могли работать с большей скоростью, выполняя дополнительные миллионы инструкций в секунду (MIPS). Устраняя ненужные инструкции и оптимизируя обработку информации, RISC-процессоры обеспечивают большую производительность по сравнению с большинством рассмотренных выше микроконтроллеров.

Основные различия между микроконтроллерами:



# Состав и назначение семейств PIC-контроллеров(66)

Микроконтроллеры семейств PIC (Peripheral Interface Controller) компании Microchip объединяют все передовые технологии микроконтроллеров: электрически программируемые пользователем ППЗУ, минимальное энергопотребление, высокую производительность, хорошо развитую RISC-архитектуру, функциональную законченность и минимальные размеры. Широкая номенклатура изделий обеспечивает использование микроконтроллеров в устройствах, предназначенных для разнообразных сфер применения.

Первые микроконтроллеры компании Microchip PIC16C5x появились в конце 1980-х годов и благодаря своей высокой производительности и низкой стоимости составили серьезную конкуренцию производившимся в то время восьмиразрядным МК с CISC-архитектурой.

Высокая скорость выполнения команд в PIC -контроллерах достигается за счет использования двухшинной гарвардской архитектуры вместо традиционной одношинной фон-неймановской. Гарвардская архитектура основывается на наборе регистров с разделенными шинами и адресными пространствами для команд и данных. Все ресурсы микроконтроллера, такие как порты ввода/вывода, ячейки памяти и таймер, представляют собой физически реализованные аппаратные регистры .

Микроконтроллеры PIC содержат RISC-процессор с симметричной системой команд, позволяющей выполнять операции с любым регистром, используя произвольный метод адресации. Пользователь может сохранять результат операции в самом регистре -аккумуляторе или во втором регистре, используемом для операции.

В настоящее время компания Microchip выпускает пять основных семейств восьмиразрядных RISC-микроконтроллеров, совместимых снизу вверх по программному коду:

PIC12CXXX – семейство микроконтроллеров, выпускаемых в миниатюрном восьмивыводном исполнении. Эти микроконтроллеры выпускаются как с двенадцатиразрядной (33 команды), так и с 14-разрядной (35 команд) системой команд. Содержат встроенный тактовый генератор, таймер /счетчик, сторожевой таймер, схему управления прерываниями. В составе семейства есть микроконтроллеры со встроенным восьмиразрядным четырехканальным АЦП. Способны работать при напряжении питания до 2,5 В;

PIC16C5X – базовое семейство микроконтроллеров с двенадцатиразрядными командами (33 команды), выпускаемое в 18, 20- и 28-выводных корпусах. Представляют собой простые недорогие микроконтроллеры с минимальной периферией. Способность работать при малом напряжении питания (до 2 В) делает их удобными для применения в переносных конструкциях. В состав семейства входят микроконтроллеры подгруппы PIC16HV5XX, способные работать непосредственно от батареи в диапазоне питающих напряжений до 15 В;

PIC16CXXX – семейство микроконтроллеров среднего уровня с 14‑разрядными командами (35 команд). Наиболее многочисленное семейство, объединяющее микроконтроллеры с разнообразными периферийными устройствами, в число которых входят аналоговые компараторы, аналогово-цифровые преобразователи, контроллеры последовательных интерфейсов SPI, USART и I2C, таймеры-счетчики, модули захвата/сравнения, широтно-импульсные модуляторы, сторожевые таймеры, супервизорные схемы и так далее;

PIC17CXXX – семейство высокопроизводительных микроконтроллеров с расширенной системой команд шестнадцатиразрядного формата (58 команд), работающие на частоте до 33 МГц, с объемом памяти программ до 16 Кслов. Кроме обширной периферии, шестнадцатиуровневого аппаратного стека и векторной системы прерываний, почти все микроконтроллеры этого семейства имеют встроенный аппаратный умножитель 8х8, выполняющий операцию умножения за один машинный цикл. Являются одними из самых быстродействующих в классе восьмиразрядных микроконтроллеров;

PIC18CXXX – семейство высокопроизводительных микроконтроллеров с расширенной системой команд шестнадцатиразрядного формата (75 команд) и встроенным 10-разрядным АЦП, работающие на частоте до 40 МГц. Содержат 31-уровневый аппаратный стек, встроенную память команд до 32 Кслов и способны адресовать до 4 Кбайт памяти данных и до 2 Мбайт внешней памяти программ. Расширенное RISC-ядро микроконтроллеров данного семейства оптимизировано под использование нового Си-компилятора.

Большинство PIC -контроллеров выпускаются с однократно программируемой памятью программ (OTP), с возможностью внутрисхемного программирования или масочным ПЗУ. Для целей отладки предлагаются более дорогие версии с ультрафиолетовым стиранием и Flash-памятью. Полный список выпускаемых модификаций PIC -контроллеров включает порядка пятисот наименований. Поэтому продукция компании перекрывает почти весь диапазон применений восьмиразрядных микроконтроллеров.

Из программных средств отладки наиболее известны и доступны различные версии ассемблеров, а также интегрированная программная среда MPLAB. Российские производители программаторов и аппаратных отладочных средств также уделяют внимание PIC -контроллерам. Выпускаются как специализированные программаторы, такие как PICPROG, программирующие почти весь спектр PIC -микроконтроллеров, так и универсальные: UNIPRO и СТЕРХ, поддерживающие наиболее известные версии PIC -контроллеров.

Наиболее распространенными семействами PIC -контроллеров являются PIC16CXXX и PIC17CXXX.

# Основные этапы разработки микропроцессорной системы на основе микроконтроллера(75)

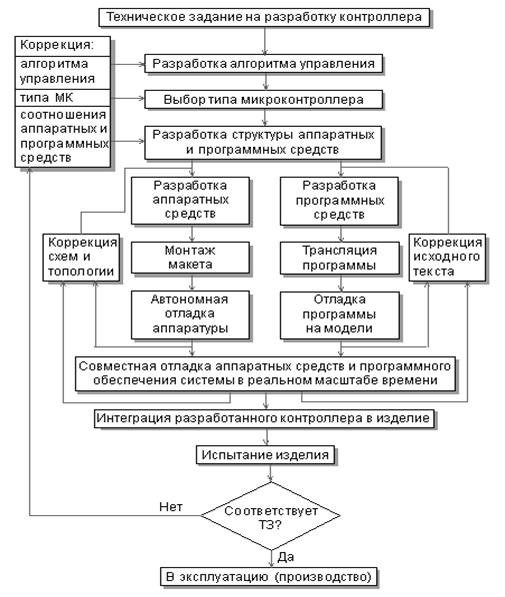
Технология проектирования контроллеров на базе МК полностью соответствует принципу неразрывного проектирования и отладки аппаратных и программных средств, принятому в микропроцессорной технике. Это означает, что перед разработчиком такого рода МПС стоит задача реализации полного цикла проектирования, начиная от разработки алгоритма функционирования и заканчивая комплексными испытаниями в составе изделия, а, возможно, и сопровождением при производстве. Сложившаяся к настоящему времени методология проектирования контроллеров может быть представлена так, как показано на рисунке 42.1.

В техническом задании формулируются требования к контроллеру с точки зрения реализации определенной функции управления. Техническое задание включает в себя набор требований, который определяет, что пользователь хочет от контроллера и что разрабатываемый прибор должен делать. Техническое задание может иметь вид текстового описания, не свободного в общем случае от внутренних противоречий.

На основании требований пользователя составляется функциональная спецификация, которая определяет функции, выполняемые контроллером для пользователя после завершения проектирования, уточняя тем самым, насколько устройство соответствует предъявляемым требованиям. Она включает в себя описания форматов данных, как на входе, так и на выходе, а также внешние условия, управляющие действиями контроллера.

Функциональная спецификация и требования пользователя являются критериями оценки функционирования контролера после завершения проектирования. Может потребоваться проведение нескольких итераций, включающих обсуждение требований и функциональной спецификации с потенциальными пользователями контроллера, и соответствующую коррекцию требований и спецификации. Требования к типу используемого МК формулируются на данном этапе чаще всего в неявном виде.

Этап разработки алгоритма управления является наиболее ответственным, поскольку ошибки данного этапа обычно обнаруживаются только при испытаниях законченного изделия и приводят к необходимости дорогостоящей переработки всего устройства. Разработка алгоритма обычно сводится к выбору одного из нескольких возможных вариантов алгоритмов, отличающихся соотношением объема программного обеспечения и аппаратных средств.



Для реализации на практике возможности выбора оптимального МК необходима достаточно глубокая проработка алгоритма управления, оценка объема исполняемой программы и числа линий сопряжения с объектом на этапе выбора МК. Допущенные на данном этапе просчеты могут впоследствии привести к необходимости смены модели МК и повторной разводки печатной платы макета контроллера. В таких условиях целесообразно выполнять предварительное моделирование основных элементов прикладной программы с использованием программно-логической модели выбранного МК.

При отсутствии МК, обеспечивающего требуемые по ТЗ характеристики проектируемого контроллера, необходим возврат к этапу разработки алгоритма управления и пересмотр выбранного соотношения между объемом программного обеспечения и аппаратных средств. Отсутствие подходящего МК чаще всего означает, что для реализации необходимого объема вычислений (алгоритмов управления) за отведенное время нужна дополнительная аппаратная поддержка. Отрицательный результат поиска МК с требуемыми характеристиками может быть связан также с необходимостью обслуживания большого числа объектов управления. В этом случае возможно использование внешних схем обрамления МК.

На этапе разработки структуры контроллера окончательно определяется состав имеющихся и подлежащих разработке аппаратных модулей, протоколы обмена между модулями, типы разъемов. Выполняется предварительная проработка конструкции контроллера. В части программного обеспечения определяются состав и связи программных модулей, язык программирования. На этом же этапе осуществляется выбор средств проектирования и отладки.

# Методы и средства совместной отладки аппаратных и программных средств(76)

Этап совместной отладки аппаратных и программных средств в реальном масштабе времени является самым трудоемким и требует использования инструментальных средств отладки. К числу основных инструментальных средств отладки относятся:

- внутрисхемные эмуляторы;

- платы развития (оценочные платы);

- мониторы отладки ;

- эмуляторы ПЗУ.

Внутрисхемный эмулятор – программно-аппаратное средство, способное заменить эмулируемый МК в реальной схеме. Стыковка внутрисхемного эмулятора с отлаживаемой системой производится при помощи кабеля со специальной эмуляционной головкой, которая вставляется вместо МК в отлаживаемую систему. Если МК нельзя удалить из отлаживаемой системы, то использование эмулятора возможно, только если этот микроконтроллер имеет отладочный режим, при котором все его выводы находятся в третьем состоянии. В этом случае для подключения эмулятора используют специальный адаптер-клипсу, который подключается непосредственно к выводам эмулируемого МК.

Платы развития, или, как принято их называть в зарубежной литературе, оценочные платы (Evaluation Boards), являются своего рода конструкторами для макетирования электронных устройств. Обычно это печатная плата с установленным на ней МК и всей необходимой ему стандартной периферией. На этой плате также устанавливают схемы связи с внешним компьютером.

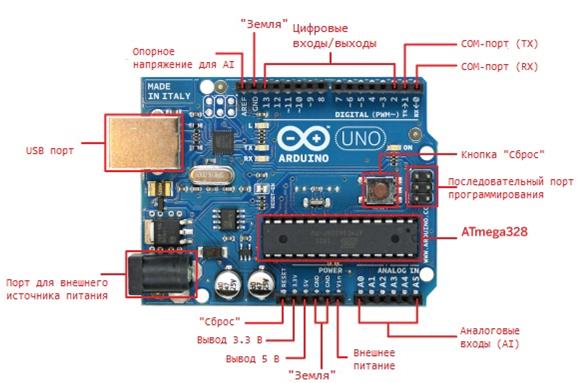
Для большего удобства платы развития комплектуются еще и простейшим средством отладки на базе монитора отладки. Используются два типа мониторов отладки: один для МК, имеющих внешнюю шину, а второй – для МК, не имеющих внешней шины.

В первом случае отладочный монитор поставляется в виде микросхемы ПЗУ, которая вставляется в специальную розетку на плате развития. Плата также имеет ОЗУ для программ пользователя и канал связи с внешним компьютером или терминалом. Во втором случае плата развития имеет встроенные схемы программирования внутреннего ПЗУ МК, которые управляются от внешнего компьютера. При этом программа монитора просто заносится в ПЗУ МК совместно с прикладными кодами пользователя. Прикладная программа должна быть специально подготовлена: в нужные места необходимо вставить вызовы отладочных подпрограмм монитора.

Эмулятор ПЗУ – программно-аппаратное средство, позволяющее замещать ПЗУ на отлаживаемой плате, и подставляющее вместо него ОЗУ, в которое может быть загружена программа с компьютера через один из стандартных каналов связи. Это устройство позволяет пользователю избежать многократных циклов перепрограммирования ПЗУ. Эмулятор ПЗУ нужен только для МК, которые могут обращаться к внешней памяти программ. Это устройство сравнимо по сложности и по стоимости с платами развития и имеет одно большое достоинство: универсальность. Эмулятор ПЗУ может работать с любыми типами МК.

# 13. Плата и распиновка Arduino Uno

Arduino/Genuino Uno-это плата микроконтроллера на базе ATmega328P (datasheet). Он имеет 14 цифровых входов/выходов (из которых 6 могут использоваться в качестве ШИМ-выходов), 6 аналоговых входов, кварцевый кристалл 16 МГц, USB-соединение, разъем питания, заголовок ICSP и кнопку сброса. Arduino UNO самая функциональная плата со стандартным набором узлов. Очень удобна при подключении к ней уже готовых узлов и плат расширения.



**Микроконтроллер ATmega328P**

Сердцем платформы Arduino Uno является восьмибитный микроконтроллер семейства AVR — ATmega328P.

**Микроконтроллер ATmega16U2**

Микроконтроллер ATmega16U2 обеспечивает связь микроконтроллера ATmega328P с USB-портом компьютера. При подключении к ПК Arduino Uno определяется как виртуальный COM-порт. Прошивка микросхемы 16U2 использует стандартные драйвера USB-COM, поэтому установка внешних драйверов не требуется.

Распиновка выглядит следующим образом☹(КАРТИНКА ВНИЗУ)

1) Последовательный интерфейс использует шины №0 (RX – получение данных), №1 (TX – передача данных).

2) Для внешнего прерывания используются выводы №2, №3.

3) Для ШИМ используются выводы за номерами 3,5, 6, 9, 10, 11. Функция analog Write обеспечивает разрешение в 8 бит.

4) Связь посредством SPI: контакты №10 (SS), №11 (MOSI), №12 (MISO), №13 (SCK).

5) Вывод №13 запитывает светодиод, который загорается при высоком потенциале.

6) Uno оснащена 6 аналоговыми входами (A0 – A5), которые имеют разрешение в 10 бит.

7) Для изменения верхнего предела напряжения используется вывод AREF (функция analog Reference).

8) Связь I2C (TWI, библиотека Wire) осуществляется через выводы №4 (SDA), №5 (SCL).

9) Вывод Reset – перезагрузка микроконтроллера.

**Пины питания**

**VIN**: Напряжение от внешнего источника питания (не связано с 5 В от USB или другим стабилизированным напряжением). Через этот вывод можно как подавать внешнее питание, так и потреблять ток, если к устройству подключён внешний адаптер.

**5V**: На вывод поступает напряжение 5 В от стабилизатора платы. Данный стабилизатор обеспечивает питание микроконтроллера ATmega328. Запитывать устройство через вывод 5V не рекомендуется — в этом случае не используется стабилизатор напряжения, что может привести к выходу платы из строя.

**3.3V**: 3,3 В от стабилизатора платы. Максимальный ток вывода — 50 мА.

**GND**: Выводы земли.

**IOREF:** Вывод предоставляет платам расширения информацию о рабочем напряжении микроконтроллера. В зависимости от напряжения, плата расширения может переключиться на соответствующий источник питания либо задействовать преобразователи уровней, что позволит ей работать как с 5 В, так и с 3,3 В устройствами.

**Порты ввода/вывода**

Цифровые входы/выходы: пины 0–13 – изображены на рисунке 44.5

Логический уровень единицы — 5 В, нуля — 0 В.

Максимальный ток выхода — 40 мА. К контактам подключены подтягивающие резисторы, которые по умолчанию выключены, но могут быть включены программно.

На плате выведены 14 цифровых пинов (контактов), любой из которых может работать как на вывод информации, так и на ввод. Для этого в коде программ применяются специальные функции: ХЗ, надо ли это

pinMode()

Функция pinMode служит для задания режима работы контакта, будет-ли он работать на выход или на вход. В данной функции задается номер контакта, которым мы в дальнейшем собираемся управлять.

digitalRead()

Функция считывает текущее значение с заданного контакта – его значение может быть HIGH или LOW.

digitalWrite()

Функция передает определенное значение на заданный контакт – оно может быть HIGH или LOW.

Все выводы обладают пятивольтовой логикой, то есть выдают логическую единицу как напряжение 5В.

Каждый вывод платы имеет нагрузочный резистор номиналом 20-50 кОм и может пропускать до 40 мА, но по умолчанию все они отключены.

Стандартная конфигурация выводов микроконтроллера позволяет не использовать функцию pinMode(), так как они изначально настроены в качестве портов ввода. Их особенностью является минимальная нагрузка на схему: порты эквивалентны внутреннему резистору, что гарантирует сопротивление в 100 МОм.

Для трансформации порта в один из двух режимов достаточно минимум тока. Благодаря этому, выходы используют для подключения датчиков касания, фотодиодов и других датчиков, имеющих схему аналогичной RC-цепи.

Если к определенному выводу ничего не подключено, то из-за помех, случайных контактов на нем могут возникать случайные величины.

Рекомендуется задавать портам, к которым ничего не подключено, определенное известное значение. Для этого используют подтягивающие резисторы 10 кОм, подключая вход к +5В или к земле.

# 14. Широтно-импульсная модуляция

Для вывода данных с Arduino традиционно применяется двоичная система с применением логических значений 0 и 1. С этой задачей превосходно справляется управление портами вывода, имеющееся у большинства микроконтроллеров от производителей микросхем. Но не всегда схемотехнику необходимо значение нуля или единицы.

С помощью PWM (ШИМ), изображенного на рисунке 44.6, позволяет создать сигнал, который имеет определенное напряжение и продолжительность. В зависимости от временного промежутка между импульсами на выходе выдается нужное значение, так называемый параметр скважности импульса. **Скважность** – это один из классификационных признаков импульсных систем, определяющий отношение его периода следования (повторения) к длительности импульса.

Микроконтроллер Arduino поддерживает восьмибитную ШИМ, что позволяет выбрать переменную в широком диапазоне значений от 0 до 256.

В качестве примера использования ШИМ можно использовать плавное включение и выключение светодиода, подключенному к 3 пину. При этом постепенно подается напряжение от 0 до 5В. Как видно на картинке ниже, при разном значении переменной мы получаем разное напряжение для светодиода: 5V, 2.5V, 1,25V, 3.75V.

Так же с помощью пьезоизлучателя и ШИМ можно воспроизводить звуки неплохого качества.

**АЦП:** пины A0–A5

АНАЛОГОВЫЕ ВХОДЫ: КОНТАКТЫ A0, A1, A2, A3, A4, A5

6 аналоговых входов, каждый из которых может представить аналоговое напряжение в виде 10-битного числа (1024 значений). Разрядность АЦП — 10 бит.

Arduino Uno имеет на своей платформе 6 аналоговых входов с разрешением 10 Бит на каждый вход. Данное разрешение говорит нам о том, что сигнал, приходящий на него, можно оцифровать в диапазоне от 0 до 1024 условных значений.

Считывать значения с данных контактов можно функцией analogRead(), а передавать значения – функцией analogWrite().

Так как Arduino Uno обладает пятивольтовой логикой, то и значение будет находиться в диапазоне от 0 до 5 вольт, однако при помощи функции analogReference() можно изменять верхний предел.

**TWI/I²C:** пины SDA и SCL

Для общения с периферией по синхронному протоколу, через 2 провода. При помощи данных контактов к Arduino можно подключать внешние цифровые устройства, умеющие общаться по протоколу I2C. Для реализации интерфейса в среде Arduino IDE присутствует библиотека Wire.

**SPI:** пины 10(SS), 11(MOSI), 12(MISO), 13(SCK).

Через эти пины осуществляется связь по интерфейсу SPI. Для работы — используйте библиотеку SPI. С помощью данных контактов происходит подключение периферии, работающей через интерфейс SPI.

(И ИНФА В ВОПРОСЕ 4)

# 15. Среда разработки Arduino IDE

Arduino – контроллер является перепрограммируемым и настраиваемым на конкретную задачу пользователя.

Среда разработки Arduino IDE состоит из набора специфичных библиотек и драйверов, необходимых для корректной работы с контроллером. Установив программу и запустив ее, будет интерфейс, похожий на простой текстовый блокнот, но с подсветкой синтаксиса. В этот текстовый блокнот и пишется программа для контроллера.

С помощью встроенных библиотек возможно подключить дополнительный датчик или дисплей к контроллеру. Имеется возможность проверить код на наличие ошибок. Программа укажет точное место ошибки в коде или где были введены некорректные данные. Если их не исправить, то программа просто не сможет скомпилировать код, и не удастся залить скетч в контроллер.

Прежде чем заливать программу в контроллер, необходимо выбрать порт подключения и модель платы Arduino. Если используется программатор, то на вкладке нужно выбрать способ закачки программы через данное устройство.

Arduino IDE имеет предустановленные скетчи для демонстрации возможности устройства. Это помогает удостовериться, что плата не повреждена и что на нее можно залить программу. Кроме того, базовые примеры позволят понять принцип программирования. В базовые примеры входят простые скетчи управления выводами.

Самый элементарный скетч позволяет управлять светодиодом, расположенным на плате. Можно подкорректировать данные и заставить светодиод моргать чаще или медленнее, или задать, например, такт мигания каждые две секунды и при этом сделать само горение светодиода на полсекунды. Также можно посмотреть работу программы аналоговых и цифровых выводов, простых и сенсорных дисплеев, сделать проверку различных звуковых элементов и т.д.

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.

Программа, написанная в среде Arduino, называется скетч. 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

# 16.Программирование в Arduino. Цифровые выводы. Аналоговые входы. Широтно-импульсная модуляция

Цифровые выводы

Выводы платформы Arduino могут работать как входы или как выходы. Также ана­логовые входы Arduino (ATmega) могут конфигурироваться и работать так же, как и цифровые порты ввода/вывода.

Выводы Arduino настроены как порты ввода, поэтому не требуется декларации в функции pinMode(). Сконфигурированные порты ввода находятся в высокоимпедансном состоянии. Это означает, что порт ввода дает слишком малую нагрузку на схему, в которую он включен. Для перевода порта ввода из одного состояния в дру­гое требуется маленькое значение тока. Если к выводу ничего не подключено, то значения на нем будут принимать случайные величины, наводимые электрически­ми помехами.

Если на порт ввода не поступает сигнал, то рекомендуется задать порту известное состояние. Это делается добавлением подтягивающих резисторов 10 кОм, подклю­чающих вход либо к питанию +5 В, либо к земле.

Микроконтроллер ATmega имеет программируемые встроенные подтягивающие резисторы 20 кОм. Программирование данных резисторов осуществляется так:

pinMode(pin, INPUT); // назначить выводу порт ввода

digitalWrite(pin, HIGH); // включить подтягивающий резистор

Выводы, сконфигурированные как порты вывода находятся в низкоимпедансном состоянии. Данные выводы могут пропускать через себя достаточно большой ток. Выводы микросхемы ATmega могут быть источником тока до 40 мА. Такого значе­ния тока 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.

**Аналоговые входы**

Микроконтроллеры ATmega, используемые в Arduino, содержат шестиканальный аналого-цифровой преобразователь (АЦП). Разрешение преобразователя составляет 10 битов, что позволяет на выходе получать значения от 0 до 1023.

Аналоговые входы могут использоваться как цифровые выводы портов ввода/вывода, при этом они имеют номера от 14 до 19:

pinMode(14,OUTPUT); digitalWrite(14, HIGH);

Для вывода, работавшего ранее как цифровой порт вывода, команда analogRead будет работать некорректно. В этом случае рекомендуется сконфигурировать его как аналоговый вход.

**Широтно-импульсная модуляция**

Широтно-импульсная модуляция (ШИМ) — это операция получения изменяюще­гося аналогового значения посредством цифровых устройств. Подавая на выход сигнал, состоящий из высоких и низких уровней, мы моделируем напряжение между максимальным значением (5 В) и минимальным (0 В). Длительность включения максимального значения называется шириной импульса. Для получения различных аналоговых величин изменяется ширина импульса. В результате на выходе будет величина напряжения, равная площади под импульсами, показанные на рисунке

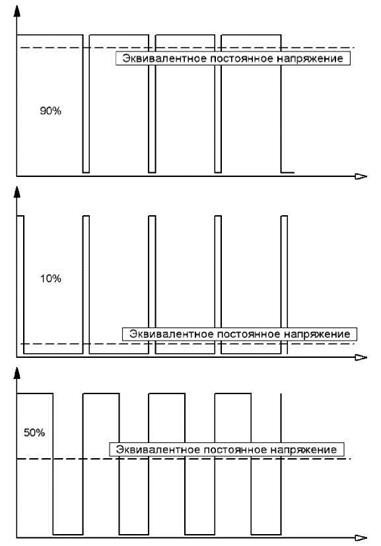


Рисунок 46.1 – Широтно-импульсная модуляция

Вызов функции analogWrite() с масштабом 0-255 означает, что значение analogWrite(255) будет соответствовать 5 В (100 % рабочий цикл — постоянное включение 5 В), а значение analogWrite(127) — 2,5 В (50 % рабочий цикл).

# 17.Память в Arduino

В микроконтроллерах ATmega168, ATmega328, ATmega1280, ATmega2560, исполь­зуемых на платформах Arduino, существует три вида памяти:

- флеш-память — используется для хранения скетчей;

- ОЗУ (статическая оперативная память) — служит для хранения и работы пере­менных;

- EEPROM (энергонезависимая память) — применяется для хранения постоянной информации.

Флеш-память и EEPROM являются энергонезависимыми видами памяти (дан­ные сохраняются при отключении питания). ОЗУ является энергозависимой па­мятью.

Микроконтроллер ATmega168 имеет:

-  16 Кбайт флеш-памяти (2 Кбайт используется для хранения загрузчика);

-  1024 байта ОЗУ;

-  512 байт EEPROM.

Для ATmega328 эти показатели следующие:

-  32 Кбайт флеш-памяти (2 Кбайт используется для хранения загрузчика);

-  2 Кбайт ОЗУ;

-  1024 байт EEPROM.

Для ATmega1280 эти показатели следующие:

-  128 Кбайт флеш-памяти (2 Кбайт используется для хранения загрузчика);

- 8 Кбайт ОЗУ;

-  4096 байт EEPROM.

Для ATmega2560 эти показатели следующие:

-  256 Кбайт флеш-памяти (2 Кбайт используется для хранения загрузчика);

-  16 Кбайт ОЗУ;

-  9182 байт EEPROM.

При отсутствии свободного места в ОЗУ могут произойти сбои программы.

# 18.Структура программы Ардуино.

Структура программы Ардуино состоит из двух частей setup() и loop().

**void setup()** {

// код выполняется один раз при запуске программы

**}**

**void loop() {**

// основной код, выполняется в цикле

**}**

Функция setup() выполняется один раз, при включении питания или сбросе контроллера. Функция должна присутствовать в программе, даже если в ней ничего нет. Перед функцией setup() идет объяв­ление переменных, подключение библиотек. Она используется для инициализации переменных, установки режима работы портов и прочих подгото­вительных для основного цикла программы действий.

После завершения setup() управление переходит к функции loop(). Она в бесконечном цикле выполняет команды, записанные в ее теле (между фигурными скобками). Собственно эти команды и совершают все алгоритмические действия контроллера.

Пример простейшей программы.

**Void setup()**

{

Serial.begin(9600);

}

**Void loop()**

{

Serial.println(millis()); delay(1000);

}

# 19) Управляющие операторы. Оператор if (условие) и операторы сравнения ==, !=, < , >.

**Оператор if.**

Оператор if используется в сочетании с операторами сравнения, он проверяет, дос­тигнута ли истинность условия

**Операторы сравнения:**

- x == y (x равно y);

- x != y (x не равно y);

- x < y (x меньше чем y);

- x > y (x больше чем y);

- x <= y (x меньше чем или равно y);

- x >= y (x больше чем или равно y).

**Оператор if..else**

Конструкция if..else предоставляет больший контроль над процессом выполнения кода, чем базовый оператор if, позволяет сделать выбор "либо, либо".

Например:

if (pinInput==HIGH)

{doFun1();}

else

{doFun2();}

Else позволяет делать отличную от указанной в if проверку, чтобы можно было осуществлять сразу несколько взаимоисключающих проверок. Каждая проверка позволяет переходить к следующему за ней оператору не раньше, чем получит ло­гический результат ИСТИНА.

# 20) Оператор for. Оператор while. Оператор do ... while

Конструкция for используется для повторения блока операторов, заключенных в фигурные скобки. Счетчик приращений обычно используется для приращения и завершения цикла. Оператор for подходит для любых повторяющихся действий и часто используется в сочетании с массивами коллекций данных/выводов.

Заголовок цикла for состоит из трех частей:

for (initialization; condition; increment) {операторы, выполняющиеся в цикле}

Инициализация (initialization) выполняется самой первой и один раз. Каждый раз в цикле проверяется условие (condition), если оно верно, выполняется блок опера­торов и приращение (increment), затем условие проверяется вновь. Когда логи­ческое значение условия становится ложным, цикл завершается.

# Оператор while

Оператор while будет вычислять 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. Это изменение может происходить как в программном коде, например, при увеличении переменной, так и во внешних условиях, напри­мер, при тестировании датчика.

# Оператор do ... while

Цикл do 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.

# 21) Оператор break. Оператор continue. Оператор return

**Оператор break**

Оператор break используется для принудительного выхода из циклов do, for или while, не дожидаясь завершения цикла по условию. Он также используется для вы­хода из оператора switch.

**Оператор continue**

Оператор continue пропускает оставшиеся операторы в текущем шаге цикла. Вместо них выполняется проверка условного выражения цикла, которая происхо­дит при каждой следующей итерации.

**Оператор return**

Оператор return прекращает вычисления в функции и возвращает значение из пре­рванной функции в вызывающую, если это нужно.

# 22) Синтаксис

**;** используется для обозначения конца оператора.

**{}** важный элемент языка программирования С. Открывающая скобка { должна всегда сопровождаться закрывающей скобкой }. Это условие, из­вестное как парность (симметричность) фигурных скобок.

Основные способы использования фигурных скобок:

а) функции:

- void НазваниеФункции (тип данных аргумента) { оператор(ы) };

б)  циклы:

- while (логическое выражение){ оператор(ы)};

- do { оператор(ы)} while (логическое выражение);

- for (инициализация; условие окончания цикла; приращения цикла)

- { оператор(ы)};

в)  условные операторы:

-  if (логическое выражение) {оператор(ы)}.

**//** это строки в программе, которые используются для информирова­ния вас самих или других о том, как работает программа. Они игнорируются ком­пилятором и не занимают место в памяти микроконтроллера.

Есть два способа пометить строку как комментарий:

 однострочный комментарий — // ;

 многострочный комментарий — /\* ... \*/ .

# 23) Типы данных

**boolean**

Логический (булевый) тип данных — boolean. Может принимать одно из двух зна­чений: true или false. Данные типа boolean занимают в памяти один байт.

**char**

Переменная типа char занимает 1 байт памяти и может хранить один алфавитно­цифровой символ (литеру). При объявлении литеры используются одиночные ка­вычки: 'A' .Символ хранится в памяти как число(код ASCII) Так как символ хранится как число, в памяти над ним возможно производить арифметические действия (например, 'A' + 1 будет 66, т. к. ASCII код для 'A' — 65).

Тип char знаковый тип, т. е. число (код), хранящийся в памяти, может принимать значения от -128 до 127

**Byte**

Хранит восьмибитовое числовое значение без десятичной точки. Имеет диапазон от 0 до 255.

**int**

Тип данных int— один из наиболее часто исполь­зуемых типов данных для хранения чисел. int занимает 2 байта памяти и может хранить числа от -32 768 до 32 767.

Для размещения отрицательных значений int использует так называемый допол­нительный код представления числа. Старший бит указывает на отрицательный знак числа, остальные биты инвертируются с добавлением 1.

**Unsigned int**

Тип данных unsigned int — беззнаковое целое число, так же как и тип int (знако­вое), занимает в памяти 2 байта. Но в отличие от int, тип unsigned int может хра­нить только положительные целые числа в диапазоне от 0 до 65 535.

Отличие кроется в том, как unsigned int использует старший бит, иногда называе­мый знаковым битом. Если старший бит равен 1, то для типа int компилятор Arduino считает, что это число отрицательное, а остальные 15 битов несут инфор­мацию о модуле целого числа в дополнительном коде представления числа, в то время как unsigned int использует все 16 битов для хранения модуля числа.

Когда переменная типа unsigned int вследствие арифметической операции дости­гает своего максимального значения, она "перескакивает" на самое минимальное значение и наоборот.

**Long**

Тип данных long используется для хранения целых чисел в расширенном диапазоне от -2 147 483 648 до 2 147 483 647. long занимает 4 байта в памяти.

**unsigned long**

Unsigned long используется для хранения положительных целых чисел в диапазоне от 0 до 4 294 967 295 и занимает 32 бита (4 байта) в памяти.

**Float**

Тип данных float служит для хранения чисел с плавающей запятой. Этот тип часто используется для операций с данными, считываемыми с аналоговых входов. Диапа­зон значений — от -3,4028235E+38 до 3,4028235E+38. Переменная типа float за­нимает 32 бита (4 байта) в памяти.

Тип float имеет точность 6-7 знаков, имеются в виду все знаки, а не только ман­тисса.

**Double**

Тип данных double, в отличие от большинства языков программирования, имеет ту же точность, что и тип float и занимает также 4 байта памяти.

Тип double поддерживается в Arduino для совместимости кода с другими платфор­мами.

**string**

Текстовые строки в Arduino объявляются как массив (array) типа char (символов, литер), оканчивающийся символом "конца строки".

# 24) Массивы

Массивы (arrays) — именованный набор однотипных переменных с доступом к отдельным элементам по их индексу. Существует несколько вариантов объявле­ния массива:

- массив может быть объявлен без непосредственной инициализации элементов массива:

int mylnts[6];

- массив может быть объявлен без явного задания размера. Компилятор сам по­считает фактическое количество элементов и создаст в памяти массив необхо­димого размера:

int myPins[] = {2, 4, 8, 3, 6};

- при объявлении массива размер может быть задан явно, одновременно с ини­циализацией элементов массива, при создании массива типа char необходим дополнительный элемент массива для нулевого символа:

int mySensVals[6] = {2, 4, -8, 3, 2};

char message[6] = "hello";

Индексация массива начинается с 0. Присваивание значения элементу массива происходит следующим образом:

mySensVals[0] = 10;

Получение значения массива: x = mySensVals[4];

Чаще всего для перебора элементов цикла используется цикл for, счетчик цикла используется как индекс для доступа к каждому элементу массива.

# 25. Void. Константы. Переменные. Объявление переменных. Границы переменных

**Void**

Ключевое слово void используется при объявлении функций, если функция не воз­вращает никакого значения при ее вызове.

**Константы**

Константы — предопределенные значения. Они используются, чтобы делать про­граммы более легкими для чтения.

**true/false** это булевы константы, определяющие логические уровни. false лег­ко определяется как 0 (ноль), а true, как 1, но может быть и чем-то другим, отлич­ным от нуля. Поэтому -1, 2 и 200 — это все тоже определяется как true.

#define true 0x1

#define false 0x0

**high/low**— уровни сигналов порта high и low:

#define HIGH 0x1

#define LOW 0x0

**input/output** — настройка цифровых портов на ввод (input) и вывод (output) сиг­налов:

#define INPUT 0x0

#define OUTPUT 0x1

Цифровые порты могут использоваться на ввод или вывод сигналов. Изменение порта с ввода на вывод производится при помощи функции pinMode() :

pinMode(13, OUTPUT); // 13 вывод будет выходом

pinMode(12, INPUT); // 12 - входом

В программе можно создавать собственные константы:

#define LEFT 0x95

#define MESS\_LEFT "поворот влево"

**Переменные**

Переменные — это способ именовать и хранить числовые значения для последую­щего использования программой. Переменные — это значения, которые могут по­следовательно меняться, в отличие от констант, чье значение никогда не меняется. Переменные нужно декларировать (объявлять). Следующий код объявляет пере­менную inputVariable, а затем присваивает ей значение, полученное от второго анало­гового порта:

int inputVariable=0;

inputVariable=analogRead(2);

Переменные могут быть названы любыми именами, которые не являются ключе­выми словами языка программирования Arduino.

**Объявление переменных**

Все переменные должны быть задекларированы до того, как они могут использо­ваться. Объявление переменной означает определение типа ее значения: int, long, float и т. д., задание уникального имени переменной, и дополнительно ей можно присвоить начальное значение. Все это следует делать только один раз в програм­ме, но значение может меняться в любое время при использовании арифметических или других разных операций.

Следующий пример показывает, что объявленная переменная inputVariable имеет тип int, и ее начальное значение равно нулю. Это называется простым присваива­нием.

int inputVariable = 0;

Переменная может быть объявлена в разных местах программы, и то, где это сде­лано, определяет, какие части программы могут использовать переенную.

**Границы переменных**

Переменные могут быть объявлены в начале программы перед void setup(), ло­кально внутри функций и иногда в блоке выражений, таком как цикл for. То, где объявлена переменная, определяет ее границы (область видимости), т. е. возмож­ность некоторых частей программы ее использовать.

**Глобальные переменные** таковы, что их могут видеть и использовать любые функ­ции и выражения программы. Такие переменные декларируются в начале програм­мы перед функцией setup().

**Локальные переменные** определяются внутри функций или таких частей, как цикл for. Они видимы и могут использоваться только внутри функции, в которой объяв­лены. Таким образом, могут существовать несколько переменных с одинаковыми именами в разных частях одной программы, которые содержат разные значения. Уверенность, что только одна функция имеет доступ к ее переменной, упрощает программу и уменьшает потенциальную опасность возникновения ошибок.

# 26. Цифровой ввод/вывод. Функция pinMode. Функция digitalWrite(), Void setup(), Void loop(), Функция digitalRead()

1. **Цифровые ввод/вывод:**

* пины с 0-13
* логический уровень единицы 5V, 0-0V
* максимальный ток вывода 40мА
* к контактам подключены подтягивающие резисторы, которые по умолчанию выключены, но могут быть включены программно.
* на плате выведены 14 цифровых пинов, любой из которых может работать, как на вывод информации, так и на ввод. Для этого в ходе программы применяются специальные функции: (рассмотрим ниже)

**Цифровой ввод/вывод**

Рассмотрим функции цифрового ввода/вывода:

- pinMode();

- digitalWrite();

- digitalRead().

**Функция pinMode**

Устанавливает режим работы заданного входа/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 значений: input или output устанавливает на вход или выход соответственно.

Пример:

int ledPin = 13; // Светодиод, подключенный к входу/выходу 13

void setup()

{

pinMode(ledPin,OUTPUT); // устанавливает режим работы - выход     }

**Функция digitalWrite()**

Подает high или low значение на цифровой вход/выход (pin).

Если вход/выход (pin) был установлен в режим выход (output) функцией pinMode(), то для значения high напряжение на соответствующем входе/выходе (pin) будет 5 В (3,3 В для плат 3,3 В) и 0 В (земля) для low.

Если вход/выход (pin) был установлен в режим вход (input), то функция digitalwrite со значением high будет активировать внутренний нагрузочный рези­стор 20 K. Подача low в свою очередь отключает этот резистор. Нагрузочного рези­стора достаточно, чтобы светодиод, подключенный к входу, светил тускло. Если вдруг светодиод работает, но очень тускло, возможно необходимо установить режим выход (output) функцией pinMode() .

Синтаксис:

digitalWrite(pin, value);

Параметры:

 pin — номер входа/выхода (pin);

 value — значение high или low.

Пример:

int ledPin = 13;// Светодиод, подключенный к входу/выходу 13

**5) Функция Void 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.

**6) Функция Void Loop()**

После вызова функции setup(), которая инициализирует и устанавливает первоначальные значения, функция loop() делает точь-в-точь то, что означает её название, и крутится в цикле, позволяя вашей программе совершать вычисления и реагировать на них. Используйте её для активного управления платой Arduino.

**Функция digitalRead()**

Функция считывает значение с заданного входа: high или low.

Синтаксис:

digitalRead(pin);

Параметр: pin — номер входа/выхода (pin), который вы хотите считать.

Пример:

int ledPin = 13; // Светодиод, подключенный к входу/выходу 13

int inPin = 7; // кнопка на входе 7

int val = 0; // переменная для хранения значения

void setup()

{

pinMode(ledPin, OUTPUT); // устанавливает режим работы - выход для 13

pinMode(inPin, INPUT); // устанавливает режим работы - вход для 7 }

void loop()

{

val = digitalRead(inPin); // считываем значение с входа digitalWrite(ledPin, val); // устанавливаем значение на светодиоде // равным значению входа кнопки

}

Если вход не подключен, то digitalRead может возвращать значения HIGH или LOW случайным образом. Аналоговые входы (analog pins) могут быть использованы как цифровые входы/выходы (digital pins). Обращение к ним идет по номерам от 14 (для аналогового входа 0) до 19 (для аналогового входа 5).

# 27. Аналоговый ввод/вывод. Функция analogRead(). Функция analogReference(), Функция analogWrite()

1. **Аналоговый ввод/вывод**

Входы: A0, A1, A2, A3, A4, A5 (6) каждый из которых может представить аналоговое напряжение в виде 10-битного числа.

**Аналоговый ввод/вывод**

Рассмотрим функции аналогового ввода/вывода:

- analogRead() ;

- analogReference() ;

- analogWrite() .

**Функция analogRead()**

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 напряжения и шаг может быть изменен функцией analogReference (). Считывание значения с аналого­вого входа занимает примерно 100 микросекунд (0,0001 сек), т. е. максимальная частота считывания приблизительно 10 000 раз в секунду.

Синтаксис:

analogRead(pin);

Параметр: 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).

Если аналоговый вход не подключен, то значения, возвращаемые функцией analogRead (), могут принимать случайные значения.

Пример:

int analogPin = 3; // номер порта, к которому подключен потенциометр

int val = 0; // переменная для хранения считываемого значения

void setup()

{

Serial.begin(9600); // установка связи по serial

}

void loop()

{

val = analogRead(analogPin); // считываем значение Serial.println(val); // выводим полученное значение

}

**Функция analogReference()**

Функция определяет опорное напряжение, относительно которого происходят ана­логовые измерения. Функция analogRead() возвращает значение с разрешением 8 битов (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);

Параметр: type — определяет используемое опорное напряжение (default, internal или external).

Внешнее напряжение рекомендуется подключать к выводу AREF через резистор 5 кОм.

Рекомендуемой настройкой для вывода AREF является external. При этом про­исходит отключение обоих внутренних источников, и внешнее напряжение будет являться опорным для АЦП.

**Функция analogWrite()**

Выдает аналоговую величину (ШИМ-волну) на порт входа/выхода. Функция может быть полезна для управления яркостью подключенного светодиода или скоростью вращения 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 входа/выхода). Частота ШИМ-сигнала приблизительно 490 Гц.

На большинстве плат Arduino (на базе микроконтроллера ATmega168 или ATmega328) ШИМ поддерживают порты 3, 5, 6, 9, 10 и 11, на плате Arduino Mega — порты с 2 по 13. На более ранних версиях плат Arduino analogWrite() ра­ботал только на портах 9, 10 и 11.

Для вызова analogWrite () нет необходимости устанавливать тип входа/выхода функцией pinMode(). Функция analogWrite() никак не связана с аналоговыми вхо­дами и с функцией analogRead ().

Синтаксис:

analogWrite(pin, value);

Параметры:

 pin — порт входа/выхода, на который подается ШИМ-сигнал;

 value — период рабочего цикла: значение между 0 (полностью выключено) и 255 (сигнал подан постоянно).

Период ШИМ-сигнала на портах входа/выхода 5 и 6 будет несколько длиннее. Это связано с тем, что таймер для данных выходов также задействован функциями millis() и delay(). Данный эффект более заметен при установке коротких периодов ШИМ-сигнала (0-10).

Пример задания яркости светодиода пропорционально значению, снимаемому с потенциометра:

int ledPin = 9; // Светодиод подключен к выходу 9

int analogPin = 3; // потенциометр подключен к выходу 3 int val = 0; // переменная для хранения значения

void setup()

{

pinMode(ledPin, OUTPUT); // установка порта на выход }

void loop()

{

val = analogRead(analogPin); // считываем значение с порта,

// подключенного к потенциометру analogWrite(ledPin, val / 4); // analogRead возвращает значения от 0 // до 1023, analogWrite должно быть // в диапазоне от 0 до 255

}

# 28. Дополнительные фунции ввода/вывода. Функция tone(). Функция noTone(). Функция shiftOut(). Функция pulseIn()

**Функция tone()**

Генерирует на порту входа/выхода сигнал — прямоугольную "волну" заданной частоты и с 50 % рабочим циклом. Длительность может быть задана параметром, в противном случае сигнал генерируется до тех пор, пока не будет вызвана функ­ция noTone(). К порту входа/выхода может быть подключен пьезо- или иной дина­мик для воспроизведения сигнала.

Воспроизводиться одновременно может только один сигнал. Если сигнал уже вос­производится на одном порту, то вызов tone() с номером другого порта в качестве параметра ни к чему не приведет, если же tone () будет вызвана с тем же номером порта, то будет установлена новая частота сигнала.

Использование функции tone() помешает использовать ШИМ на портах вхо­да/выхода 3 и 11 (кроме платы Arduino Mega).

Синтаксис:

tone(pin, frequency);

tone(pin, frequency, duration);

Параметры:

- pin — номер порта входа/выхода, на котором будет генерироваться сигнал;

- frequency — частота сигнала в герцах;

- duration — длительность сигнала в миллисекундах.

**Функция noTone()**

Останавливает сигнал, генерируемый на порту входа/выхода, вызовом функции tone () . Если сигнал не генерировался, то вызов noTone() ни к чему не приводит.

Если необходимы сигналы на разных портах, то следует сначала остановить один сигнал функцией noTone(), а лишь затем создавать новый сигнал на другом порту функцией Tone().

Синтаксис:

noTone(pin);

Параметр: pin — номер порта входа/выхода, на котором прекращается сигнал.

**Функция shiftOut()**

Выводит байт информации на порт входа/выхода последовательно (побитно). Вы­вод может осуществляться как с первого (левого), так и с последнего (правого) бита. Каждый бит последовательно подается на заданный порт, после чего подается сигнал на синхронизирующий порт входа/выхода, информируя о доступности к считыванию бита.

Такой способ передачи данных называется последовательным протоколом с син­хронизацией. Он часто используется для взаимодействия микроконтроллеров с дат­чиками и сенсорами, а также другими микроконтроллерами. Последовательная передача с синхронизацией позволяет устройствам связываться на максимальной скорости. Смотрите также документацию (на англ. языке) по протоколу последова­тельного периферийного интерфейса (SPI, Serial Peripheral Interface Protocol).

Синтаксис:

shiftOut(dataPin, clockPin, bitOrder, value);

Параметры:

- dataPin — номер порта входа/выхода, на который выводятся биты (int);

- clockPin — номер порта, по которому производится синхронизация (int);

- bitOrder — используемая последовательность вывода битов. msbfirst (Most Significant Bit First) — слева или lsbfirst (Least Significant Bit First) — справа;

-  value — значение (байт) для вывода (byte).

Порт вывода (dataPin) и синхронизирующий порт (clockPin) должны быть предвари­тельно сконфигурированы как порты вывода с помощью функции pinMode ().

Текущая реализация функции shiftOut() может выводить только один байт (8 би­тов) информации, поэтому необходимо произвести несколько действий, чтобы вывести значения больше 255.

Пример вывода:

// Вывод будет MSBFIRST с первого (левого) бита int data = 500;

// выводим старший байт

shiftOut(dataPin, clock, MSBFIRST, (data >> 8));

// выводим младший бит

shiftOut(dataPin, clock, LSBFIRST, data);

// выводим старший бит

shiftOut(dataPin, clock, LSBFIRST, (data >> 8));

Пример вывода счетчика от 0 до 255 на сдвиговый регистр с последовательным вводом 74HC595:

// Порт, подключенный к ST\_CP 74HC595 int latchPin = 8;

// Порт, подключенный к SH\_CP 74HC595 int clockPin = 12;

// Порт, подключенный к DS 74HC595 int dataPin = 11; void setup()

{

// устанавливаем режим порта выхода pinMode(latchPin, OUTPUT); pinMode(clockPin, OUTPUT); pinMode(dataPin, OUTPUT);

}

void loop()

{

for (int j = 0; j < 256; j++)

{

// устанавливаем LOW на latchPin, пока не окончена передача байта digitalWrite(latchPin, LOW); shiftOut(dataPin, clockPin, LSBFIRST, j);

// устанавливаем HIGH на latchPin, чтобы проинформировать регистр, что // передача окончена. digitalWrite(latchPin, HIGH); delay(1000);

}

}

**Функция pulseIn()**

Считывает длину сигнала на заданном порту (high или low). Например, если задано считывание high функцией pulsein(), функция ожидает, пока на заданном порту не появится high. Когда high получен, включается таймер, который будет остановлен, когда на порту входа/выхода будет low. Функция pulsein() возвращает длину сиг­нала в микросекундах. Функция возвращает 0, если в течение заданного времени (тайм-аута) не был зафиксирован сигнал на порту.

Возможны некоторые погрешности в измерении 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 входа/выхода, на котором будет ожидаться сигнал;

- value — тип ожидаемого сигнала: high или low;

- timeout время ожидания сигнала (тайм-аут) в секундах (unsigned long).

Возвращаемые значения: длина сигнала в микросекундах или 0, если сигнал не по­лучен до истечения тайм-аута (тип unsigned long).

Пример использования функции:

int pin = 7; unsigned long duration; void setup()

{

pinMode(pin, INPUT);

}

void loop()

{

duration = pulseIn(pin, HIGH);

}

# 29. Работа со временем. Функция millis(). Функция micros(). Функция delay(). Функция delayMicroseconds()

**Работа со временем**

**Функция millis()**

Возвращает количество миллисекунд с момента начала выполнения текущей про­граммы на плате Arduino. Это количество сбрасывается на ноль вследствие пере­полнения значения приблизительно через 50 дней.

Параметров нет.

Возвращаемое значение — количество миллисекунд с момента начала выполнения программы (тип unsigned long).

Пример использования функции:

unsigned long time; void setup()

{

Serial.begin(9600);

}

void loop()

{

Serial.print("Time: "); time = millis();

// выводит количество миллисекунд с момента начала выполнения программы Serial.println(time);

// ждет секунду перед следующей итерацией цикла. delay(1000);

}

**Функция micros()**

Возвращает количество микросекунд с момента начала выполнения текущей про­граммы на плате Arduino. Значение переполняется и сбрасывается на ноль прибли­зительно через 70 минут. На платах Arduino с 16 МГц (Duemilanove и Nano) функ­ция micros() имеет разрешение 4 секунды (возвращаемое значение всегда крат­но 4). На платах с 8 МГц (Arduino Lilypad) — разрешение функции 8 секунд.

Параметров нет.

Возвращаемое значение — количество микросекунд с момента начала выполнения программы (unsigned long).

Пример использования функции:

unsigned long time; void setup()

{

Serial.begin(9600);

}

void loop()

{

Serial.print(”Time: ”); time = micros();

// выводит количество микросекунд с момента начала выполнения

// программы

Serial.println(time);

// ждет секунду перед следующей итерацией цикла. delay(1000);

}

**Функция delay()**

Останавливает выполнение программы на заданное в параметре количество милли­секунд (1000 миллисекунд в 1 секунде).

Синтаксис:

delay(ms);

Параметр: ms — количество миллисекунд, на которое приостанавливается выпол­нение программы ( тип unsigned long).

Пример использования функции:

int ledPin = 13; // светодиод подключен на порт 13

void setup()

{

pinMode(ledPin, OUTPUT); // устанавливается режим порта - выход }

void loop()

{

digitalWrite(ledPin, HIGH); // включаем светодиод

delay(1000); // ожидаем секунду

digitalWrite(ledPin, LOW); // выключаем светодиод delay(1000); // ожидаем секунду

}

Не рекомендуется использовать эту функцию для событий длиннее 10 миллисе­кунд, т. к. во время останова не могут быть произведены манипуляции с портами, не могут быть считаны сенсоры или произведены математические операции. В ка­честве альтернативного подхода возможно контролирование времени выполнения тех или иных функций с помощью millis(). При использовании функции delay() работа прерываний не останавливается, продолжается запись последовательно (serial) передаваемых данных на RX‑порту, ШИМ-сигнал (analogWrite) продолжает генерироваться на портах.

**Функция delayMicroseconds()**

Останавливает выполнение программы на заданное в параметре количество микро­секунд (1 000 000 микросекунд в 1 секунде).

В данной версии Arduino максимальная пауза, воспроизводимая корректно, — 16 383. Возможно, это будет изменено в следующих версиях Arduino. Для останов­ки выполнения программы, более чем на несколько тысяч микросекунд, рекомен­дуется использовать функцию delay().

Синтаксис:

delayMicroseconds(us);

Параметр: us — количество микросекунд, на которое приостанавливается выпол­нение программы (unsigned int).

Пример использования функции:

int outPin = 8; // цифровой порт входа/выхода 8

void setup()

{

pinMode(outPin, OUTPUT); // устанавливается режим порта - выход }

void loop()

{

digitalWrite(outPin, HIGH); // подаем HIGH на выход delay(50); // ожидаем 50 микросекунд

digitalWrite(outPin, LOW); // устанавливаем LOW на выходе delay(50); // ожидаем 50 микросекунд

}

# 30. Математические функции: map() ; pow() ; sq(); sqrt (); min(); max(); abs(); constrain();

**Математические функции**

В языке представлены следующие математические функции:

- map() ;

- pow() ;

- sq();

- sqrt () .

- min() ;

- max() ;

- abs() ;

- constrain();

**Функция min(x,yx)**

Возвращает наименьшее из двух значений.

Параметры:

-  x — первое число, любой тип;

-  y — второе число, любой тип.

Возвращаемое значение — возвращает меньшее из двух сравниваемых значений.

Пример использования функции:

sensVal = min(sensVal, 100);

// проверяем, если sensVal больше 100, то senseVal будет присвоено 100

**Функция max(x, y)**

Возвращает большее из двух значений.

Параметры:

- x — первое число, любой тип;

- y — второе число, любой тип.

Возвращаемое значение — возвращает большее из двух сравниваемых значений.

Пример использования функции:

sensVal = max(sensVal, 20);

// проверяем, если sensVal меньше 20, то senseVal будет присвоено 20

Функция max() зачастую используется для ограничения нижней границы значений переменной. Функцией min() ограничивают верхнюю границу переменной. В силу специфики реализации функции max() следует избегать использования других функций в качестве параметров.

Например:

max(a—, 0); // может привести к некорректным результатам

a-- ;

max(a, 0); // так корректно

**Функция abs()**

Возвращает модуль числа.

Параметр: x — число.

Возвращаемые значения:

-  x — если x больше или равен 0;

-  -x — если x меньше 0.

В силу специфики реализации функции abs() следует избегать использования дру­гих функций в качестве параметров:

- abs (a++); // может привести к некорректным результатам a++;

- abs(a, 0); //  так корректно

**Функция constrain(x, a, b)**

Функция проверяет и, если надо, задает новое значение так, чтобы оно было в об­ласти допустимых значений, заданной параметрами.

Параметры:

-  x — проверяемое значение, любой тип;

-  a — нижняя граница области допустимых значений, любой тип;

-  b — верхняя граница области допустимых значений, любой тип.

Возвращаемое значение:

-  x — если x входит в область допустимых значений [a..b];

-  a — если x меньше a;

-  b — если x больше b.

Пример:

sensVal = constrain(sensVal, 10, 150);

// ограничиваем значения sensVal диапазоном от 10 до 150

Функция пропорционально переносит значение (value) из текущего диапазона зна­чений (fromLow . . fromHigh) в новый диапазон (toLow . . toHigh), заданный пара­метрами.

Функция map() не ограничивает значение рамками диапазона, как это делает функ­ция constrain(). Функция contsrain () может быть использована до или после вы­зова map (), если необходимо ограничить допустимые значения заданным диапазо­ном.

Обратить внимание, что "нижняя граница" может быть как меньше, так и больше "верхней границы". Это может быть использовано, чтобы "перевернуть" диапазон:

y = map(x, 1, 50, 50, 1);

Возможно использование отрицательных значений:

y = map(x, 1, 50, 50, -100);

Функция map() оперирует целыми числами. При пропорциональном переносе дробная часть не округляется по правилам, а просто отбрасывается.

Параметры:

- value — значение для переноса;

- fromLow — нижняя граница текущего диапазона;

- fromHigh — верхняя граница текущего диапазона;

- toLow — нижняя граница нового диапазона, в который переносится значение;

- toHigh — верхняя граница нового диапазона.

Возвращаемое значение — значение в новом диапазоне.

Пример использования функции:

// Переносим значение с аналогового входа // (возможные значения от 0 до 1023) в 8 бит (0..255) void setup()

{;}

void loop()

{

int val = analogRead(0);

val = map(val, 0, 1023, 0, 255);

analogWrite(9, val);

}

**Функция pow(base, exponent)**

Вычисляет значение, возведенное в заданную степень. Функция pow() может воз­водить и в дробную степень.

Параметры:

-  base — число (тип float);

-  exponent — степень, в которую будет возводиться число (тип float). Возвращаемое значение — результат возведения в степень, число (тип double).

**Функция sq(x)**

Функция возвращает квадрат числа, заданного параметром.

Параметр: x — число, любой тип.

Возвращаемое значение — квадрат числа.

**Функция sqrt(x)**

Функция вычисляет квадратный корень числа, заданного параметром.

Параметры: x — число, любой тип.

Возвращаемое значение — квадратный корень числа (тип double).

 Тригонометрические функции

В языке представлены следующие тригонометрические функции: sin(); cos(); tan().

**Функция sin( rad)**

Возвращает синус угла, заданного в радианах в передаваемом параметре. Результат функции всегда в диапазоне -1 .. 1.

Параметр: rad — угол в радианах (float).

Возвращаемое значение: синус угла (тип double).

**Функция cos(rad)**

Возвращает косинус угла, заданного в радианах в передаваемом параметре. Резуль­тат функции всегда находится в диапазоне -1 .. 1.

Параметр: rad — угол в радианах (тип float).

Возвращаемое значение: косинус угла (тип double).

**Функция tan(rad)**

Возвращает тангенс угла, заданного в радианах в передаваемом параметре. Резуль­тат функции в диапазоне от минус бесконечности до плюс бесконечности.

Параметр: rad — угол в радианах (тип float).

Возвращаемое значение: тангенс угла (тип double).

# 31. Генераторы случайных значений: randomSeed(); random() .

**Генераторы случайных значений**

Функции формирования случайных чисел:

- randomSeed() ;

- random() .

**Функция randomSeed(seed)**

Функция randomSeed() 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.

Параметр: seed — параметр, задающий начало выдачи псевдослучайных значений на последовательности (тип int, long).

**Функция random()**

Функция random() возвращает псевдослучайное число.

Синтаксис:

random(max); random(min, max);

Параметры:

-  min — нижняя граница случайных значений, включительно (опционально);

-  max — верхняя граница случайных значений, включительно.

Возвращаемое значение: случайное число между min и max - 1 (тип long).

Если при каждом запуске программы необходимо получать разные последователь­ности значений, генерируемых функцией random (), то необходимо инициализиро­вать генератор псевдослучайных чисел со случайным параметром. Например, мож­но использовать значение, отдаваемое функцией analogRead() c неподключенного порта входа/выхода. В некоторых случаях необходимо получать одинаковую по­следовательность при каждом запуске программы на Arduino. Тогда инициализиро­вать генератор псевдослучайных чисел следует вызовом функции randomSeed() с фиксированным параметром.

Пример использования функции:

long randNumber; void setup()

{

Serial.begin(9600);

}

void loop()

{

// выводим случайное число из диапазона 0..299 randNumber = random(300);

Serial.println(randNumber);

// выводим случайное число из диапазона 0..19 randNumber = random(10, 20);

Serial.println(randNumber); delay(50);

}