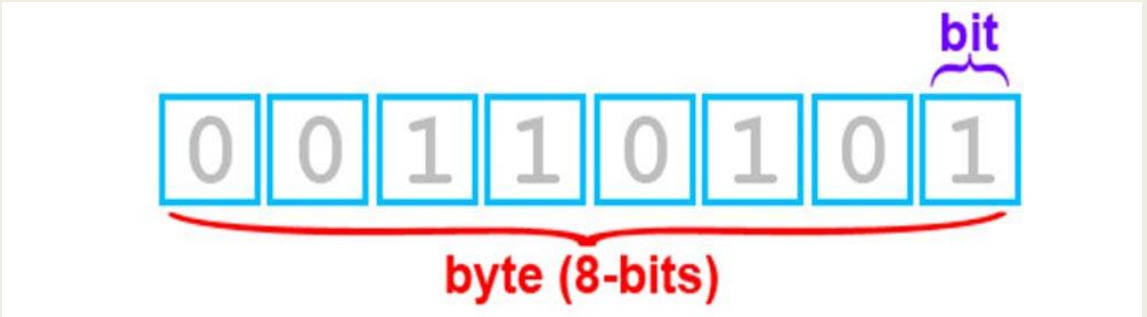
Основы кибернетики и робототехники
Лекция 8

Битовые операции



Данный урок посвящён битовым операциям (операциям с битами, битовой математике, bitmath). Из него вы узнаете, как оперировать с битами.

Бит – элементарная ячейка памяти микроконтроллера.

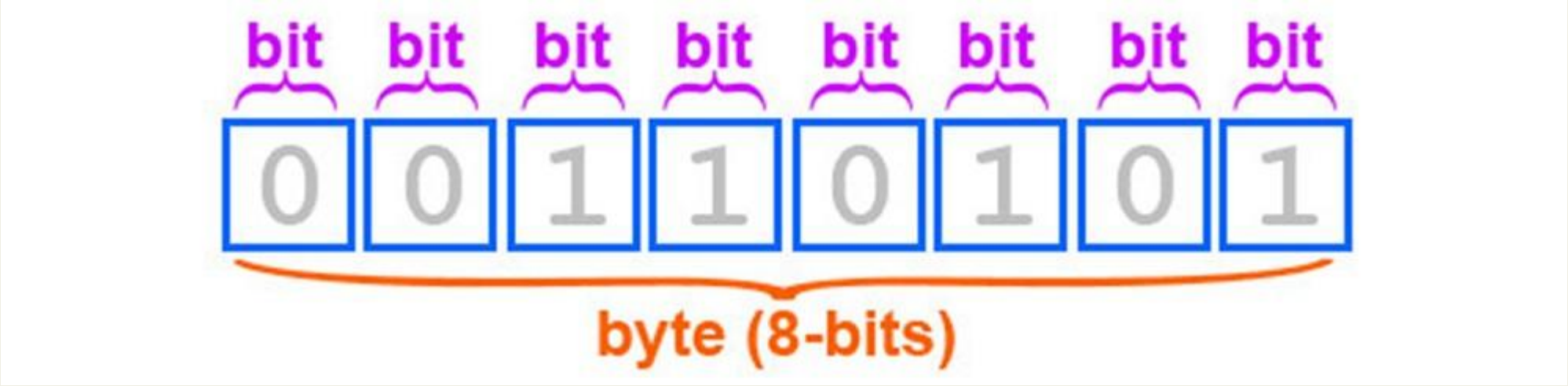
Зачем вообще нужно уметь работать с битами:

- **Гибкая и быстрая работа напрямую с регистрами микроконтроллера.**
- **Работа напрямую с внешними микросхемами (датчики и прочее), управление которыми состоит из записи и чтения регистров, данные в которых могут быть запакованы в байты самым причудливым образом.**
- **Более эффективное хранение данных: упаковка нескольких значений в одну переменную и распаковка обратно.**
- **Создание символов и другой информации для матричных дисплеев.**
- **Максимально быстрые вычисления.**
- **Разбор чужого кода.**

Двоичная система

В цифровом мире, к которому относится также микроконтроллер, информация хранится, преобразуется и передается в цифровом виде, то есть в виде нулей и единиц. Соответственно элементарная ячейка памяти, которая может запомнить **0 или 1**, называется **бит (bit).**

Минимальная ячейка памяти, которую мы можем изменить – 1 бит, а ячейка памяти, которая имеет адрес в памяти и мы можем к ней обратиться – байт, который состоит из 8-ми бит, каждый занимает своё место (примечание: в других архитектурах в байте может быть больше или меньше бит, в данном уроке речь идёт об AVR(семейство восьмибитных микроконтроллеров) и 8-ми битном байте).



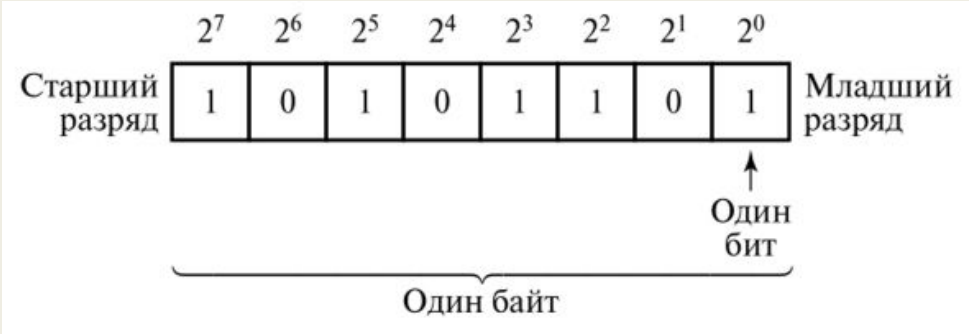
Вспомним двоичную систему счисления из школьного курса информатики:

Двоичная система	Десятичная система
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10

Здесь также нужно увидеть важность степени двойки – на ней в битовых операциях завязано абсолютно всё. Давайте посмотрим на первые 8 степеней двойки в разных системах счисления:

2 в степени	DEC	BIN
0	1	0b00000001
1	2	0b00000010
2	4	0b00000100
3	8	0b00001000
4	16	0b00010000
5	32	0b00100000
6	64	0b01000000
7	128	0b10000000

Таким образом, степень двойки явно “указывает” на номер бита в байте, считая справа налево (примечание: в других архитектурах может быть иначе). Напомню, что абсолютно неважно, в какой системе исчисления вы работаете – микроконтроллеру всё равно и он во всём видит единицы и нули. Если мы “включим” все биты в байте, то получится число 0b11111111 в двоичной системе или 255 в десятичной.



Если сложить полный байт в десятичном представлении каждого бита: 128+64+32+16+8+4+2+1 – получится 255. Нетрудно догадаться, что число 0b11000000 равно 128+64, то есть 192. Именно таким образом и получается весь диапазон от 0 до 255, который умещается в один байт. Если взять два байта – будет всё то же самое, просто ячеек будет 16, то же самое для 4 байт – 32 ячейки с единицами и нулями, каждая имеет свой номер согласно степени двойки.

Битовые операции

- Битовое И
- Битовое ИЛИ
- Битовое НЕ
- Битовое исключающее ИЛИ
- Битовый сдвиг

Битовое И

И (AND), оно же “логическое умножение”, выполняется оператором & или AND. Побитовое И работает с каждой битовой позицией окружающих выражений независимо в соответствии с этим правилом: если оба входных бита равны 1, результирующий выход равен 1, иначе выход равен 0. Другой способ выразить это так:

```
0 & 0 == 0
0 & 1 == 0
1 & 0 == 0
1 & 1 == 1
```

Основное применение операции И – битовая маска. Позволяет “взять” из байта только указанные биты:

```
myByte = 0b11001100;
myBits = myByte & 0b10000111;
// myBits теперь равен 0b10000100
```

То есть при помощи & мы взяли из байта 0b11001100 только биты 10000111, а именно – 0b11001100, и получили 0b10000100.

Также можно использовать составной оператор &=

```
myByte = 0b11001100;
myByte &= 0b10000000; // берём старший бит
// myByte теперь 10000000
```

&= – оператор присвоения битового "и«

a &= b –> a = a & b

Битовое ИЛИ

ИЛИ (OR), оно же “логическое сложение”, выполняется оператором | или **or**.
Подобно оператору **&**, |работает независимо с каждым битом в двух окружающих его целочисленных выражениях, но то, что он делает, отличается (конечно). Побитовое ИЛИ двух битов равно 1, если один или оба входных бита равны 1, в противном случае оно равно 0. Другими словами:

```
0 | 0 == 0
0 | 1 == 1
1 | 0 == 1
1 | 1 == 1
```

Основное применение операции ИЛИ – установка бита в байте:

```
myByte = 0b11001100;
myBits = myByte | 0b00000001; // ставим бит №0
// myBits теперь равен 0b11001101

myBits = myBits | bit(1); // ставим бит №1
// myBits теперь равен 0b11001111
```

Также можно использовать составной оператор |=

```
myByte = 0b11001100;
myByte |= 16; // 16 - 2 в 4, включаем бит №4
// myByte теперь 0b11011100
```

|= это оператор присвоения битового "или", аналогичен

a |= b –> a = a | b

Битовое исключающее ИЛИ

Битовая операция исключающее ИЛИ (XOR) выполняется оператором ^ или xor и делает следующее:

```
0 ^ 0 == 0
0 ^ 1 == 1
1 ^ 0 == 1
1 ^ 1 == 0
```

Данная операция обычно используется для инвертирования состояния отдельного бита:

```
myByte = 0b11001100;
myByte ^= 0b10000000; // инвертируем 7-ой бит
// myByte теперь 01001100
```

То есть мы взяли бит №7 в байте 0b11001100 и перевернули его в 0, получилось 0b01001100, остальные биты не трогали.

Битовое НЕ

Битовая операция НЕ (NOT) выполняется оператором ~ и просто инвертирует бит.
В отличие от &и |, побитовый оператор НЕ применяется к одному операнду справа от него. Побитовое НЕ изменяет каждый бит на его противоположность: 0 становится 1, а 1 становится 0. Например:

```
~0 == 1
~1 == 0
```

Также она может инвертировать байт:

```
myByte = 0b11001100;
myByte = ~myByte; // инвертируем
// myByte теперь 00110011
```

Битовый сдвиг

Битовый сдвиг – очень мощный оператор, позволяет буквально “двигать” биты в байте вправо и влево при помощи операторов >> и <<, и соответственно составных >>= и <<=. Если биты выходят за границы блока (8 бит, 16 бит или 32 бита) – они теряются.

```
myByte = 0b00011100;
myByte = myByte << 3; // двигаем на 3 влево
// myByte теперь 0b11100000

myByte >>= 5;
// myByte теперь 0b00000111

myByte >>= 2;
// myByte теперь 0b00000001
// остальные биты потеряны!
```

Битовые операции – самые быстрые, например, если вам требуется максимальная скорость вычислений. Так же при помощи битовых операций можно экономить немного памяти. Битовые операции часто используются во время программирования для установки или считывания состояния регистров или выводов на основе сохраненных или переданных данных.

Функции для работы с битами

bitRead() - функция позволяет прочитать значение определенного бита в указанной переменной.
Мы вызываем ее с двумя параметрами. Первый из них — это имя переменной, значение бита которой мы хотим получить, второй — номер бита для чтения.
Отсчет всегда от нуля, то есть у нас есть бит нулевой, первый, второй и т. д. Функция возвращает значение LOW или HIGH.

bitWrite() - функция позволяет записать определенный бит. Вызывается с тремя параметрами, первый из них — это переменная, второй — это номер бита, а третий — это информация о том, что бит необходимо установить (HIGH) или сбросить (LOW).

```
byte x=0xF0; // в двоичном 11110000

bitRead(x,3); // функция возвращает LOW

bitRead(x,7); // функция возвращает HIGH

bitWrite(x,1,HIGH); // x = 11110010

bitWrite(x,7,LOW); // x = 01110010
```

Альтернативным способом изменения состояния (значения) бита для функции bitWrite() — функции bitSet() и bitClear().

В функции bitWrite() в качестве параметра мы задаем установить или сбросить бит. Используя одну из функций bitSet () или bitClear (), мы явно определяем, хотим ли мы, чтобы бит был установлен или сброшен.

```
byte a=0xAA; // в двоичном виде 10101010

bitSet(a,6); // a = 11101010

bitClear(a,7); // a = 01101010

bitClear(a,7); // a = 01101010

bitClear(a,6); // a = 00101010

bitSet(a,7); // a = 10101010
```

Функция бит () вычисляет значение указанного бита. Ниже примеры:

```
byte x;
x = bit (0); // x = 1
x = bit (2); // x = 4
x = bit (4); // x = 16
x = bit (7); // x = 128
```

Arduino Uno

Arduino Uno является стандартной платой Arduino и возможно наиболее распространенной.

Она основана на чипе ATmega328, имеющем на борту 32 КБ флэш-памяти, 2 Кб SRAM и 1 Кбайт EEPROM памяти.

На периферие имеет 14 дискретных (цифровых) каналов ввода / вывода и 6 аналоговых каналов ввода / вывода.

Arduino Leonardo

Большее количество аналоговых входов (12 против 6) для сенсоров, больше каналов ШИМ (7 против 6), больше пинов с аппаратным прерыванием (5 против 2), раздельные независимые serial-интерфейсы для USB и UART.

Arduino Leonardo может притворяться клавиатурой или мышью (HID-устройством) для компьютера. Это позволяет легко сделать своё собственное устройство ввода.

Arduino Mega

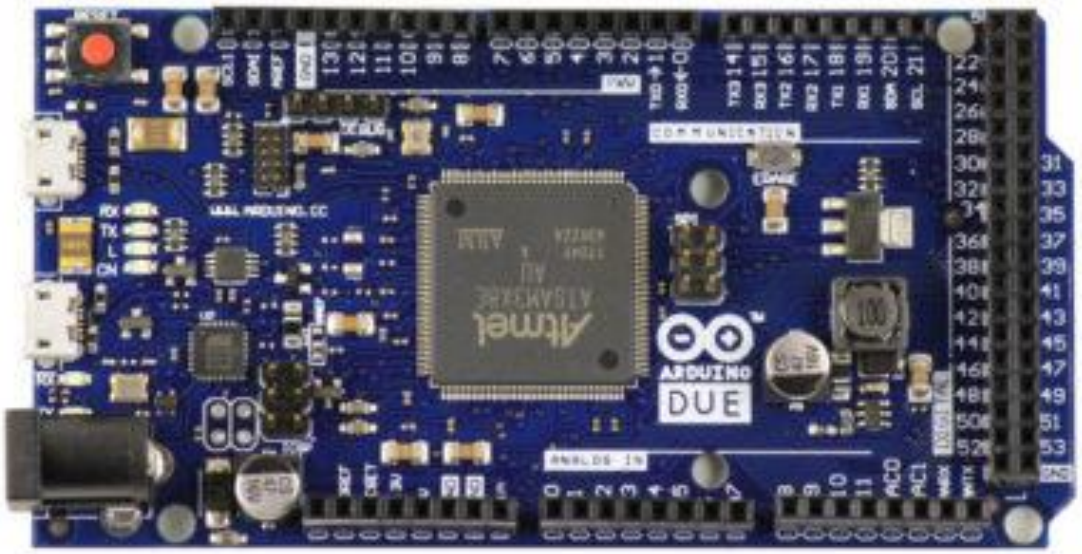
Как Arduino Uno, но на базе более мощного микроконтроллера той же архитектуры. Отличный выбор «на вырост» или если Arduino Uno перестала справляться.

В разы больше памяти: 256 КБ против 32 КБ постоянной и 8 КБ против 2 КБ оперативной. В разы больше портов: 60 из них 16 аналоговых и 15 с ШИМ. Немного длиннее базовой Arduino Uno: 101×53 мм против 69×53 мм.

Arduino Due

Процессор на 84 МГц и 512 КБ памяти. 66 пинов ввода-вывода, из которых 12 могут быть аналоговыми входами, 12 поддерживают ШИМ и все 66 могут быть настроены, как аппаратные прерывания.

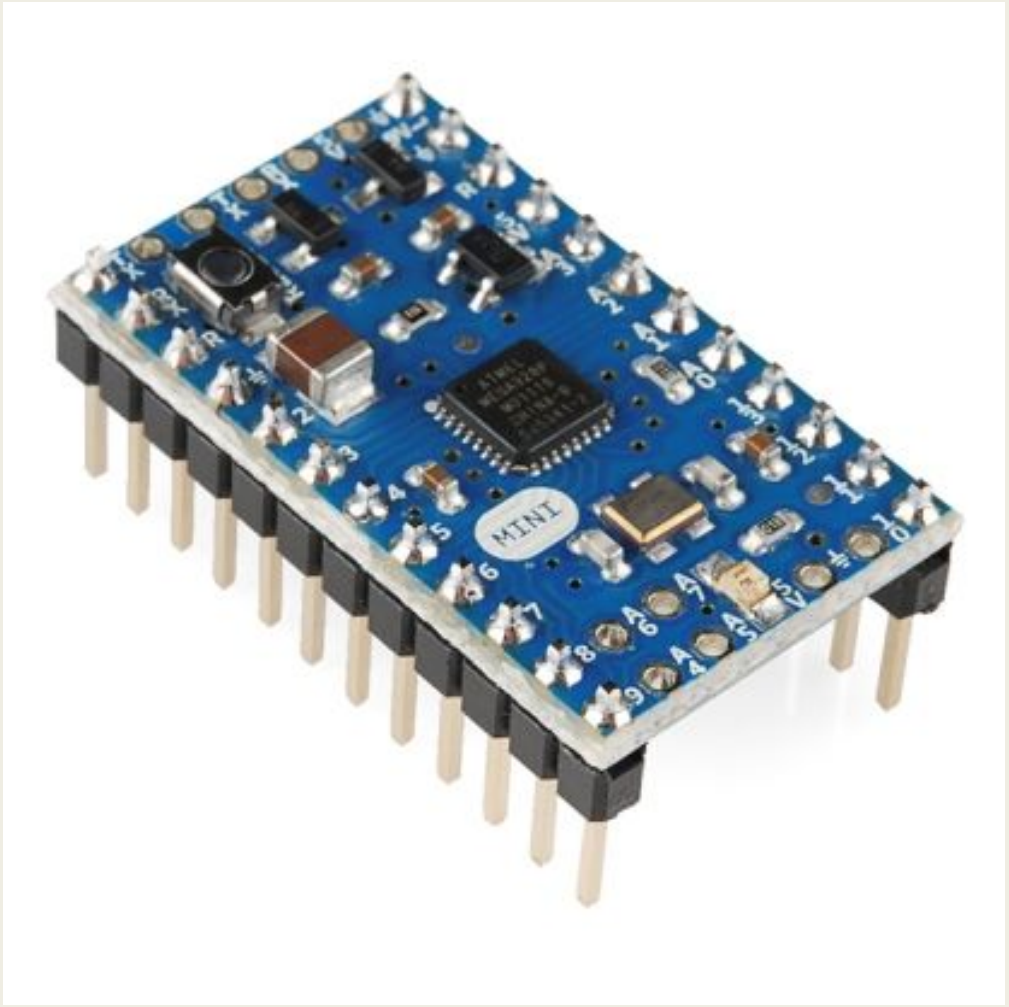
Встроенный контроллер шины CAN позволяет создавать сеть из Due или взаимодействовать с автомобильной электроникой. Два канала ЦАП позволяют синтезировать стереозвук с разрешением в 4,88 Гц. Родным напряжением для платы является 3.3 В, а не традиционные 5 В.



Arduino Mini

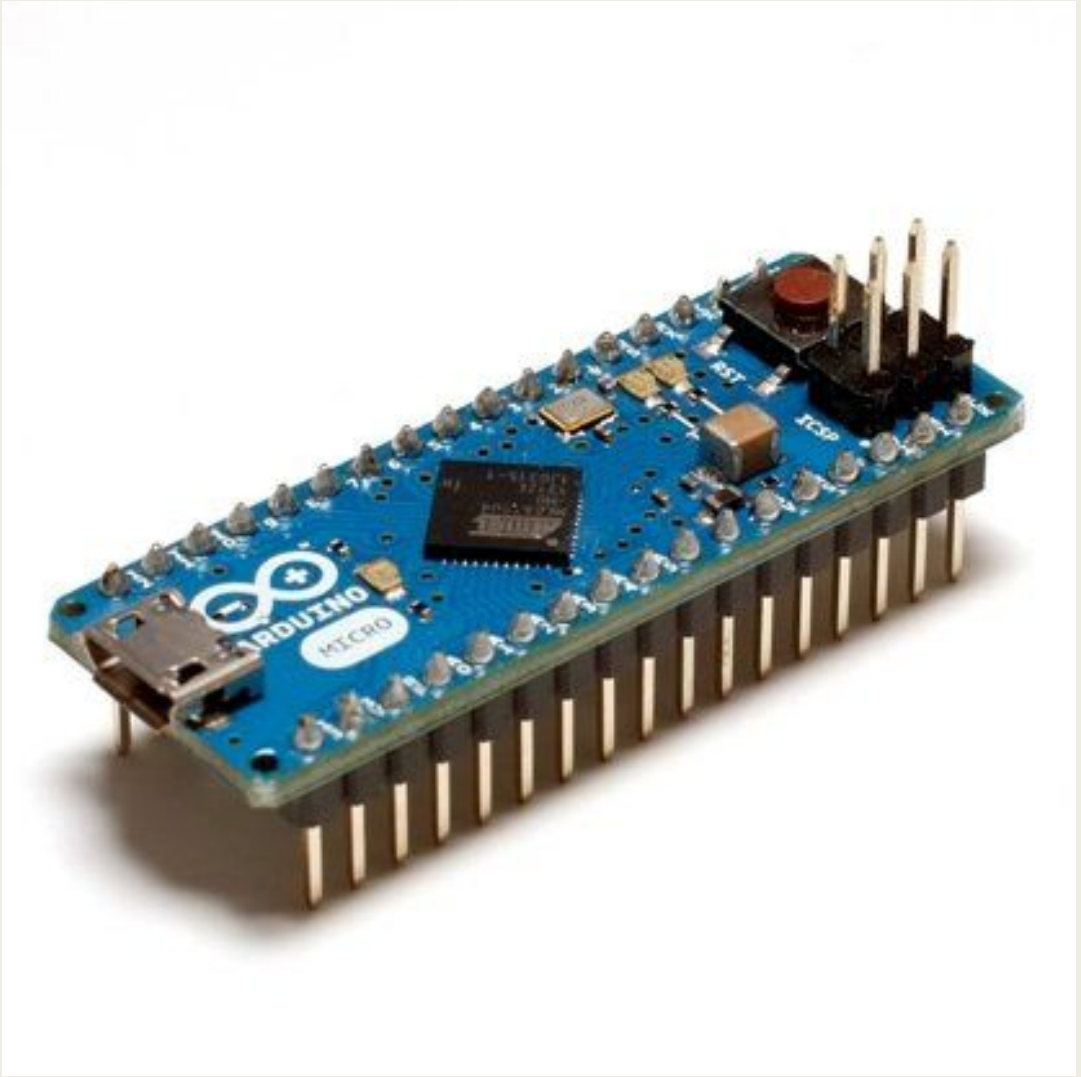
Та же Arduino Uno, но в другом форм-факторе. Компактная: всего 30×18 мм. Из-за форм-фактора нельзя без ухищрений устанавливать платы расширения Arduino.

Предполагается соединение с дополнительными модулями проводами и/или через макетную плату. На плате нет USB-порта, поэтому прошивать нужно через отдельный USB-Serial адаптер.



Arduino Micro

Arduino Micro — это Arduino Leonardo, исполненный на компактной плате. Отличие заключается в отсутствии собственного гнезда для внешнего питания, но оно может быть подведено непосредственно к контакту Vi. В остальном, начинка и способы взаимодействия совпадают с Arduino Leonardo. Он также имеет один микроконтроллер ATmega32u4 и для прошивки через USB, и для исполнения программ; также может выступать в роли клавиатуры или мыши; предоставляет то же количество памяти, цифровых, аналоговых и ШИМ-портов.

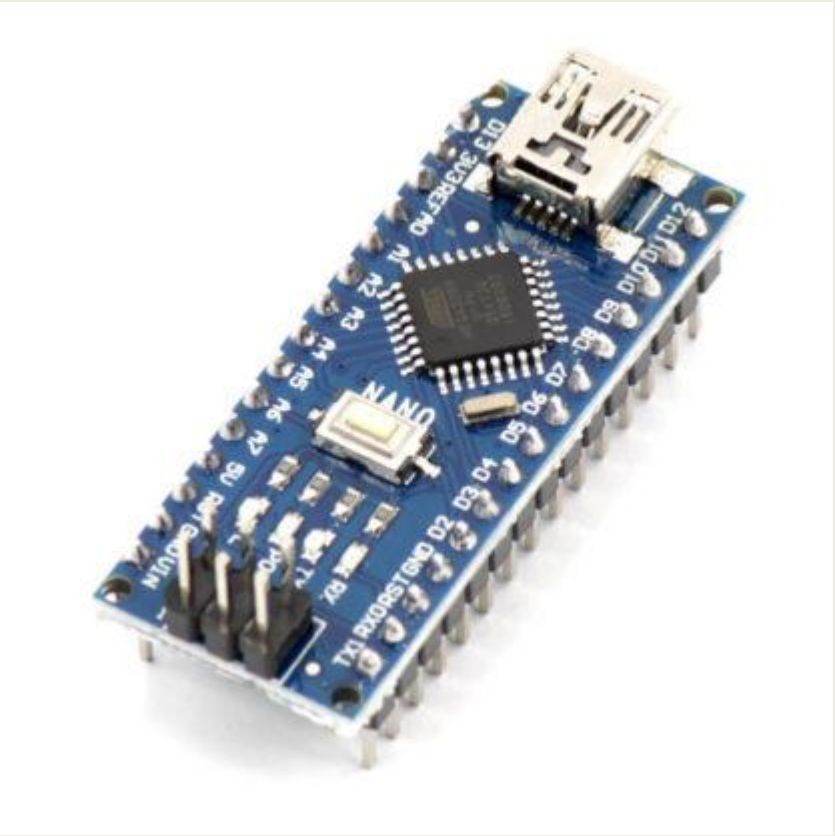


Arduino Nano

Arduino Nano — это функциональный аналог Arduino Uno, но размещённый на миниатюрной плате.

Отличие заключается в отсутствии собственного гнезда для внешнего питания, использованием чипа FTDI FT232RL для USB-Serial преобразования и применением mini-USB кабеля для взаимодействия вместо стандартного.

В остальном, начинка и способы взаимодействия совпадают с базовой моделью. Платформа имеет штырьковые контакты, что позволяет легко устанавливать её на макетную плату.

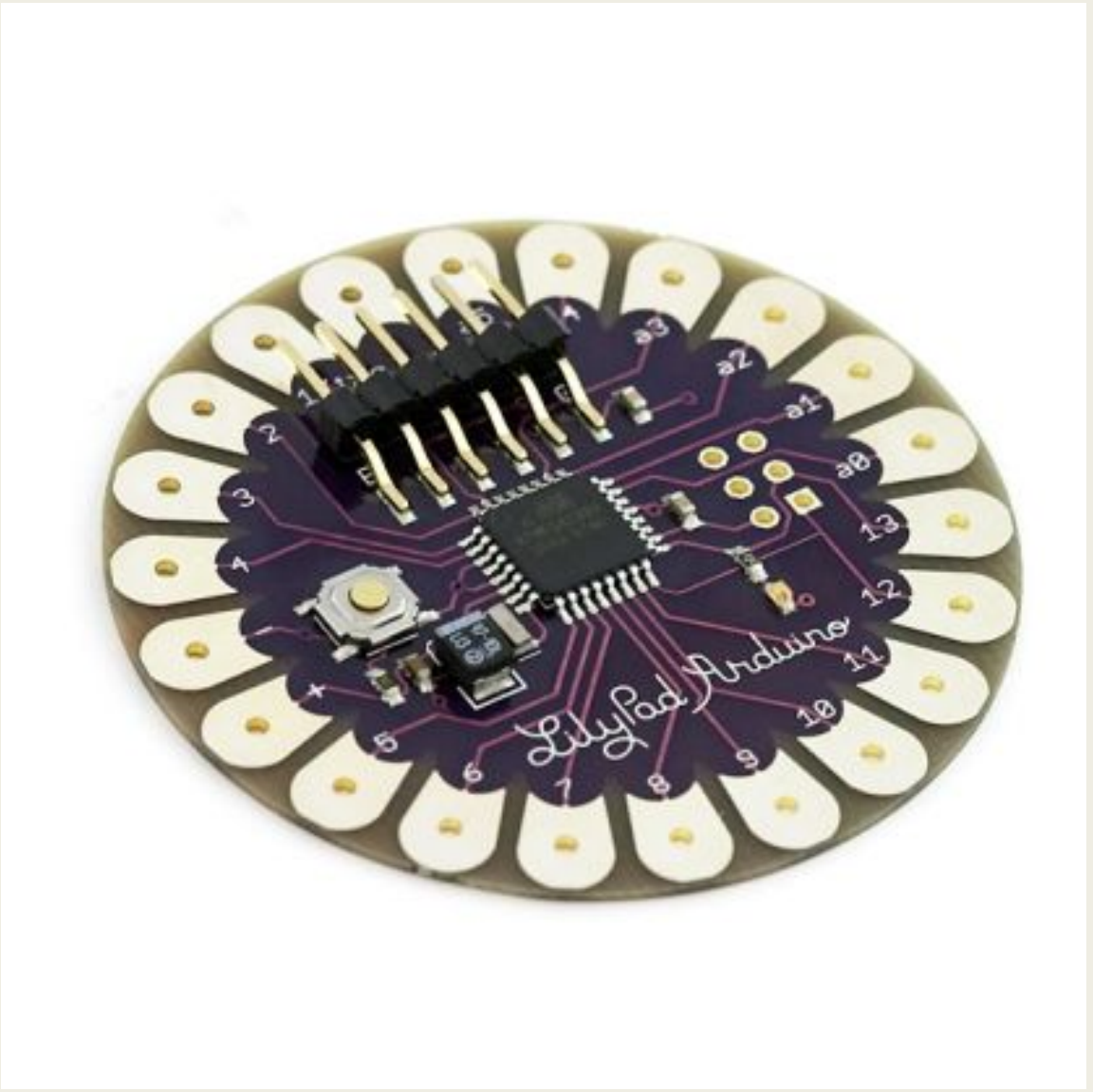


Arduino M0

Забудьте про экономию памяти программ и ресурсов на Arduino Uno. С платой Arduino M0 выполнять сложные математические расчёты, получать более точные аналоговые значения и при этом слушать музыку напрямую с микроконтроллера. Arduino M0 основана на 32-битном ARM-процессоре ATSAMD21G18 от Atmel с вычислительном ядром Cortex® M0. Микроконтроллер работает на частоте 48 МГц. А благодаря своей 32-битной архитектуре он выполняет большинство операций над целыми числами всего за один такт. В отличии от большинства плат Arduino, родным напряжением Arduino M0 Pro является 3.3 В, а не 5 В. Соответственно, выходы для логической единицы выдают 3.3 В, а в режиме входа ожидают принимать не более 3.3 В. Arduino M0 смотрит в сторону USB через виртуальный serial-порт, не через аппаратный. Это означает, что 0-й и 1-й контакты аппаратного порта остаются свободными и вы можете использовать их одновременно с коммуникацией с компьютером. Виртуальный serial-порт доступен через объект SerialUSB, а аппаратный — через объект Serial1.

Arduino LilyPad

Arduino LilyPad — довольно интересное устройство. Оно выпадает из привычных стереотипов об обычном Arduino, потому что имеет не прямоугольную, а круглую форму. Во-вторых, оно не поддерживает механические соединения с шилдами. Оно предназначено для, небольших автономных устройств. Круглая форма продиктовала то, что разъемы равномерно распределены по окружности, и его небольшой размер (2 дюйма в диаметре) делает его идеальным для переносных устройств. Это устройство легко спрятать, и несколько производителей разработали устройства, специально для LilyPad: экраны, датчики света, даже коробки для батарей питания, которые могут быть зашиты в ткань. Для того, чтобы сделать LilyPad как можно меньше и как можно легче, на сколько возможно, были принесены некоторые жертвы. У LilyPad нет регулятора напряжения на борту, так что ему для питания будет необходимо обеспечить по крайней мере 2.7 вольт, и не более 5.5 вольт.



Платы расширения для ардуино

Плата расширения Arduino – это законченное устройство, предназначенное для выполнения определенных функций и подключаемое к основному контроллеру с помощью стандартных разъемов. Такие платы, совершенно логично называемые платами расширения, служат для выполнения самых разнообразных задач и могут существенно упростить жизнь ардуинщика.

Другое популярное название платы расширения – англоязычное Arduino shield или просто шилд. На плате расширения установлены все необходимые электронные компоненты, а взаимодействие с микроконтроллером и другими элементами основной платы происходят через стандартные пины ардуино.

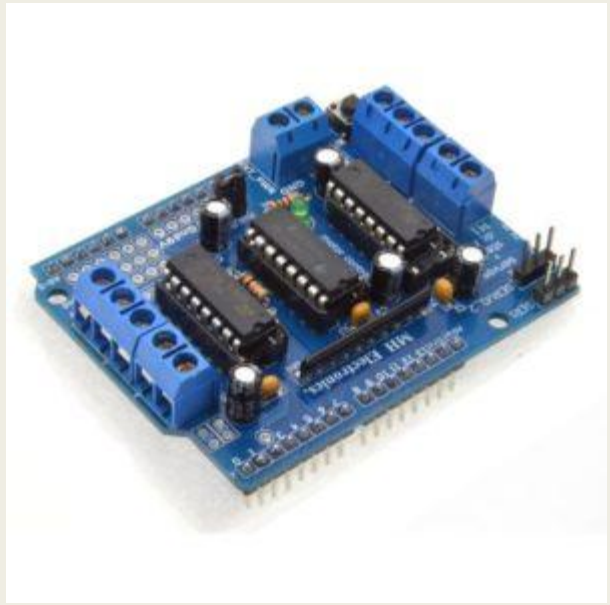
Существует несколько версий плат расширения. Все они отличаются количеством и видом разъемов. Наиболее популярными сегодня являются версии Sensor Shield v4 и v5.

Arduino Sensor Shield

Как правило, эта плата расширения идет в наборах ардуино и поэтому именно с ней ардуинщики встречаются чаще всего. Шилд достаточно прост – его основная задача предоставить более удобные варианты подключения к плате Arduino. Это осуществляется за счет дополнительных разъемов питания и земли, выведенных на плату к каждому из аналоговых и цифровых пинов. Также на плате можно найти разъемы для подключения внешнего источника питания (для переключения нужно установить перемычки), светодиод и кнопка перезапуска.

Arduino Motor Shield

Для чего нужен шилд ардуино очень важен в робототехнических проектах, т.к. позволяет подключать к плате Arduino сразу обычный и серво двигатели. Основная задача шилда – обеспечить управление устройствами потребляющими достаточно высокий для обычной платы ардуино ток. Дополнительным возможностями платы является функция управления мощностью мотора (с помощью ШИМ) и изменения направления вращения. Существует множество разновидностей плат motor shield. Общим для всех них является наличие в схеме мощного транзистора, через который подключается внешняя нагрузка, теплоотводящих элементов (как правило, радиатора), схемы для подключения внешнего питания, разъемов для подключения двигателей и пины для подключения к ардуино.



Arduino Ethernet Shield

Организация работы с сетью – одна из самых важных задач в современных проектах. Для подключения к локальной сети через Ethernet существует соответствующая плата расширения.

Платы расширения для прототипирования

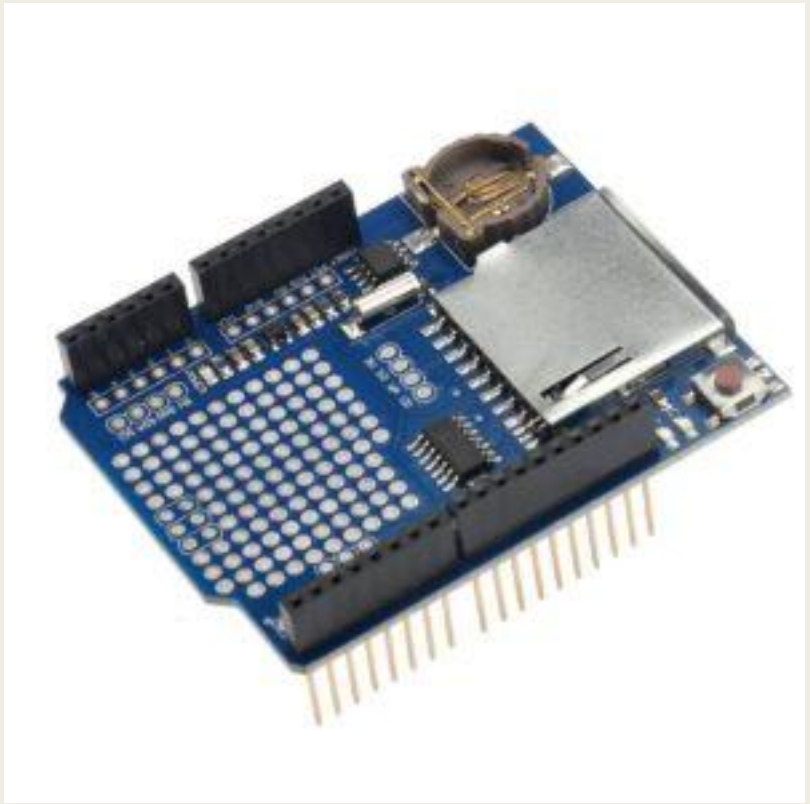
Эти платы достаточно просты – на них расположены контактные площадки для монтажа элементов, выведена кнопка сброса и есть возможность подключения внешнего питания. Предназначение данных шилдов – повысить компактность устройства, когда все необходимые компоненты располагаются сразу над основной платой.

Arduino LCD shield и tft shield

Данный тип шилдов используется для работы с LCD-экранами в ардуино. Как известно, подключение даже самого простого 2-строчного текстового экрана далеко не тривиальная задача: требуется правильно подключить сразу 6 контактов экрана, не считая питания. Гораздо проще вставить готовый модуль в плату ардуино и просто загрузить соответствующий скетч.

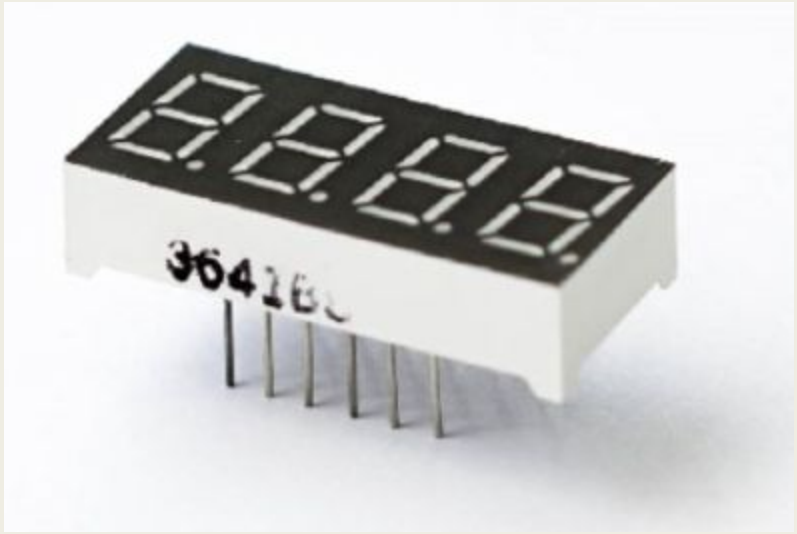
Arduino Data Logger Shield

Еще одна задача, которую достаточно трудно реализовывать самостоятельно в своих изделиях – это сохранение данных, полученных с датчиков, с привязкой по времени. Готовый шилд позволяет не только сохранить данные и получать время со встроенных часов, но и подключить датчики в удобном виде путем пайки или на монтажной плате.

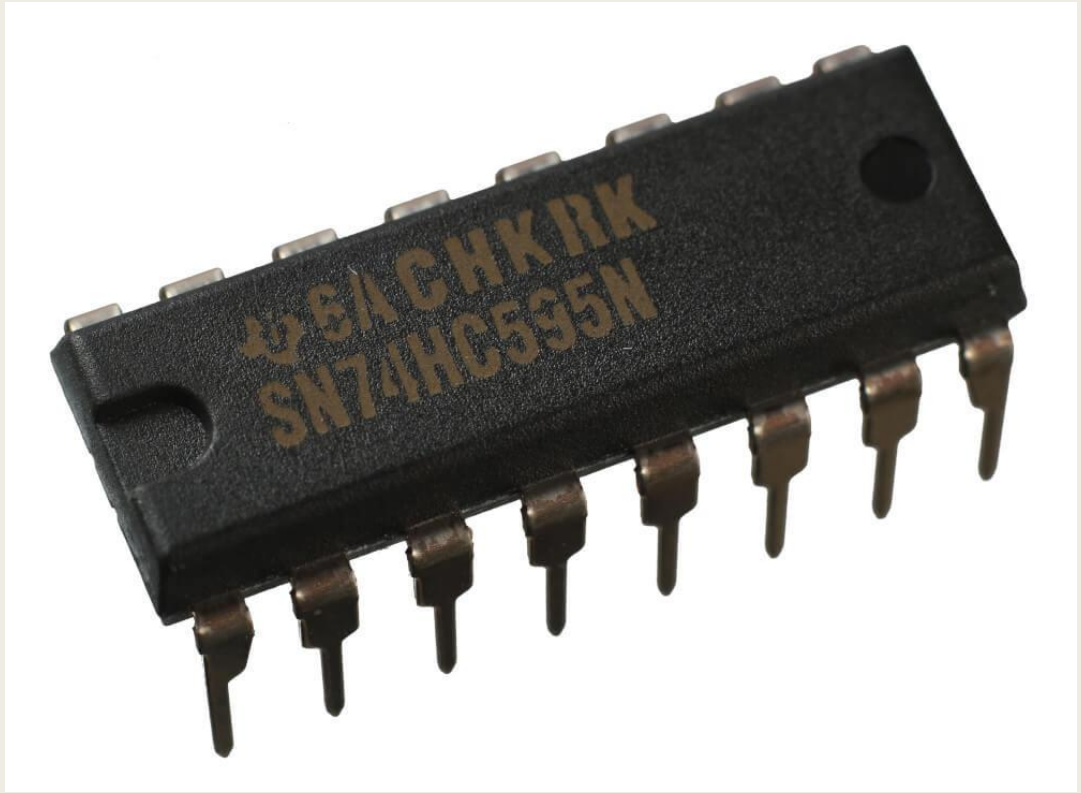


Сдвиговые регистры

Плата Arduino содержит ограниченное число выводов и при сложном проекте их не хватает для полноценной работы. К примеру, для подключения сегментного индикатора необходимо задействовать восемь выводов, два индикатора займут уже 16 выводов. Сдвиговый регистр позволяет сэкономить число используемых выводов, беря часть управления выводами на себя.



Сегментный индикатор



Сдвиговый

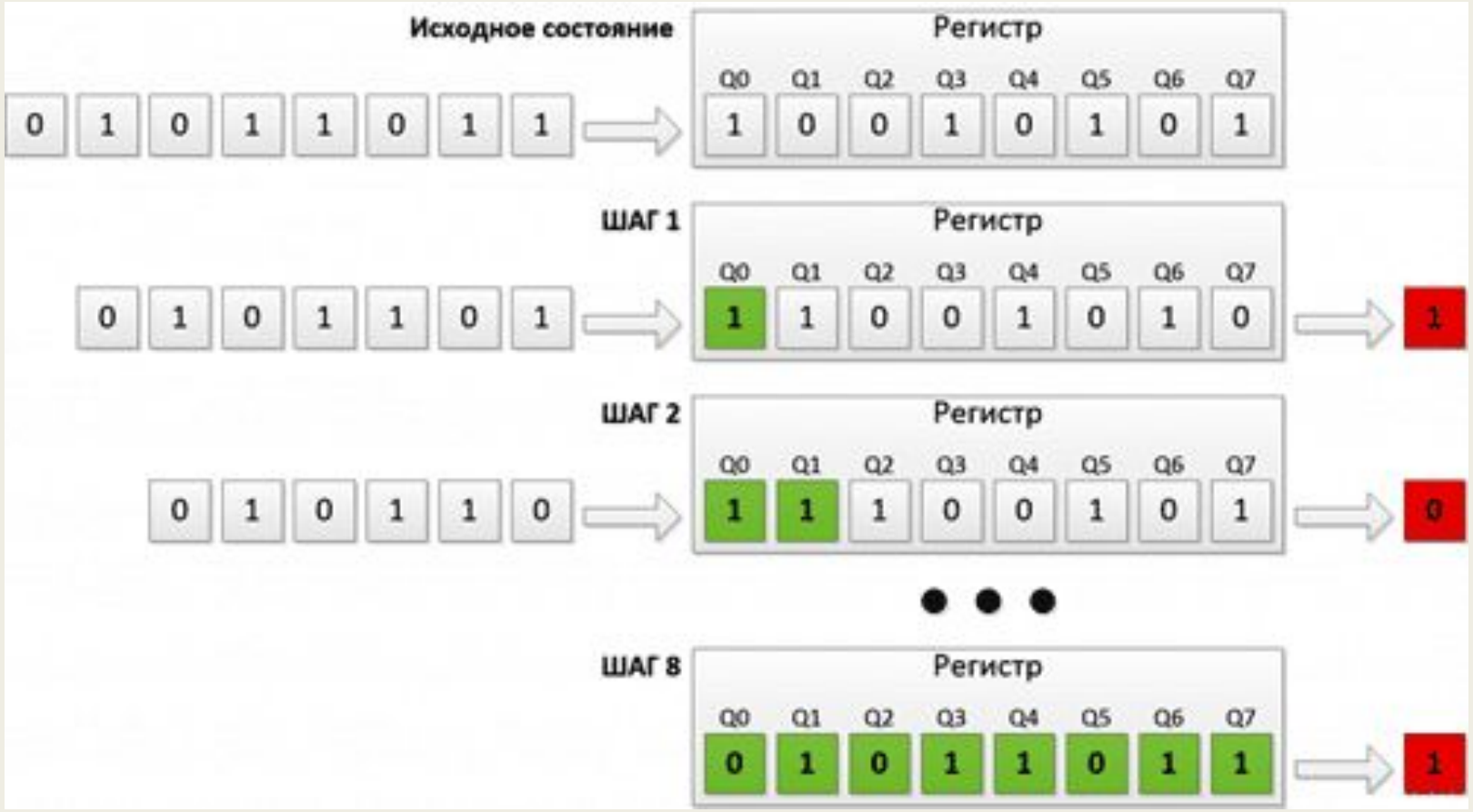
Что такое сдвиговый регистр

В электронике регистром называют устройство, которое может хранить небольшой объем данных для быстрого доступа к ним. Они есть внутри каждого контроллера и микропроцессора, включая и микроконтроллер Atmega328, который входит в состав платы Arduino Uno. Как правило регистры представляют собой сборку из D-триггеров — элементарных ячеек памяти. Записывать данные в регистр можно либо последовательно, либо параллельно. Регистры первого типа называются сдвиговыми, второго типа — параллельными.

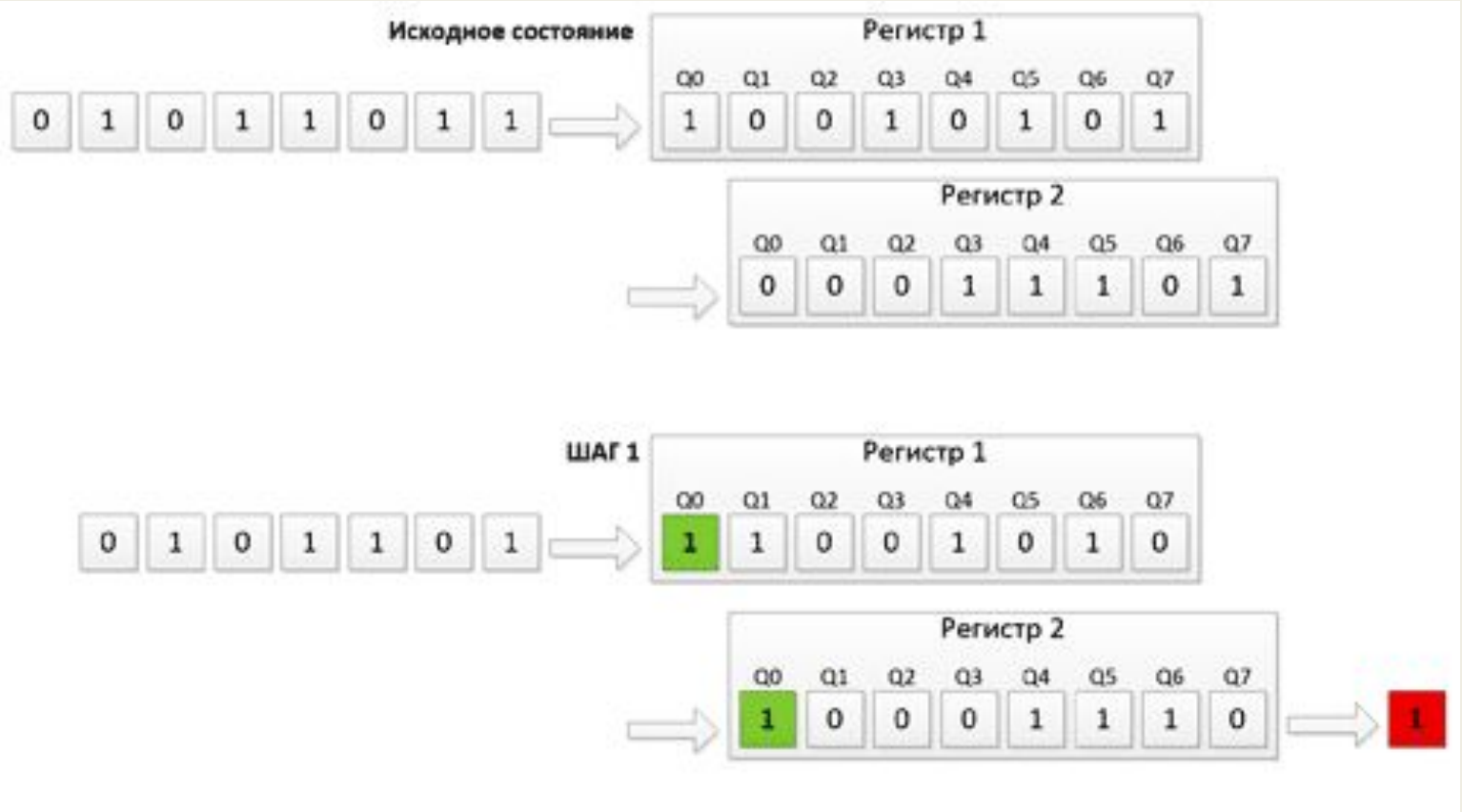
Считывать данные из регистра можно одновременно из всех ячеек. Именно это его свойство помогает нам работать с кучей светодиодов.

Регистр называется сдвиговым, потому что при добавлении каждого нового бита в него, мы как бы сдвигаем все остальные в сторону. Вспомним, что один бит позволяет нам хранить ноль или единицу, истину или ложь. Посмотрим на диаграмме, как это происходит.

Пусть в начальном состоянии регистр уже заполнен какими-то восемью битами. Попробуем «задвинуть» в него восемь новых бит: 11011010.



Регистры можно соединять в цепочку. В таком случае, вытесненный бит не будет пропадать без следа, а отправится в начало следующего регистра. При этом увеличивается число доступных выводов.

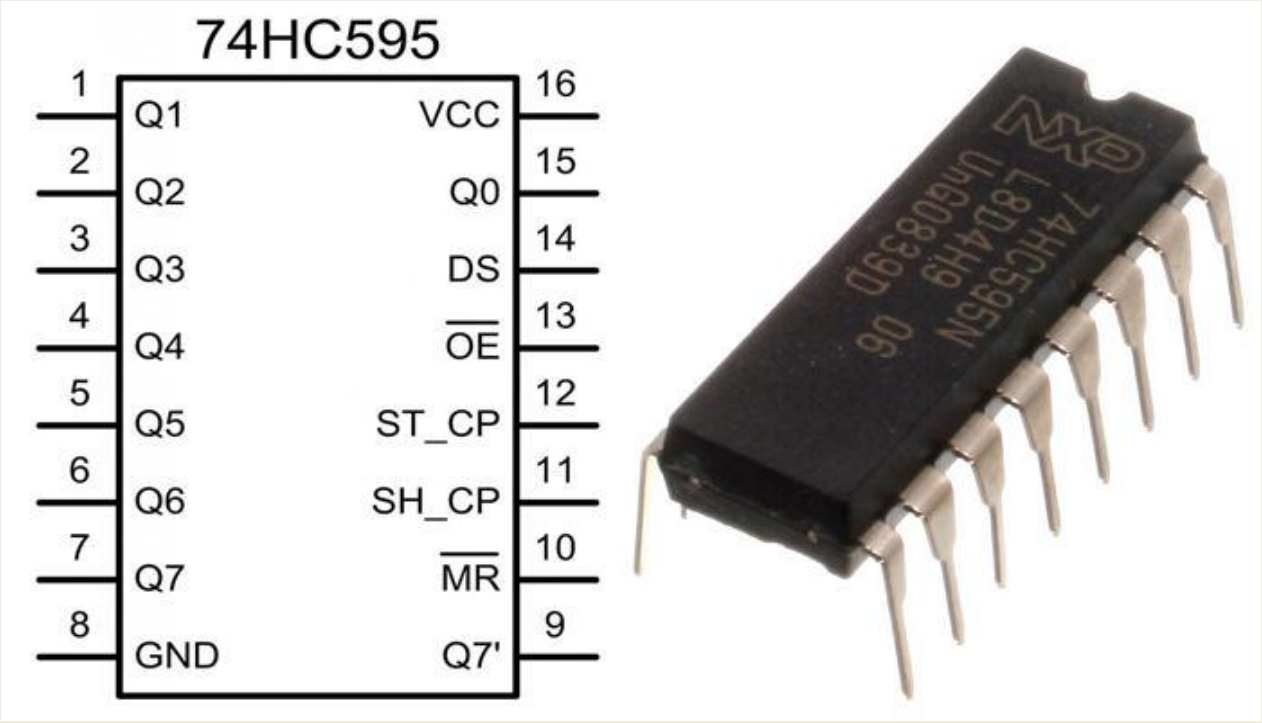


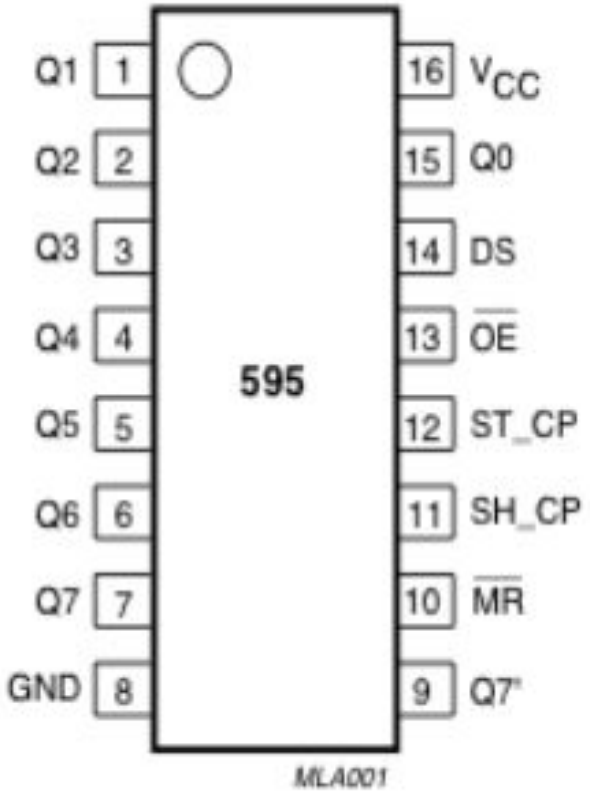
74HC595

Самым популярным является восьмиразрядный сдвиговый регистр 74HC595.

74HC595 — восьмиразрядный (8 управляемых выходов) сдвиговый регистр с последовательным вводом, последовательным или параллельным выводом информации, с триггером-защёлкой и тремя состояниями на выходе.

Другими словами этот регистр позволяет контролировать 8 выходов, используя всего несколько выходов на самом контроллере. При этом несколько таких регистров можно объединять последовательно для каскадирования.



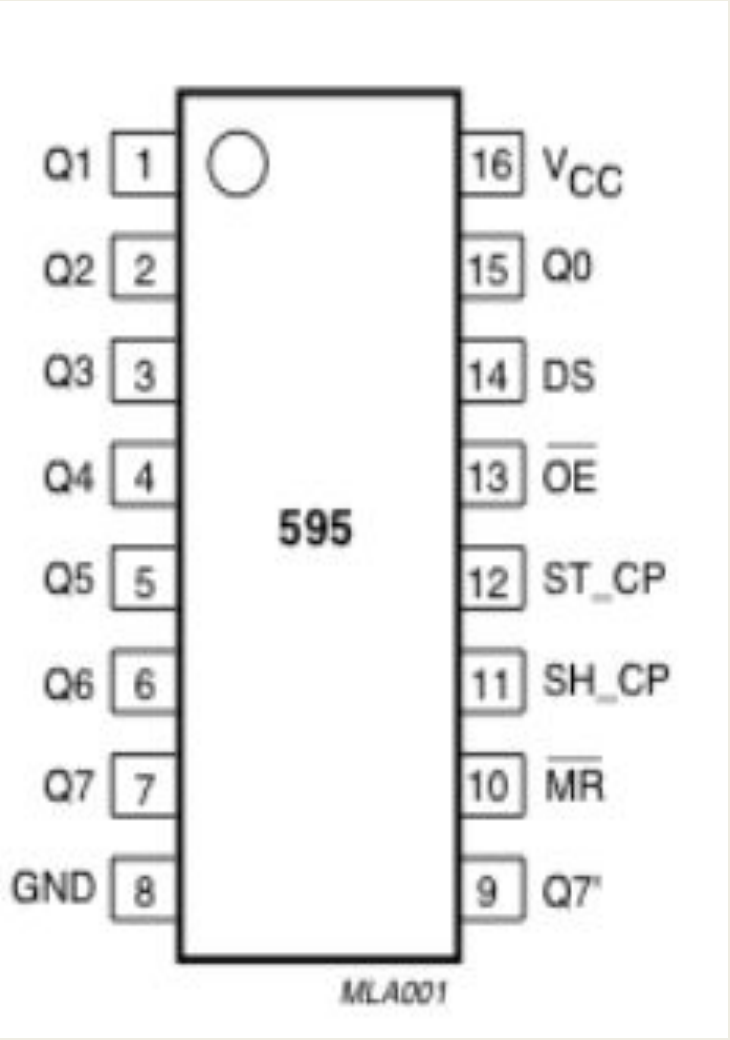


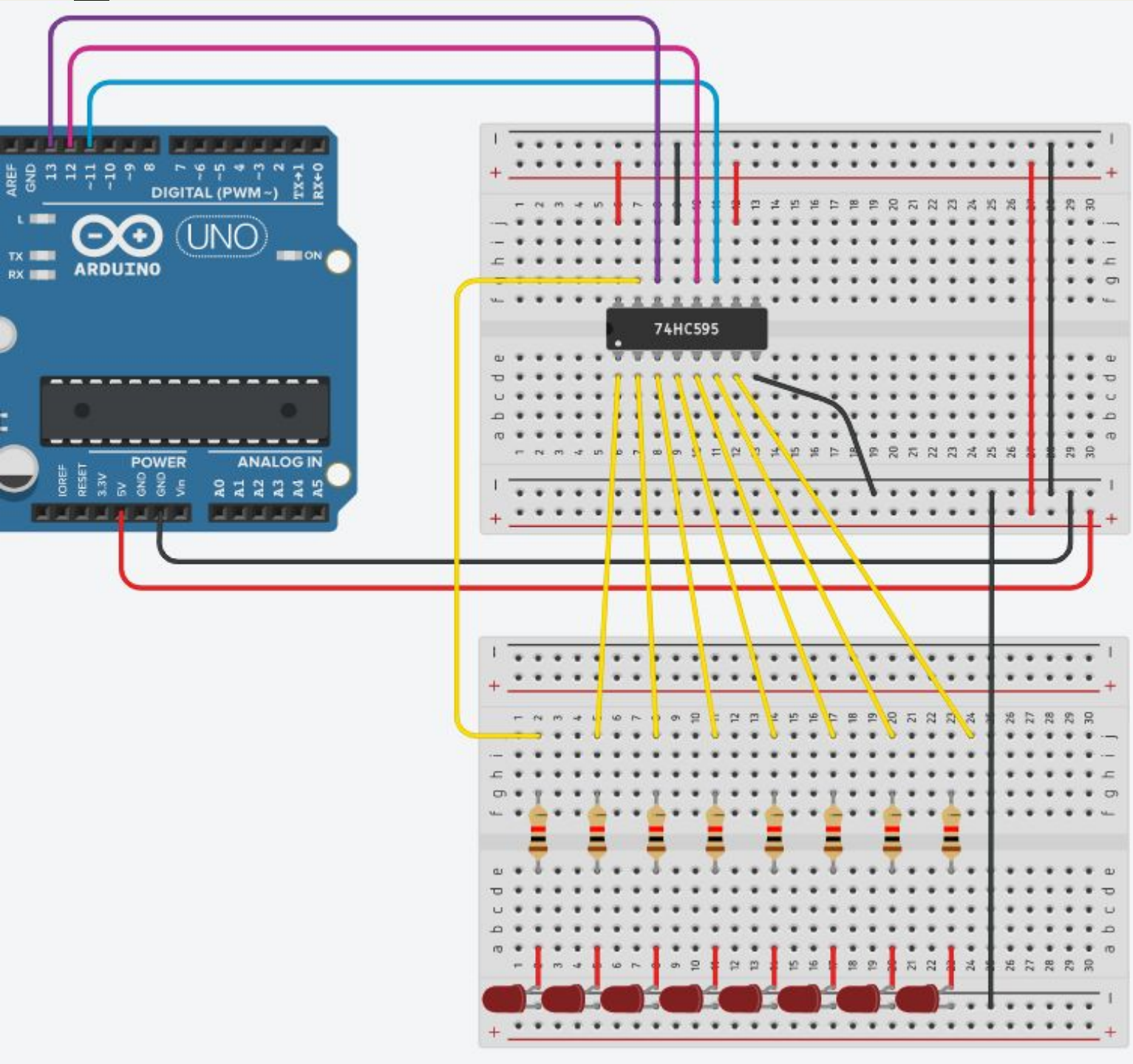
Пины 1-7, 15	Q0 " Q7	Параллельные выходы
Пин 8	GND	Земля
Пин 9	Q7"	Выход для последовательного соединения регистров
Пин 10	MR	Сброс значений регистра. Сброс происходит при получение LOW
Пин 11	SH_CP	Вход для тактовых импульсов
Пин 12	ST_CP	Синронизация ("защелкивание") выходов
Пин 13	OE	Вход для переключения состояния выходов из высокоомного в рабочее
Пин 14	DS	Вход для последовательных данных
Пин 16	Vcc	Питание

Чтобы записать выходы через Arduino, мы должны отправить двоичное значение в регистр сдвига, и из этого числа сдвиговый регистр может определить, какие выходы использовать. Например, если мы отправили двоичное значение 10100010, контакты Q2, Q6 и Q0 будут активными, а остальные будут неактивными.

Это означает, что самый правый бит сопоставляется как Q7, а левый бит сопоставляется с Q0. Выход считается активным, когда бит, сопоставленный с ним, установлен на 1. Важно помнить об этом, так как иначе вам будет очень сложно узнать, какие контакты вы используете.

Теперь, когда у нас есть основное понимание того, как мы используем смещение битов, чтобы указать, какие контакты использовать, мы можем начать подключать его к нашему Arduino.





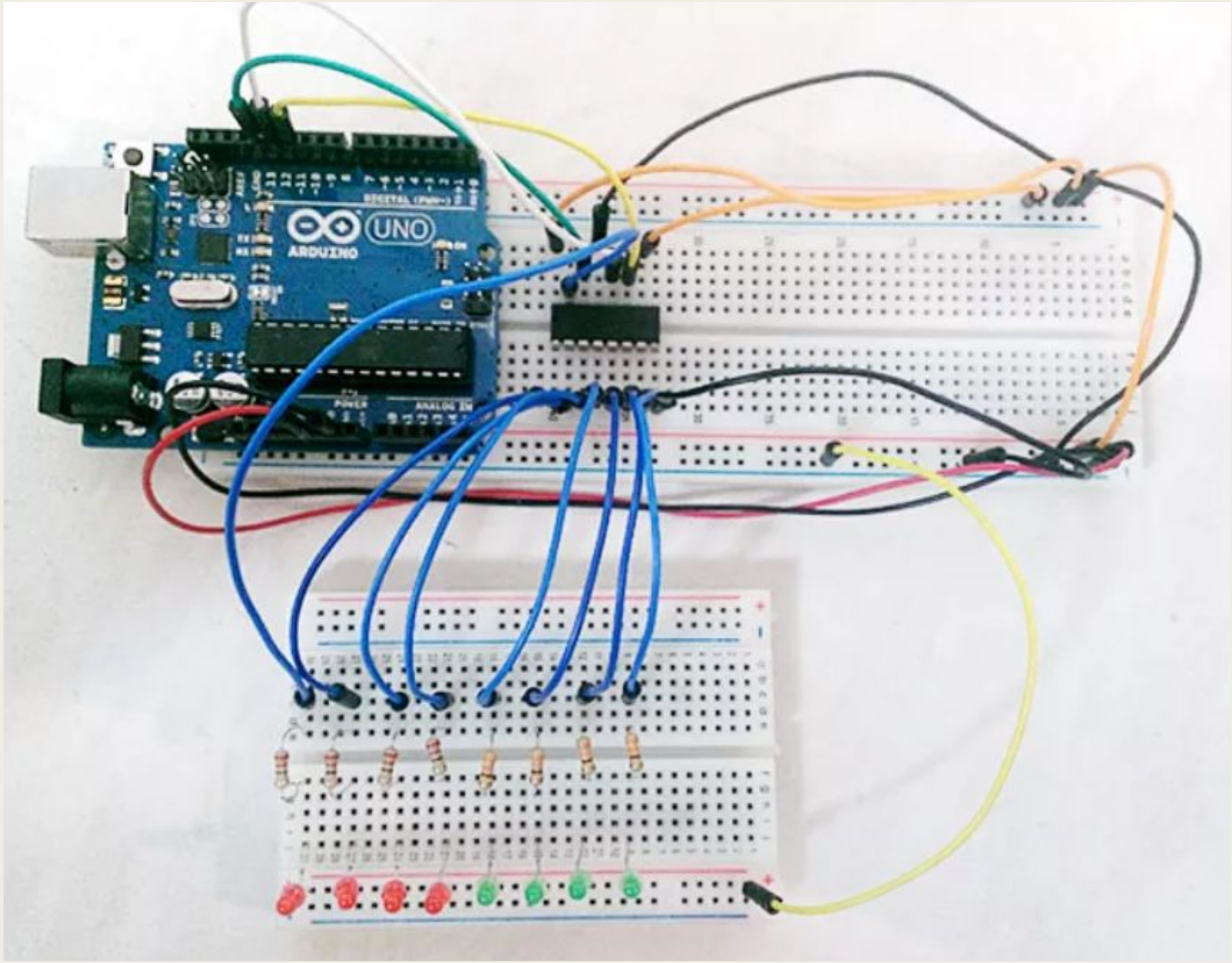
Соберём схему, для которой понадобится сдвиговый регистр и восемь светодиодов с резисторами.

При этом обратите внимание, что в нашем распоряжении восемь выводов регистра для светодиодов, а на плате используем только три цифровых вывода (экономия пяти выводов).

Соединяем три контакта, которыми мы будем управлять сдвиговым регистром:

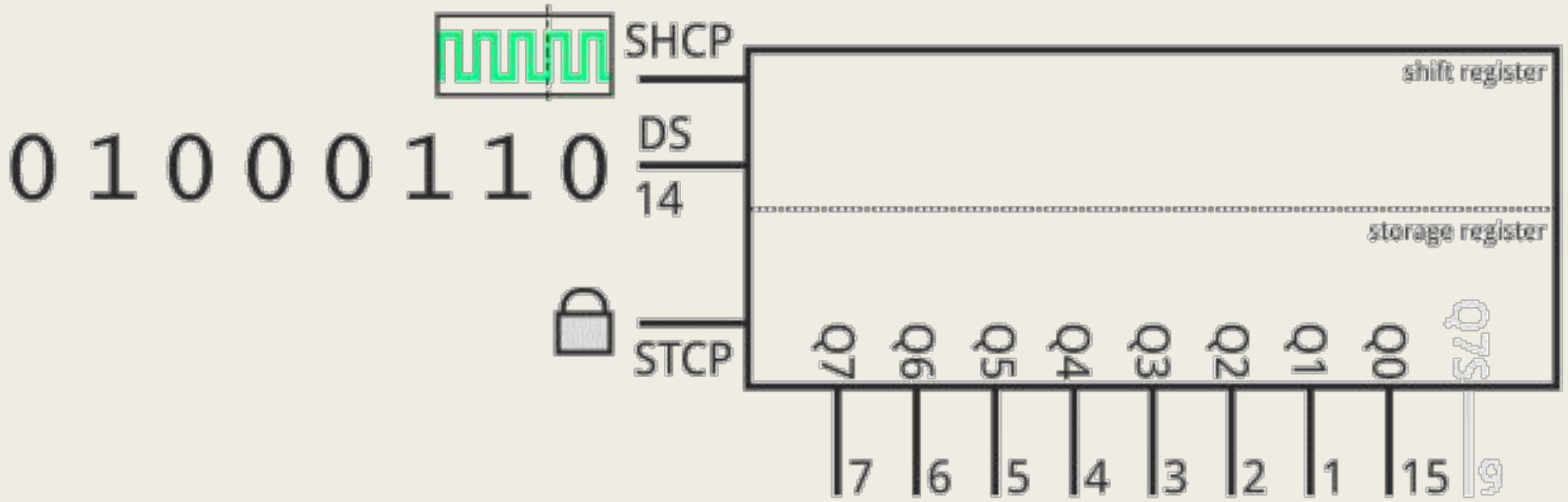
- Вывод 11 (SH_CP, SRCLK) на вывод 11 на Arduino (**синхронизация**)
- Вывод 12 (ST_CP, RCLK) на вывод 12 на Arduino (**защёлка**)
- Вывод 14 (DS, SER) на вывод 9 на Arduino (**данные**)

Вот так бы эта схема выглядела в реальной жизни.



```
int latchPin = 10;  //Пин подключен к ST_CP входу 74HC595
int clockPin = 11;  //Пин подключен к SH_CP входу 74HC595
int dataPin  = 12;  //Пин подключен к DS входу 74HC595
```

```
digitalWrite(latchPin, LOW);  // устанавливаем синхронизацию "защелки" на LOW
shiftOut(dataPin, clockPin, MSBFIRST, numberToDisplay); // передаем последовательно на dataPin
digitalWrite(latchPin, HIGH); //"защелкиваем" регистр, тем самым устанавливая значения на выходах
```



Задание 1. Включаем один светодиод.

Попробуем включить один светодиод. Сначала указываем используемые выводы платы (тактовая линия - clockPin, данные - dataPin, защёлка - latchPin).

В setup() устанавливаем для них режим OUTPUT и ставим защёлке высокий уровень, чтобы регистр не принимал сигналов.

В loop() попробуем что-нибудь отправить на регистр. Сначала ставим LOW на защёлку (начинаем передачу данных. Теперь регистр принимает сигналы с Arduino). Далее отправляем данные в двоичном виде. Например, отправим байт 0b10000000 (должен будет загореться первый светодиод). В конце выставляем HIGH на защёлку (заканчиваем передавать данные).

```
int dataPin = 9;     // к выводу 14 регистра SD
int clockPin = 11;   // к выводу 11 регистра (SH_CP)
int latchPin = 12;   // к выводу 12 регистра (ST_CP)

void setup() {
    pinMode(latchPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
    digitalWrite(latchPin, LOW);
}

void loop() {
    digitalWrite(latchPin, LOW); // начинаем передачу данных
    shiftOut(dataPin, clockPin, LSBFIRST, 0b10000000);
    digitalWrite(latchPin, HIGH); // прекращаем передачу данных
}
```

Если в shiftOut() поменять LSBFIRST на MSBFIRST, то включится не первый, а последний светодиод в цепочке схемы.

В собранной **схеме** есть одна неточность. Ваша задача найти её и исправить. Код верный.

Задание 2. Несколько светодиодов.

При работе с несколькими светодиодами не очень удобно постоянно писать три строчки кода для каждого светодиода в отдельности. Поэтому оформим код в виде функции и будем мигать третьим светодиодом.

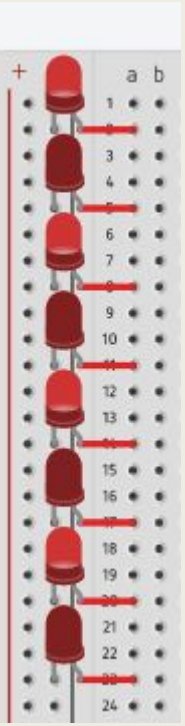
```
int dataPin = 9;     // к выводу 14 регистра SD
int clockPin = 11;   // к выводу 11 регистра (SH_CP)
int latchPin = 12;   // к выводу 12 регистра (ST_CP)

void setup() {
    pinMode(latchPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
    digitalWrite(latchPin, LOW);
}

void loop() {
    setByte(0b00100000);
    delay(1000);
    setByte(0b00000000);
    delay(1000);
}

void setByte(byte value) {
    digitalWrite(latchPin, LOW); // начинаем передачу данных
    // устанавливаем нужный байт
    shiftOut(dataPin, clockPin, LSBFIRST, value);
    digitalWrite(latchPin, HIGH); // прекращаем передачу данных
}
```

Сделайте так, что бы светодиоды замигали в следующем порядке:



Задание 3. Анимация

```
int dataPin = 9;     // к выводу 14 регистра
int clockPin = 11;   // к выводу 11 регистра (SH_CP)
int latchPin = 12;   // к выводу 12 регистра (ST_CP)

byte path[3] = {
  B11000011,
  B00111100,
  B00100100
};

void setup() {
    pinMode(latchPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
}

void loop() {
    for (int i = 0; i < 3; i++) {
        digitalWrite(latchPin, LOW);
        shiftOut(dataPin, clockPin, LSBFIRST, path[i]);
        digitalWrite(latchPin, HIGH);
        delay(250);
    }
}
```

Добавьте несколько разных вариантов включения светодиодов (добавьте еще 4 элемента массива).

Задание 4. Последовательноее мигание светодиодов

```
int dataPin = 9;     // к выводу 14 регистра
int clockPin = 11;   // к выводу 11 регистра (SH_CP)
int latchPin = 12;   // к выводу 12 регистра (ST_CP)

void setup() {
    pinMode(latchPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
}

void loop() {
    byte Bt = 0; //Создаем пустой байт B00000000
    for (int i = 0; i < 8; i++) { // В переменной хранится позиция изменяемого бита
        Bt = 0; // Обнуляем байт при каждом проходе
        bitWrite(Bt, i, HIGH); // При i=0 получим B00000001, при i=1 - B00000010, при i=2 - B00000100 и т.д.

        digitalWrite(latchPin, LOW);
        shiftOut(dataPin, clockPin, MSBFIRST, Bt); // Инвертируем сигнал при помощи MSBFIRST, грузим с первого бита
        digitalWrite(latchPin, HIGH);
        delay(50);
    }
}
```

Добавьте в схему второй сдвиговый регистр, подключите к нему так же 8 светодиодов (пусть они будут другого цвета).