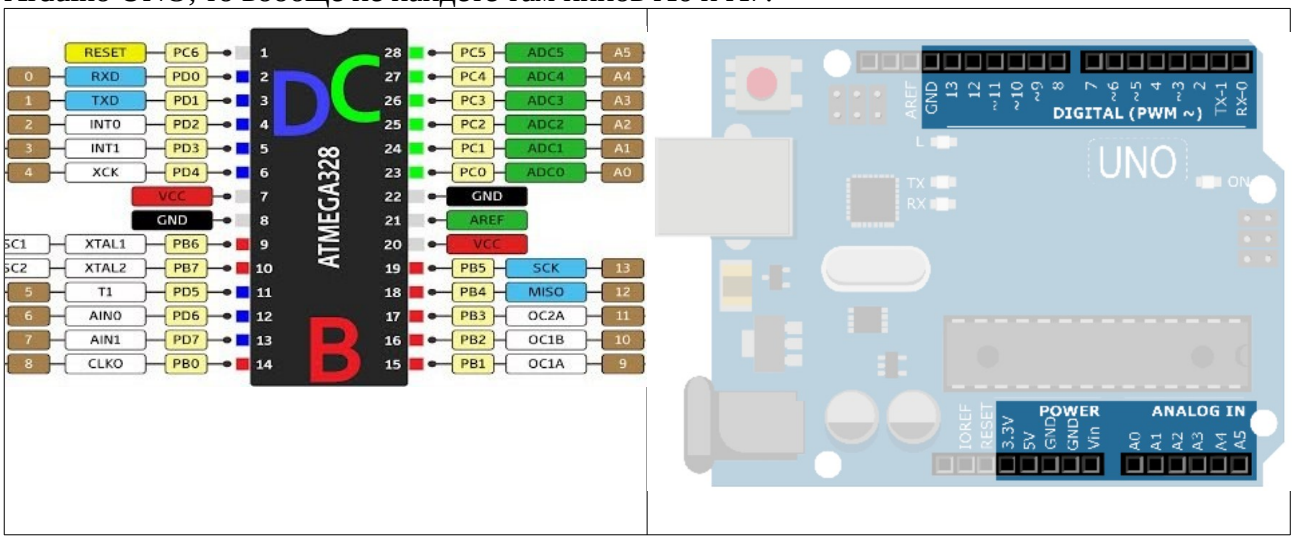
Лабораторная работа №2
GPIO, управление портами ввода-ввывода в Arduino

GPIO

GPIO, с англ. General Purpose Input-Output, входы-выходы общего назначения, на плате они подписаны как D0–D13 и A0–A5. Они называются PD*, PB* и PC*, (вместо звёздочки – цифра). Почему “официально” они называются PD/PB/PC? Потому что пины объединены в пОрты по несколько штук (не более 8), на примере Нано есть три порта: D, B и C, соответственно пины так и подписаны: PD3 – Port D 3 – третий выход порта D. Это цифровые пины, способные выдавать логический сигнал (0 или VCC) и считывать такой же логический сигнал. VCC это напряжение питания микроконтроллера, при обычном использовании обычной платы Ардуино это 5 Вольт, соответственно это 5 вольтовая логика: 0V – сигнал низкого уровня (LOW), 5V – высокого уровня (HIGH). Напряжение питания микроконтроллера играет очень большую роль, об этом мы ещё поговорим.

На плате пины подписаны по-другому, просто по порядку. Таким образом мы видим, что все пины с D0-D13 и A0-A5 являются GPIO, то есть цифровыми входами-выходами. Многие называют пины A0–A7 аналоговыми, на некоторых неофициальных распиновках они прям подписаны как analog pin, и это вводит новичков в заблуждение, потому что A0-A5 являются такими же цифровыми пинами, как D0-D13. Но у этих пинов есть дополнительная функция в виде чтения аналогового сигнала. А вот пины A6 и A7 являются именно аналоговыми, потому что у них есть только выход на АЦП, эти пины не являются GPIO, и с ними нельзя работать функциями для цифровых пинов. Если вы посмотрите на распиновку Arduino UNO, то вообще не найдёте там пинов A6 и A7.



Нумерация пинов

Пины пронумерованы на плате как “цифровые” D* пины и аналоговые A* пины. К цифровым пинам мы будем обращаться просто по их номеру, т.е. D3 это просто 3. С аналоговыми пинами чуть сложнее:

- Обратиться можно с буквой A (A3, A5)
- Можно цифрой по порядку после цифровых, так например у Нано последний цифровой – D13, следующий за ним “аналоговый” A0 имеет номер 14, а например A5 имеет номер 19, по которому к нему тоже можно обратиться, что позволяет управлять всеми пинами при помощи циклов.

Режимы работы пинов

Цифровой пин может находиться в двух состояниях, вход и выход. В режиме входа пин может считывать напряжение от 0 до напряжения питания МК, а в режиме выхода – выдавать такое же напряжение. Режим работы выбирается при помощи функции **pinMode(pin, mode),** где pin это номер пина, а mode это режим:

mode – режим работы

Определены следующие режимы работы GPIO:

INPUT – вход

OUTPUT – выход

INPUT_PULLUP – подтянутый к питанию вход

Если со входом/выходом всё понятно, то с подтяжкой давайте разберёмся. В режиме входа пин микроконтроллера не подключен никуда и ловит из воздуха всякие наводки, получая практически случайное значение. Для задания пину “состояния по умолчанию” используют подтяжку резистором к земле или питанию. Вот режим INPUT_PULLUP включает встроенную в микроконтроллер подтяжку пина к питанию.

По умолчанию все пины сконфигурированы как входы (INPUT)

Используя информацию из предыдущего пункта и урока о циклах, можно изменить режим работы например для всех пинов с D2 по A5:

```
for (byte i = 2; i <= 19; i++) {   // с 2 по 19 (D2-D13, A0-A5)
  pinMode(i, OUTPUT);              // делаем выходами
}
```

Вывод цифрового сигнала

Цифровой пин в режиме выхода (OUTPUT) может генерировать цифровой сигнал, т.е. выдавать напряжение. Так как понятие “цифровой” обычно связано с двумя состояниями, 0 и 1, цифровой пин может выдать 0 или 1, точнее: сигнал низкого или высокого уровня. Сигнал низкого уровня это 0 Вольт, грубо говоря в этом состоянии пин подключается к GND микроконтроллера. Сигнал высокого уровня подключает пин к VCC микроконтроллера, то есть к питанию. Если вы вспомните урок по питанию платы, то поймёте, что сигнал высокого уровня на цифровом пине будет варьироваться в зависимости от того, как питается плата Arduino. При питании от источника 5V на пине будет 5V, при питании от USB с потерей на защитном диоде мы получим около 4.7 Вольт на цифровом пине в режиме выхода с высоким сигналом.

Самый главный момент касательно цифровых пинов: микроконтроллер – это логическое устройство, которое создано для управления другими устройствами при помощи логических (цифровых) сигналов. Под словом логическое я подразумеваю не силовое, то есть питать что-то от микроконтроллера нельзя, за редким исключением. На картинке с распиновкой выше вы можете найти надпись “Absolute MAX per pin 40mA, recommended 20mA“. Это означает, что максимум можно снять с пина 40 миллиампер, а рекомендуется не больше 20 миллиампер. Поверьте, для микроконтроллера это очень много. В других микроконтроллерах ограничение по току на пин может составлять 5-10 мА. Также есть общее ограничение на ток с цифровых пинов – 200 мА: “Absolute MAX 200mA for entire package“. Эту информацию можно найти в любом официальном источнике информации об Arduino и микроконтроллере в целом, в том числе в даташите на микроконтроллер.

Что произойдёт, если снять с пина больше, чем он может отдать? Всё очень просто – он сгорит. Что будет, если снять с нескольких пинов больше, чем может отдать микроконтроллер в целом? Правильно – сгорит микроконтроллер. Поэтому ничего мощнее светодиода и маленькой пищалки к микроконтроллеру подключать нельзя. Никаких моторчиков, лампочек, нагревателей, мощных радио-модулей и прочего питать от цифровых пинов нельзя. Цифровые пины служат для подачи команд другим устройствам, например реле/транзисторам для коммутации нагрузок. Но об этом мы поговорим отдельно.

Сейчас вернёмся к вопросу подачи цифрового сигнала: для этого у нас есть функция digitalWrite(pin, value):

pin – цифровой пин МК, подписанный на плате как D.

value – уровень сигнала: HIGH высокий, LOW низкий. Также можно использовать цифры 0 и 1

Пример, в котором пины инициализируются как выходы, и на них подаётся сигнал:

```
void setup() {
  pinMode(10, OUTPUT);  // D10 как выход
  pinMode(A3, OUTPUT);  // A3 как выход
  pinMode(19, OUTPUT);  // A5 как выход (Nano/UNO)
  digitalWrite(10, HIGH); // высокий сигнал на D10
  digitalWrite(A3, 1);    // высокий сигнал на A3
  digitalWrite(19, 1);    // высокий сигнал на A5
}
void loop() {}
```

Пин, настроенный как OUTPUT, по умолчанию имеет сигнал LOW

Ещё интересный момент: в старых версиях IDE не было варианта режима работы INPUT_PULLUP, и подтяжка делалась вручную. Запомните, что вот эти два варианта являются равноценными, вы можете встретить второй в старых скетчах из Интернета, не пугайтесь. Оба варианта делают пин подтянутым к питанию в режиме входа

```
// современный вариант
pinMode(10, INPUT_PULLUP);  // D10 как подтянутый вход
// старый вариант
pinMode(10, INPUT);     // D10 как вход
digitalWrite(10, HIGH); // "подтянуть" D10
```

Чтение цифрового сигнала

Цифровой пин может “измерять” напряжение, но сообщить он может только о его отсутствии (сигнал низкого уровня, LOW) или наличии (сигнал высокого уровня, HIGH), причём отсутствием напряжения считается промежуток от 0 до ~2.1V. Соответственно от ~2.1V до VCC (до 5V) микроконтроллер считает за наличие сигнала высокого уровня. Таким образом микроконтроллер спокойно может работать с логическими устройствами, которые шлют ему высокий сигнал с напряжением 3.3V, он такой сигнал примет как HIGH.

Нельзя подавать на цифровой пин (да и на любой другой пин тоже) напряжение выше напряжения питания микроконтроллера.

Для чтения уровня сигнала на пине используется функция digitalRead(pin), где пин – номер пина согласно подписи на плате. Это пины, подписанные как D, а также пины A0-A5 у Arduino Nano/Uno/Pro Mini. Данная функция возвращает 0, если сигнал низкого уровня, и 1 – если высокого. Простой пример:

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.println(digitalRead(5));
}
```

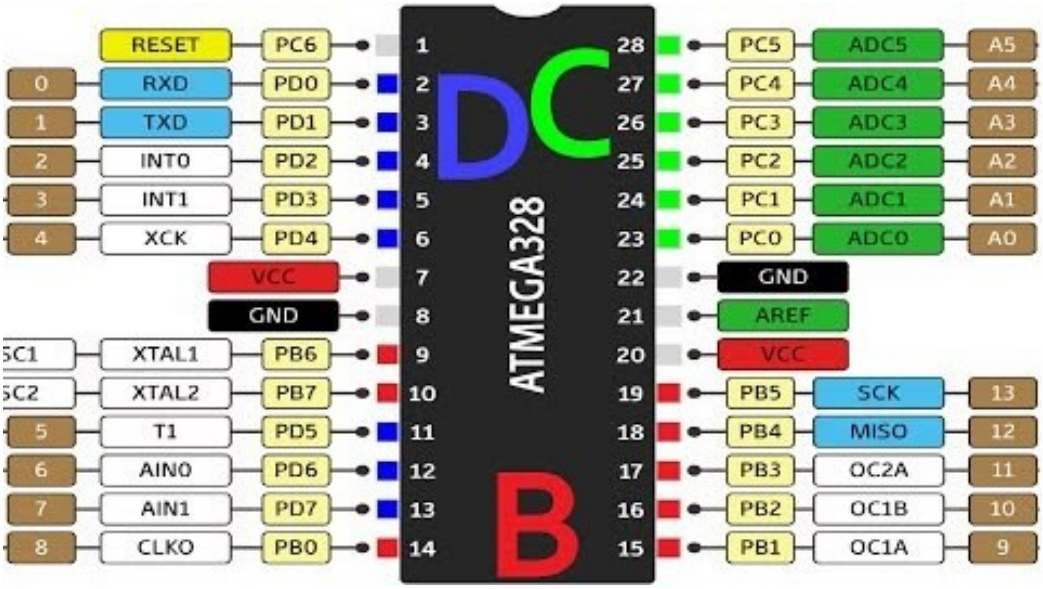
Данный код будет выводить в порт сигнал на пине D5. Если подключить его проводом к VCC – получим 1, если к GND – получим 0.

Альтернативный способ – управление портами через регистры Atmega и ускорение работы кода

Управление через стандартные функции Arduino конечно простое, но в этом случае есть два недостатка – большее потребление памяти и низкое быстродействие при работе с портами. Но вспомните что такое Arduino независимо от варианта платы (uno, micro, nano)? В первую очередь, это микроконтроллер AVR семейства ATMEGA, в последнее время используется МК atmega328.

В Arduino IDE вы можете программировать на родном для этого семейства языке C AVR, так, как если бы вы работали с отдельным микроконтроллером. Но обо всем по

порядку. Чтобы управлять портами Ардуино таким образом вам нужно сначала внимательно



рассмотреть следующую иллюстрацию.

Здесь вы видите соответствие выводов Ардуино и названий портов атмеги. Итак, у нас есть 3 порта:

PORTB;
PORTC;
PORTD.

Пин Atmega 328\168	Пин Arduino
PORTD	
PD0	RX (0)
PD1	TX (1)
PD2	2
PD3	3(PWM)
PD4	4
PD5	5 (PWM)
PD6	6 (PWM)
PD7	7
PORTC	
PC0	A0
PC1	A1
PC2	A2
PC3	A3
PC4	A4
PC5	A5
PC6	RESET
PORTB	
PB0	8
PB1	9 (PWM)
PB2	10 (PWM)
PB3	11 (PWM)
PB4	12
PB5	13
PB6	КВ. РЕЗ.
PB7	КВ. РЕЗ.

Исходя из изображенного на рисунках, я составил таблицу соответствия портов Ардуино и Атмеги, она пригодится вам в дальнейшем.У Atmega есть три регистра длиной в 8 бит, которые управляют состоянием портов, например, порта B разберемся в их назначении проведя аналогии со стандартными средствами wiring описанными в начале статьи:

PORTB – Управление состоянием вывода. Если пин находится в режиме «Выхода», то 1 и 0 определяют наличие этих же сигналов на выходе. Если же пин находится в режиме «Входа», то 1 подключает подтягивающий резистор (тоже что и INPUT_PULLUP рассмотренный выше), если 0 – высокоимпедансное состояние (аналог INPUT);

PINB – регистр чтения. Соответственно в нём находится информация о текущем состоянии выводов порта (логическая единица или ноль).

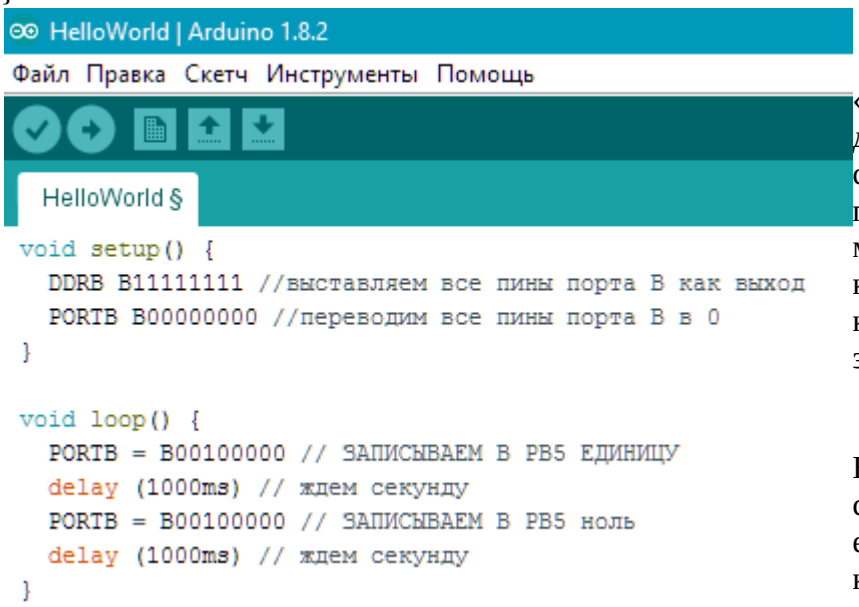
DDRB – регистр направления порта. С его помощью вы указываете микроконтроллеру чем является порт – входом или выходом, при этом «1» - выход, а «0» - вход.

Вместо буквы «B» может быть любая другая согласно названиям портов, например, PORTD или PORTC аналогично работают и другие команды.

Помигаем светодиодом, заменим стандартную функцию digitalWrite(). Для начала вспомним как выглядит исходный пример из библиотеки Arduino IDE.

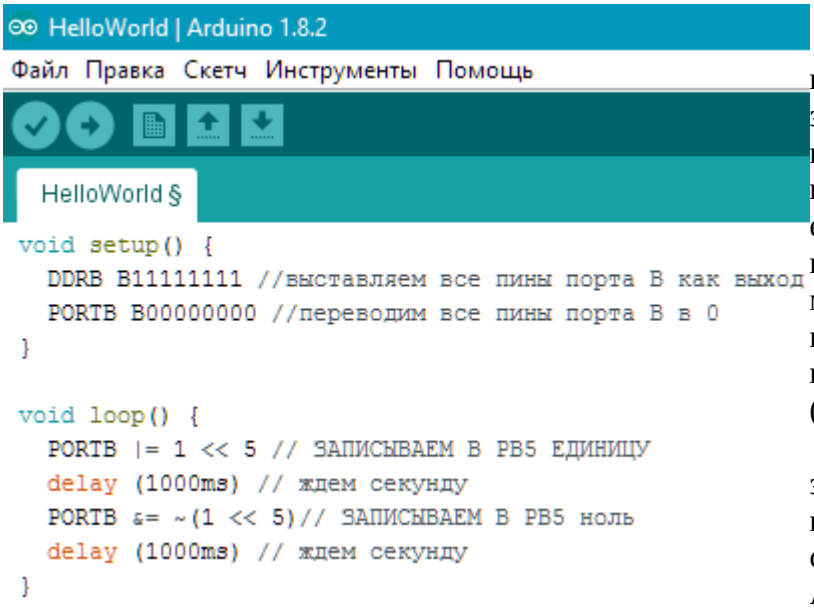
```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```



Это код всем известного «blink», который демонстрирует мигание светодиодом, встроенным в плату через регистры микроконтроллера.В комментариях даны пояснения к коду. Логика такой работы заключается в следующем.

Команда PORTB B00100000 переводит PB5 в состояние логической единицы, смотрим, а те картинки и таблицу что расположены ниже и видим, что PB5 соответствует 13 пину Ардуины.



Буква «B» перед цифрами говорит о том, что мы в записываем значения в двоичном виде. Нумерация в двоичном коде идёт справа налево, т.е. здесь единица стоит в шестом с правого края бите, что говорит микроконтроллеру о взаимодействии с состоянием шестого бита регистра порта B (PB5).

Вы можете задавать значение не в двоичном, а в шестнадцатеричном виде.В таком случае переносим это всё в Arduino IDE, но уже вместо приставки «B» будет «0x».

```
void loop() {
  PORTB 0x10// ЗАПИСЫВАЕМ В PB5 ЕДИНИЦУ
  delay (1000ms) // ждем секунду
```

Но при таком вводе возникает проблема. Если у вас к другим пинам подключено что-либо, то внося команду типа B00010000 – вы все выводы кроме 13 (PB5) обнулите. Вы можете вносить данные на каждый пин по отдельности. Это будет выглядеть следующим образом:Операция логического сложения, |= значит прибавить к содержимому порту что-либо.

А это значит, что нужно сложить слово из 8 бит в регистре с единицей, смещенной на 5 бит – в результате, если было 11000010 получается 11010010. На этом примере видно, что изменился только PB5, остальные биты этого регистра остались без изменений, как и остались неизменными состояния выводов микроконтроллера.

Но при логическом сложении возникает проблема – вы не можете превратить единицу в ноль, потому что:

0+0=1

1+0=1

0+1=1

Нам на помощь придёт логическое умножение и инвертирование:

&= значит умножить содержимое порта на определенное число.

А это число, на которое мы умножает. Знак «~» обозначает инвертирование. В нашем случае проинвертированная единица является нулем. То есть мы умножаем содержимое порта на ноль, сдвинутый на 5 бит. Например, было 10110001, стало 10100001. Остальные биты остались без изменений.

Чтение с портов, аналог digitalRead() выполняют с помощью регистра PIN, на практике это выглядит так:

```
void setup() {
  DDRB B11101111 //выставляем все пины порта B как выход, A PB4 КАК ВХОД
  PORTB B00000000 //переводим все пины порта B в 0
}

void loop() {
  if (PINB==B00010000) //ЕСЛИ ВЫРАЖЕНИЕ В СКОБКАХ ВЕРНО, ТО ВЫПОЛНЯЕМ КОД
  { //сюда пишем любую функцию, например:
  PORTB |= 1 << 5 // ЗАПИСЫВАЕМ В PB5 ЕДИНИЦУ
  delay (1000ms) // ждем секунду
  PORTB &= ~(1 << 5)// ЗАПИСЫВАЕМ В PB5 ноль
  delay (1000ms) // ждем секунду
  }
}
```

Здесь мы проверяем равно ли выражение в скобках реальному состоянию портов, т.е. аналогично тому, если бы мы написали if (digitalRead(12) == 1).

Работа со временем в Arduino

Функции задержек

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

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

ИСПОЛЬЗОВАТЬ ЗАДЕРЖКИ НЕ РЕКОМЕНДУЕТСЯ! ОНИ ПОЛНОСТЬЮ "ВЕШАЮТ" СИСТЕМУ!

Функции таймера

millis() - возвращает количество миллисекунд, прошедших с момента включения МК. Макс. значение: 4’294’967’295 мс или 50 суток. Разрешение: 1 миллисекунда.

micros() - возвращает количество микросекунд, прошедших с момента включения МК. Макс. значение: 4’294’967’295 мкс или 70 мин Разрешение: 4 микросекунды

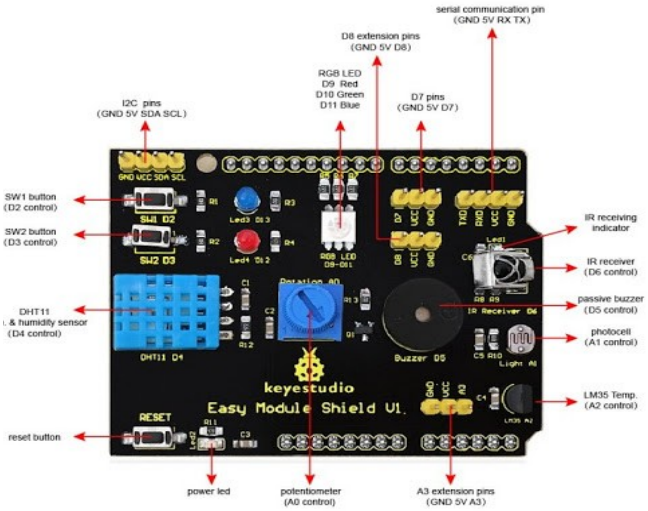
ВНИМАНИЕ: Пишем (long) перед умножением, чтобы программатор выделил нужное количество памяти для проведения операции! (для работы с большими числами).
Пример: (long)23*24*60*60*1000, если хотим получить ПРАВИЛЬНЫЙ результат умножения в виде числа миллисекунд, равного 23 дням.

На основе millis(), переменной unsigned long и условия можно сделать простой таймер, который будет срабатывать через указанные промежутки времени.

```
unsigned long last_time; // глобальная переменная!!!
void loop() {
if (millis() - last_time >= 5000) { // если прошло больше 5000 мс (5 секунд)
Serial.println(millis()); // код, который выполняется каждые 5 секунд
last_time = millis();      // сброс таймера
}
}
```

Приложения:

Макет



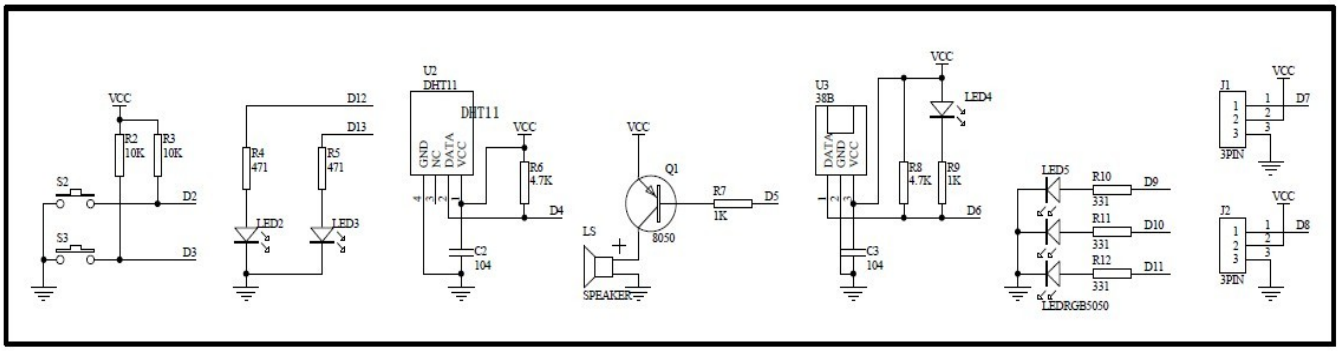
На плате установлены следующие датчики и устройства:

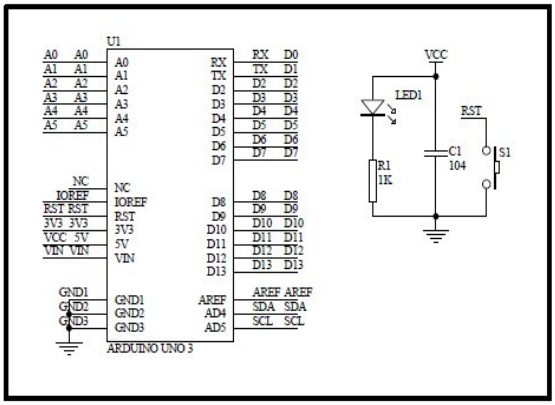
- цифровой датчик температуры и влажности DHT11 (подключен к D4);
- аналоговый датчик температуры LM35 (A2);
- аналоговый датчик освещенности (A1);
- приемник ИК сигналов с пульта (D6);
- динамик для генерации простейших звуковых сигналов (D5);
- две кнопки (D2, D3);
- потенциометр (A0);
- светодиод синий D13;
- светодиод красный D12;
- RGB светодиод (D9, D10, D11).

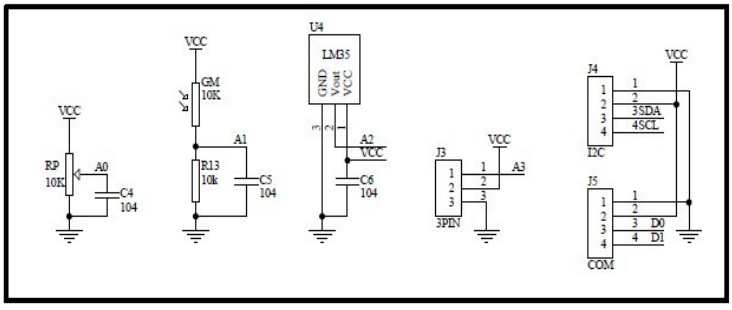
Есть возможность подключения еще одного аналогового датчика (например, датчика влажности почвы) и нескольких цифровых (например цифровой датчик температуры DS18B20).

Также можно подключить устройства с интерфейсом UART (GPS модуль или Bluetooth) и интерфейсом I2C (барометр или гироскоп).

Схема макета







Задания к лабораторной работе №2

1. Мигание светодиодом с выводом информации о состоянии его в консоль.
2. Подача звукового сигнала
3. Чтение состояния кнопки. Программное подавление «дребезга» контактов.
4. Зажигание светодиода d13 от кнопки
5. Управление цветом трехцветного светодиода с консоли
6. Управление через консоль. Должны поддерживаться команды
   help
   led on/off/flash — включение/выключение светодиода
   flash — мигание светодиодом длительностью delay
   beep — подача звукового сигнала длительностью delay
   delay — длительность задержки

Дополнительное задание

7. Ускорение операций работы с GPIO путем прямой работы с портом. Оценка увеличения быстродействия.