МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Воронежский государственный технический университет»

Кафедра радиоэлектронных устройств и систем

ОСНОВЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-АППАРАТНЫХ КОМПЛЕКСОВ И СИСТЕМ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторной работы №1
для студентов специальности 11.05.01
«Радиоэлектронные системы и комплексы»
очной формы обучения

Воронеж 2024

УДК 621.396.6.001.63(07)
ББК 32.844я7

Составитель А. И. Сукачев

Основы проектирования программно-аппаратных комплексов и систем: методические указания к выполнению лабораторной работы №1 для студентов специальности 11.05.01 «Радиоэлектронные системы и комплексы» очной формы обучения / ФГБОУ ВО «Воронежский государственный технический университет»; сост.: А. И. Сукачев. – Воронеж: Изд-во ВГТУ, 2024. – 32 с.

В соответствии с рабочими учебными программами дисциплин приведены описания методов измерений и методик выполнения лабораторных работ, изложены теоретические сведения, содержащие основы использования микропроцессорной техники. По каждой лабораторной работе в описание включены: теоретические материалы, используемое оборудование и приборы, порядок подготовки и проведения работы.

Предназначены для студентов специальности 11.05.01 «Радиоэлектронные системы и комплексы» очной формы обучения.

Методические указания подготовлены в электронном виде и содержатся в файле МУ_ОППАКиС_ЛР1.pdf.

Ил. 28. Табл. 1. Библиогр.: 6 назв.

УДК 621.396.6.001.63(07)
ББК 32.844я7

Рецензент – А. В. Останков, д-р техн. наук, профессор
кафедры радиотехники ВГТУ

Издается по решению редакционно-издательского совета
Воронежского государственного технического университета

ЛАБОРАТОРНАЯ РАБОТА №1
ОСНОВЫ РАЗРАБОТКИ УСТРОЙСТВ С ИСПОЛЬЗОВАНИЕМ МИКРОКОНТРОЛЛЕРА ATMEGA328P

Цель: изучение принципов разработки устройств с использованием микроконтроллера Atmega328p.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Ранее в рамках других дисциплин изучалось программирование, и писались программы под компьютер или смартфон, которые выполняли какие-то расчёты или обрабатывали и хранили данные, введённые пользователем. Однако, для автоматизации многих процессов необходимо производить измерения тех или иных величин (к примеру, температуры воздуха), в таком случае использование полноценного компьютера нецелесообразно.

Для небольших задач автоматизации в электронике используют микроконтроллеры – небольшие микросхемы, которые можно запрограммировать для выполнения не сильно сложных задач: от замены какой-то цифровой схемы на логических элементах, до целого небольшого сервера.

Микроконтроллеры являются сложными в применении устройствами, так как помимо проблем программирования (которое в них далеко не самое простое) накладываются и проблемы схемотехники. То есть необходимо правильно составить схему и написать программу.

Всё это отпугивало радиолюбителей и других интересующихся от темы микроконтроллеров, но благодаря **платформе Arduino** уровень знаний для работы с подобными устройствами снизился до *начальной школы*. Поэтому первая лабораторная работа будет посвящена работе с данной платформой, так как именно она становится в 90% случаев аспектом курсовой или дипломной работ.

Платформа Arduino включает в себя множество микроконтроллеров (и отладочных плат на основе них), но на первом занятии основное внимание будет уделено платам, основанных на микроконтроллерах семейства AVR, в частности ATMEGA328P, которая ставится в платы Arduino Uno и Nano (рис. 1).

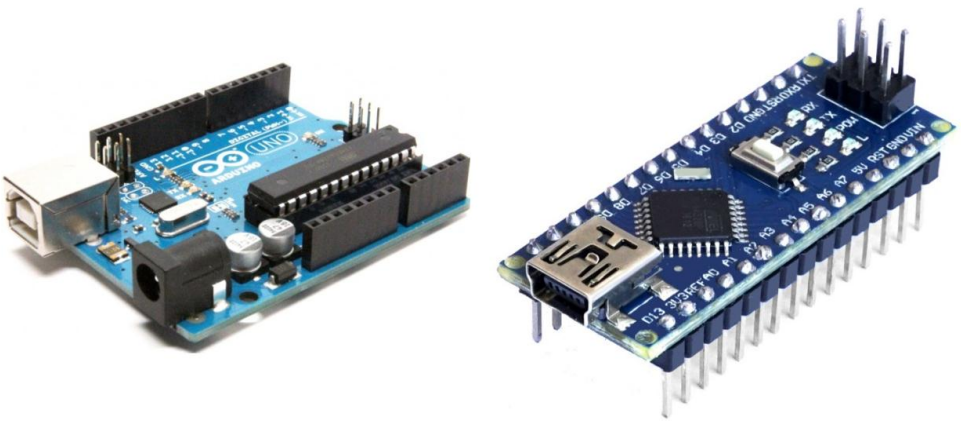

Рис. 1. Платы Arduino Uno (слева) и Arduino Nano (справа)

Обе платы основаны на микроконтроллере ATMEGA328P, характеристики которого сведены в табл. 1.

Таблица 1

Характеристики ATMEGA328P

Flash память	32 кБ
ОЗУ	2 кБ
EEPROM	1 кБ
Тактовая частота	16 МГц
Разрядность ядра	8 бит
Интерфейсы	I2C, UART, SPI
Напряжение питания	От 3 до 5В
Логические уровни	Зависят от напряжения питания

Как видно, объем памяти у контроллера небольшой, и, следовательно, ее следует использовать экономно. И так, Flash память служит для хранения программы, то есть итогового машинного кода, а также констант, если указать их для записи во Flash. ОЗУ – оперативная память служит для хранение всех переменные. EEPROM – энергонезависимая память, куда могут быть записаны различные настройки, или другая информация. Число перезаписей данного типа памяти ограничено, следовательно при частой перезаписи, хранимой на ней информации, она быстро придет в негодность.

Микроконтроллеры AVR на текущий момент довольно сильно устарели, но благодаря платформе Arduino, до сих пор пользуются популярностью. Поэтому прямо сейчас и рассмотрим базовый функционал данной платформы, на основе Arduino Nano или Arduino Uno.

2. ПОДГОТОВКА ПРОГРАММ И ДРАЙВЕРОВ

Ниже приведена пошаговая инструкция под ОС Windows:

1. Пойти на сайт производителя arduino.cc и скачать необходимую версию Arduino IDE. На странице для скачивания нажать кнопку «Just Download» (рис. 2).

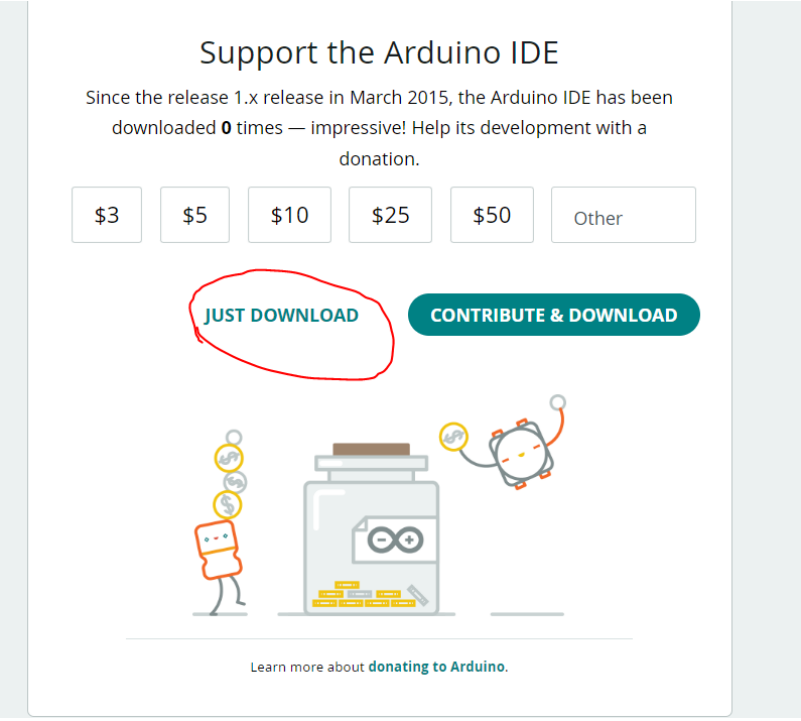

Рис. 2. Скачивание Arduino IDE

2. Запустить установщик и следовать его инструкциям (помимо самой Arduino IDE он также устанавливает и драйвер для оригинальных плат Arduino).
3. Так как платы Arduino у нас в большинстве случаев китайские и далеко не оригинальные, то для них необходимо скачать собственный драйвер, а какой именно зависит от платы. Дело в том, что сам микроконтроллер не имеет USB интерфейса, поэтому для программирования платы через USB используется микросхема-преобразователь USB-UART, которую китайские производители устанавливают какую попало. Т.о. для того, чтобы поставить нужный драйвер нужно определить название микросхемы, к примеру, у Arduino Nano USB-UART преобразователь находится снизу (рис. 3).

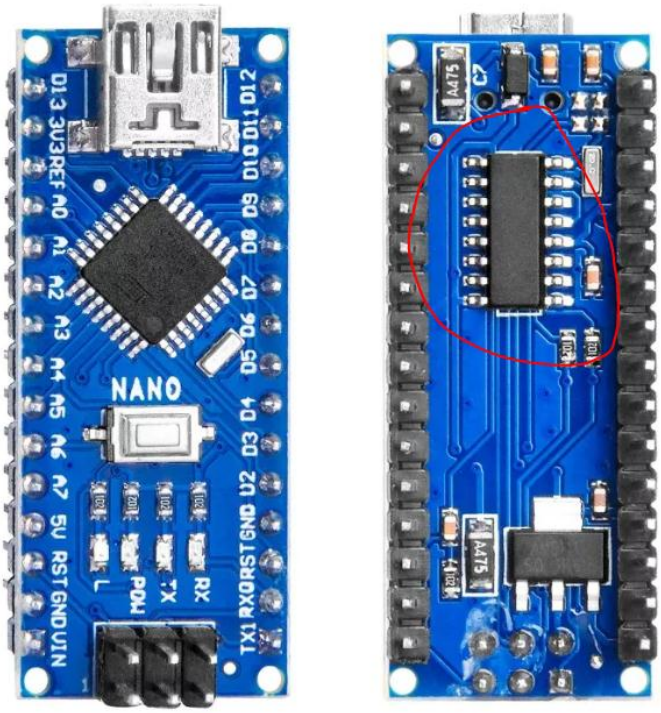

Рис. 3. Расположение преобразователя USB-UART

4. Узнав название микросхемы, пишем её в интернет со словом «…. driver». И ищем нужную версию. Обычно используются микросхемы CH340 или CH341, но в частности в Arduino Nano, что есть в лаборатории, стоит FT232.
5. Скачиваем драйвер и устанавливаем согласно инструкции. Драйвер для CH340 идёт как .exe файл и требует подключенной Arduino в компьютер для установки. Драйвер для FT232 идёт набором файлов (рис. 4).

Name	Size	Packed	Type	Modified	CRC32
..			Folder		
amd64			Folder	01.02.2012 15:29	
i386			Folder	01.02.2012 15:29	
Static			Folder	01.02.2012 15:29	
ftd2xx.h	27 161	6 351	C++ Header file	26.01.2011 16:29	22F3997F
ftdibus.cat	12 114	5 448	Security Catalog	12.04.2011 12:56	34A7105A
ftdibus.inf	5 477	1 809	Setup Information	18.03.2011 13:11	8F6E178A
ftdiport.cat	11 210	5 314	Security Catalog	12.04.2011 12:56	600742B0
ftdiport.inf	5 382	1 907	Setup Information	18.03.2011 13:11	03E3B2A3
LogoVerificationReport.pdf	42 808	34 377	Adobe Acrobat ...	12.04.2011 15:32	BD6EB220

Рис. 4. Драйвер для FT232

6. Открываем диспетчер устройств, подключаем плату к компьютеру и ищем неизвестное USB устройство. Когда таковое будет найдено необходимо нажать по нему ПКМ-Обновить драйверы-Поиск драйверов на этом компьютере, в качестве папки поиска выбираем папку, куда скачивали драйвер.

7. Обновляем диспетчер устройств, если вместо неизвестного USB устройства появился COM порт – то установка прошла успешно, в противном случае ищем другой драйвер.

В случае использования Linux шаги остаются теми же, но со своими особенностями. В частности, драйвер CH340/CH341 идёт в комплекте многих дистрибутивов Linux.

И так, теперь можно запустить Arduino IDE (рис. 5).

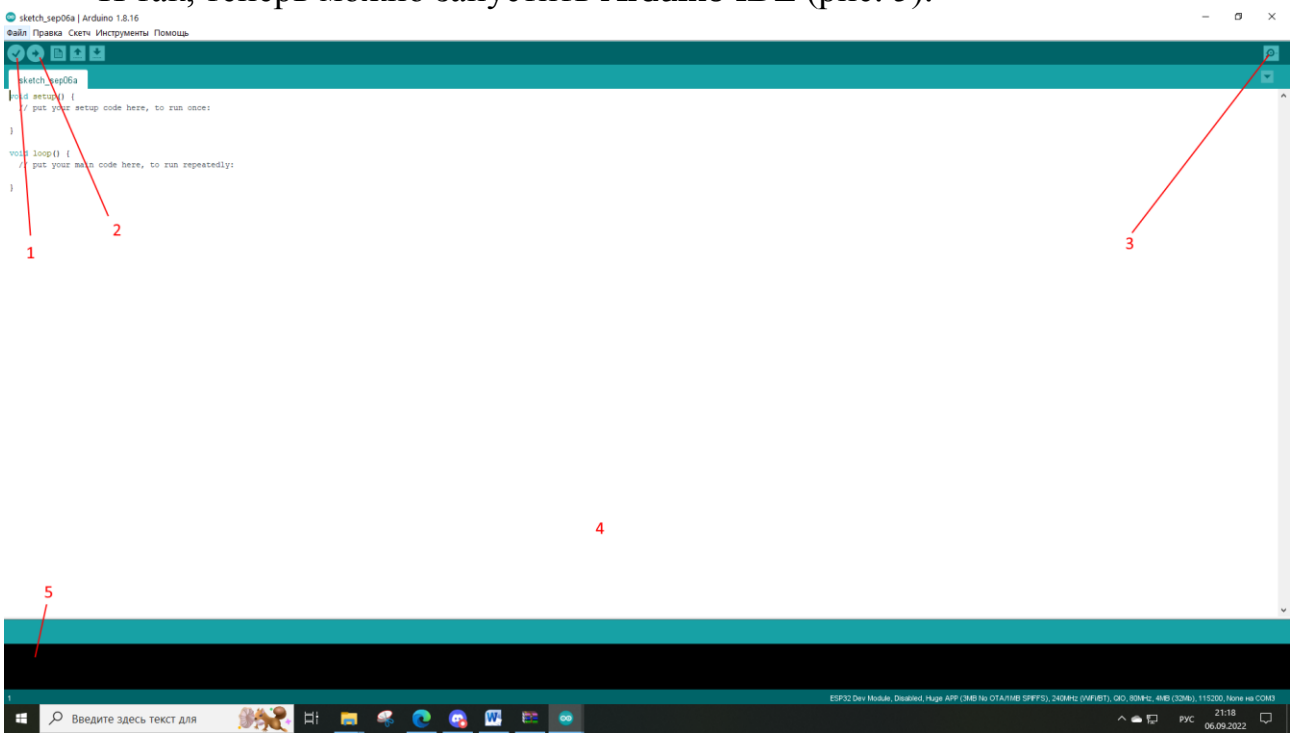

Рис. 5. Интерфейс Arduino IDE

Цифрами на рис. 5 помечены: 1 – Кнопка проверки программы, 2 – Кнопка загрузки программы в микроконтроллер, 3 – Терминал последовательного порта, 4 – Редактор кода, 5 – Терминал компилятора и загрузчика.

Первым этапом разберёмся со структурой программы. Дело в том, что несмотря на использование C++, как языка программирования. Синтаксис немного отличается, здесь нет привычной **int main()**, так как в платформе Arduino она «спрятана» и заменена на **void setup()** и **void loop()**. И так, **void setup() –** выполняется один раз при запуске микроконтроллера, **void loop()** - цикл, в котором происходит работа основного кода.

Такое изменение связано с упрощением кода для неопытных пользователей, в AVR C, на котором, в конечном счете, и пишутся программы под AVR, есть **int main()**, внутри которого дальше есть бесконечный цикл while. Именно это и скрыли от нас разработчики платформы. Вообще, именно из-за сильного упрощения ценой потери производительности опытные разработчики и не любят данную платформу.

На плате Arduino Nano установлен светодиод на выводе 13 (рис. 6). Целью первой программы примем периодическое его включение с интервалом в 1 секунду.

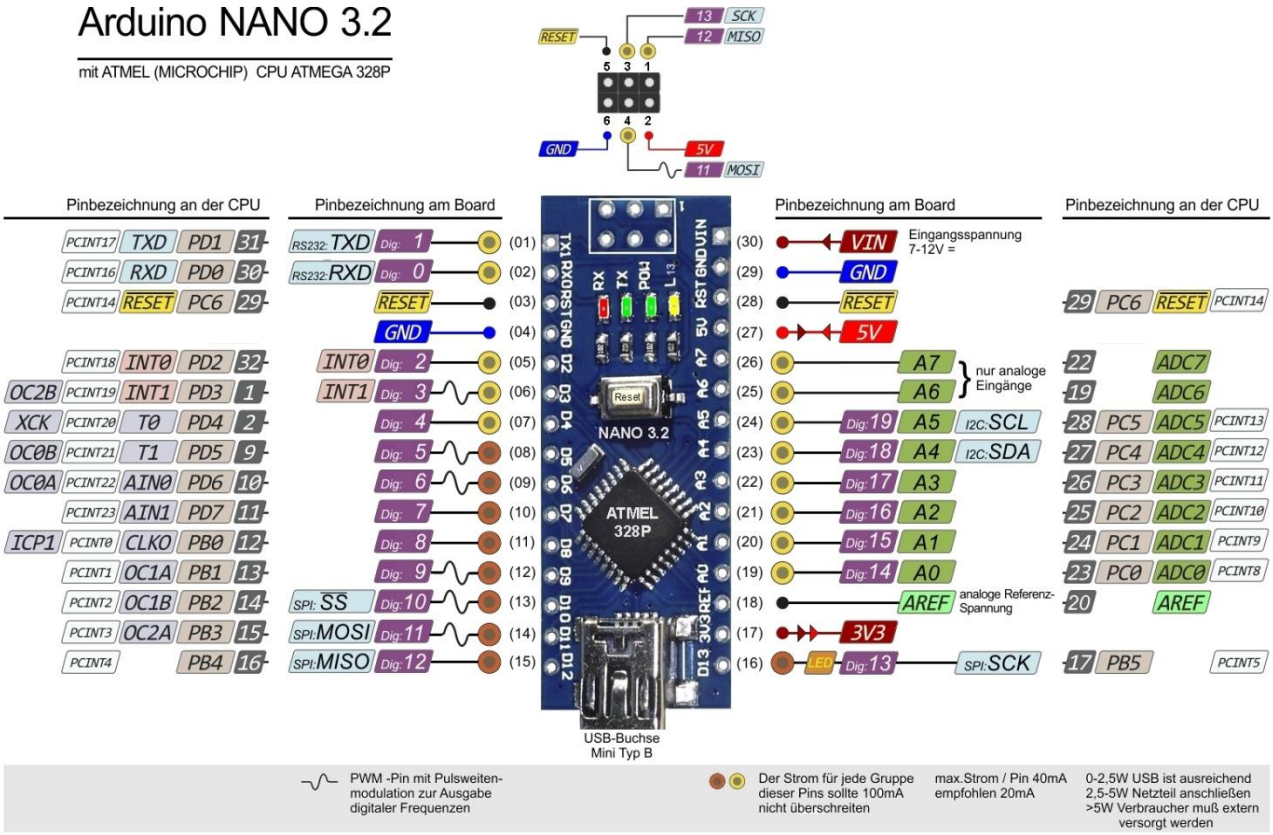

Рис. 6. Выводы платы Arduino Nano

Для этого напишем простую программу:

```
void setup() {
  // put your setup code here, to run once:
pinMode(13, OUTPUT);//Определяем вывод 13, как выход
}

void loop() {
  // put your main code here, to run repeatedly:
digitalWrite(13,HIGH);//Ставим высокий лог. уровень на выводе 13
delay(1000);//Ждём 1000мс или 1с
digitalWrite(13,LOW);//Ставим низкий лог. уровень на выводе 13
delay(1000);//Ждём 1000мс или 1с
}
```

Программа в микроконтроллере выполняется последовательно, сверху вниз, delay() – это задержка выполнения всех команд в контроллере на заданное время, поэтому оно крайне не рекомендуется к использованию. Это первая причина низкой скорости работы кода, но в рамках примера рассмотрим его.

Далее данную программу нужно загрузить в микроконтроллер, для этого во вкладке «Инструменты – Порт», выбираем порт, на котором находится плата Arduino. Также в качестве платы, указываем, используемую плату (в примере Arduino Uno), на практике в лаборатории лежит Arduiono Nano. Также в некоторых, случаях для Arduino Nano нужно выбрать режим «Old Bootloader», в частности имеющаяся в лаборатории Nano, работает только в таком режиме (рис. 7).

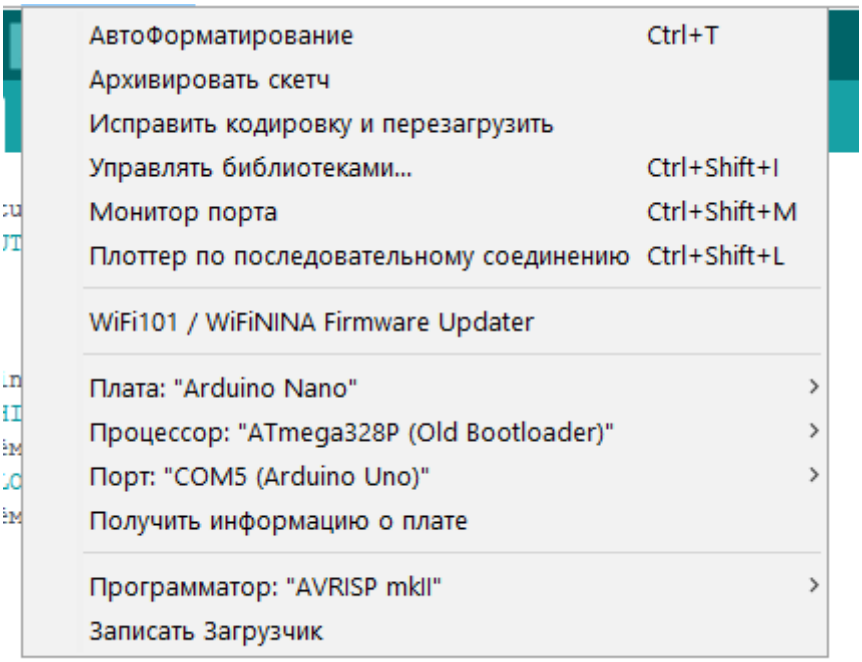

Рис. 7. Настройка параметров загрузки программы

Bootloader – он же загрузчик, специальная подпрограмма, уже записанная в микроконтроллер, отвечает за загрузку программы по UART, по умолчанию для программирования «чистого» контроллера требуется специальный программатор. Она занимает некоторое место, и поэтому на платах Arduino не доступна вся Flash память, часть её занята загрузчиком.

После настройки параметров загрузки желательно, зайти в настройки (Файл – Настройки) и выставить параметры как на рис. 8.

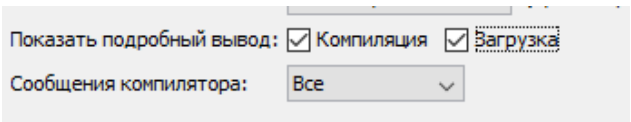

Рис. 8. Подробный вывод компилятора и загрузчика

Данный пункт позволит быстрее понять причину ошибки.

Далее нужно нажать кнопку «Загрузка», и если плата подключена к компьютеру, и всё настроено правильно, то программа загрузиться, светодиоды «RX», «TX» на плате Arduino замигают и светодиод «L», начнёт мигать раз в секунду.

Язык Arduino достаточно прост, но как было сказано ранее, не эффективен, поэтому для сравнения приведём программу на AVR C, под микроконтроллер ATMEGA328P:

```
int main()
{
 DDRB |= (1<<PB5); //Ставим вывод PB5 как выход
 while(true)
 {
  PORTB |= (1<<PB5); //Включаем светодиод
  _delay_ms(1000); //Ждём 1000мс
  PORTB &= (0<<PB5);//Выключаем светодиод
  _delay_ms(1000);//Ждём 1000мс
 }
}
```

И код стало сложнее читать, для проверки его работы, его можно загрузить вместо предыдущего и посмотреть на то, что он прекрасно работает (только под ATMEGA328).

В случае возникновения проблем и ошибок необходимо обратиться в интернет.

Писать на AVR C в данных лабораторных работах не нужно, но при желании, можете попробовать.

3. БАЗОВЫЙ ФУНКЦИОНАЛ ПЛАТФОРМЫ ARDUINO

Добавим в существующую программу мигания светодиодом вывод сообщений в COM порт. Как было сказано ранее, USB на данной плате реализован через USB-UART преобразователь, который создаёт виртуальный COM порт, то есть на уровне микроконтроллера мы работаем с UART.

UART – Universal Asynchronous Receiver Transmitter – асинхронный протокол передачи данных по двум переводам (Приём/Передача) (рис. 9).

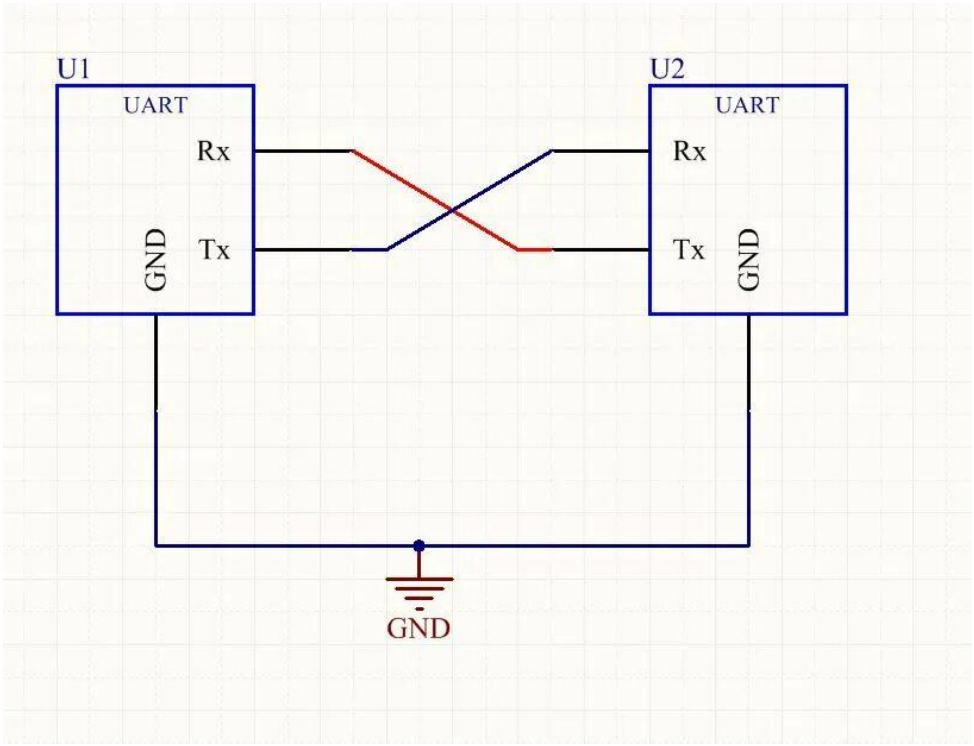

Рис. 9. Принцип работы UART

В данном случае по нему соединены микроконтроллер и микросхема-преобразователь USB-UART.

Для работы с UART в Arduino есть библиотека Serial, которая позволяет отправлять и принимать информацию по данному протоколу. Для вывода информации о состоянии светодиода напишем следующий код:

```
void setup() {
  // put your setup code here, to run once:
pinMode(13, OUTPUT);//Определяем вывод 13, как выход
Serial.begin(9600);//Запустим UART на скорости 9600 бод
}

void loop() {
  // put your main code here, to run repeatedly:
digitalWrite(13,HIGH);//Ставим высокий лог. уровень на выводе 13
Serial.println("On"); //Выведем сообщение
delay(1000);//Ждём 1000мс или 1с
digitalWrite(13,LOW);//Ставим низкий лог. уровень на выводе 13
Serial.println("Off"); //Выведем сообщение
delay(1000);//Ждём 1000мс или 1с
}
```

На рис. 10 показан результат работы программы в терминале COM порта.

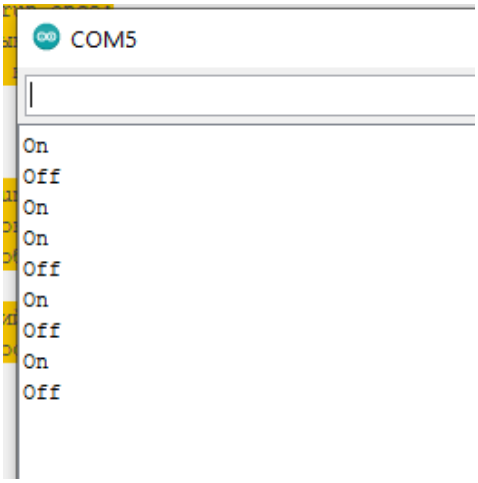

Рис. 10. Результат работы программы

При неправильно выбранной скорости порта в настройках терминала текст будет распознаваться неверно (рис. 11).

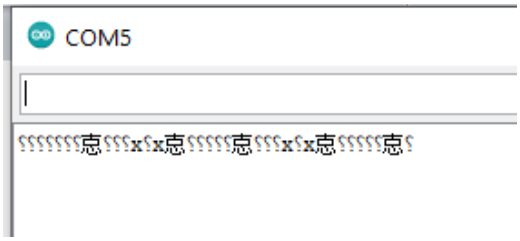

Рис. 11. Неверная скорость порта в терминале

Из этого следует, что для нормальной работы приёмник и передатчик должны работать на одной и той же скорости.

Далее рассмотрим АЦП. АЦП – аналогово-цифровой преобразователь, который служит для измерения напряжения и других величин, выраженных через напряжение. В Arduino Nano (рис. 6) есть 8 аналоговых входов. Сам модуль АЦП 10-битный, следовательно, измеренное значение варьируется от 0 до 1023 включительно. В качестве примера работы АЦП, подключим потенциометр (переменный резистор) по схеме (рис. 12).

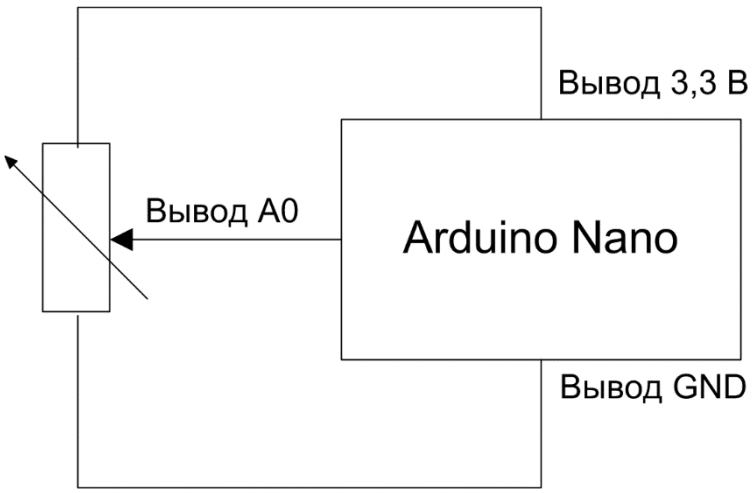

Рис. 12. Схема подключения потенциометра

Теперь напишем программу:

```
void setup() {
  // put your setup code here, to run once:
pinMode(A0, INPUT);//Определяем вывод 13, как выход
Serial.begin(9600);//Запустим UART на скорости 9600 бод
}

void loop() {
  // put your main code here, to run repeatedly:
Serial.println(analogRead(A0)); //Считаем и выведем значение АЦП
// на выводе A0
}
```

После открытия терминала мы увидим огромный поток значений, который при повороте ручки резистора будет изменяться.

Так как АЦП измеряет значения относительно опорного источника, то диапазон измерений можно варьировать. К примеру, по умолчанию, используется напряжение питание микроконтроллера (3,3В на Arduino Nano) в качестве этого напряжения. Но его можно изменить на внутренний источник (1,1В для Arduino Nano и Uno) или на внешний (вывод AREF). В Arduino за выбор источника опорного напряжения служит команда: **analogReference**.

Дополнительная информация: В данном случае можно включить плоттер по последовательному порту: Инструменты – Плоттер по последовательному соединению. Таким образом, изменение значений можно увидеть графически.

4. ШИМ

Это довольно важная часть данной работы, так как помимо самой Arduino будет задействован осциллограф. Для того, чтобы увидеть результат работы программы, нужно воспользоваться осциллографом. В лаборатории имеется USB осциллограф Hantek. При необходимости необходимо установить ПО с диска, а также скопировать папку Drivers, нажать кнопку на осциллографе в положение "ВКЛ", и подключить прибор к ПК. Он отобразиться в диспетчере устройств, как неизвестное USB устройство. Обновляем драйвер, на тот, что идёт в комплекте на диске.

Далее запускаем программу Hantek 6022BE. И видим окно осциллографа. Для работы с осциллографом необходимо подключить в один из каналов щуп и для проверки работоспособности прикоснуться им, к выводу встроенного генератора. Если на экране появился меандр – то всё хорошо. В случае возникновения проблем – обращаться к преподавателю.

И так, ШИМ – широтно-импульсная модуляция (рис. 13) представляет последовательность прямоугольных импульсов с разной скважностью, которая зависит от некоторых входных данных. Используется для управления яркостью ламп, скоростью вентилятора и т.д. Также применятся в импульсных источниках питания.

В Arduino для формирования ШИМ используется команда **analogWrite().**

С её помощью можно сформировать ШИМ сигнал на одном из специальных выводов, на рис. 6 они помечены знаком «~». В качестве примера, сформируем ШИМ с коэффициентом заполнения 30% на выводе 9. Для этого нужно вспомнить математику: ШИМ в микроконтроллере 8-битный, т.е. напряжение имеет уровни на выходе от 0 до 255. 0 и 255 – это минимальное и максимальное значения, которые соответствуют положениям – выключено и включено полностью. Для режима 30% - нужно найти 30% от 256, что составляет приблизительно 85. ШИМ модуляция формируется за счёт таймера, встроенного в микроконтроллер (всё равно не понятно, углубляться не будем), это означает то, что он будет генерироваться независимо от выполнения кода программы.

Пишем программу:

```
void setup() {
  // put your setup code here, to run once:
pinMode(9, OUTPUT);//Определяем вывод 9, как выход
analogWrite(9, 85); //Формируем ШИМ на выводе 9
}
void loop() {
  // put your main code here, to run repeatedly:
}
```

Результат выполнения программы показан на рисунке 13.

Рис. 13. ШИМ сигнал

Под конец данного раздела объединим все, изученное ранее: сделаем установку значения ШИМ из терминала COM:

```
uint8_t PWM = 85; //Экономим память и используем uint8_t вместо int
void setup() {
  // put your setup code here, to run once:
pinMode(9, OUTPUT);
Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
if(Serial.available()>1)//Если что-то пришло из терминала
{
  PWM = Serial.parseInt();//Парсим целое число (тип int)
  analogWrite(9,PWM);//Применяем изменения
}
}
```

Если программа работает верно, то при вводе числа в терминал, коэффициент заполнения ШИМ будет изменяться.

4.1. РАЗГОН ШИМ

Для данного действия потребуется базовое понимание происходящего внутри микроконтроллера: таймеры. Таймер – это счётчик, который считает от 0 до некоторого числа, а потом сбрасывается. Если правильно настроить таймер, то его можно заставить генерировать ШИМ сигнал (рис. 14).

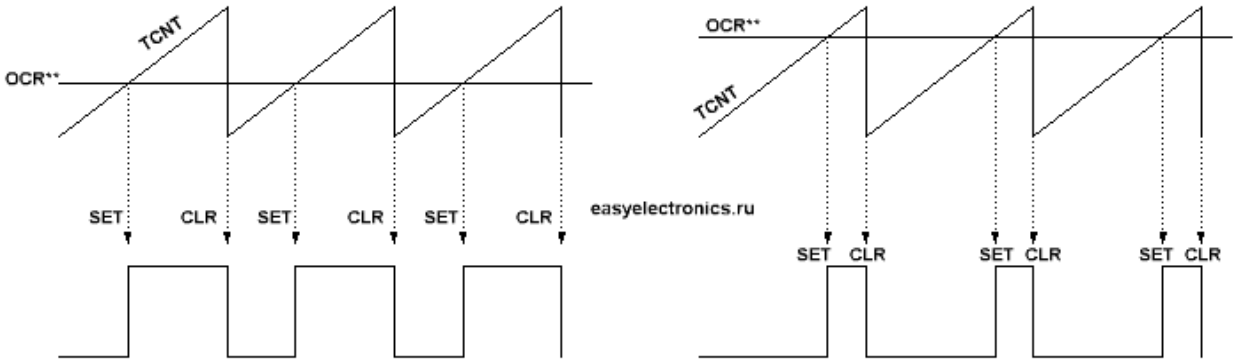

Рис. 14. Генерирование ШИМ через таймеры

Причём **analogWrite** делает эту настройку за нас, но делает это с фиксированной частотой 490 Гц. Что подходит для большинства задач, но иногда требуется большая частота. Поэтому приходится прибегать к разгону ШИМ. Для этого придётся обратиться к документации на ATMEGA328P и выяснить из неё про режим быстрого ШИМ (Fast PWM), который можно настроить через следующие регистры (рис. 15).

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 15. Регистр TCCR2A для настройки таймера 2

Далее, согласно документации, нужно выставить «1» в нужные позиции:

```
        void setup() {
           DDRB = 1 << (PB3);
           TCCR2A = 1 <<(COM2A0) | 1 <<(WGM20) | 1 << (WGM21);
//Включаем выход шим на OC2A (D11 на Arduino Nano) в режиме Fast PWM
           TCCR2B = 1 << (CS20) | 1 <<(WGM22); // Устанавливаем режим Fast
PWM и ставим пределитель равный 1.
           OCR2A = 20; //Считаем до 20
           OCR2B = 10; //Выключаем шим после 10
           //Оба регистра 8 битные
           //Таким образом, имеем скважность равную 2
           //Итоговая частота ШИМ: F = 8MHz/20 = 400kHz

        }
```

Работа программы показана на рис. 16.

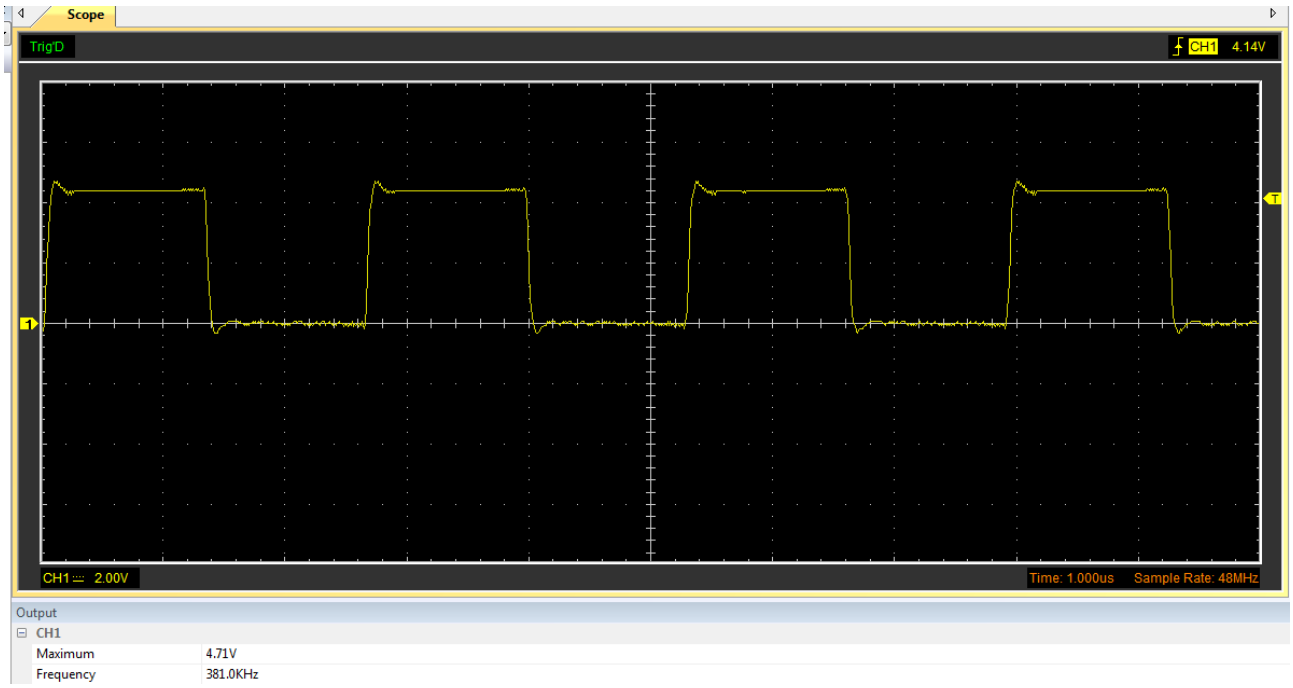

Рис. 16. Разогнанный ШИМ

Вообще, если использовать нулевой таймер, то можно получить большую точность и плавность регулировки, но тогда это помешает правильной работе функции millis().

4.2. ИСПОЛЬЗОВАНИЕ ШИМ ДЛЯ УПРАВЛЕНИЯ НАГРУЗКОЙ

После того как мы познакомились с возможностями ШИМ на Arduino Nano, пора его применить для управления электродвигателем.

Напрямую управлять мотором нельзя, так как максимально допустимая сила тока на одном цифровом выходе ATMEGA328P составляет 20 мА, в противном случае микроконтроллер может **сгореть**.

Для управления мощной нагрузкой необходимо построить усилитель сигнала, так как линейность (сохранение исходной формы сигнала) в данном случае не важна, то для данных целей можно воспользоваться транзисторным ключом. Его можно построить либо на биполярном транзисторе (рис. 17), либо на полевом с изолированным затвором (рис. 18).

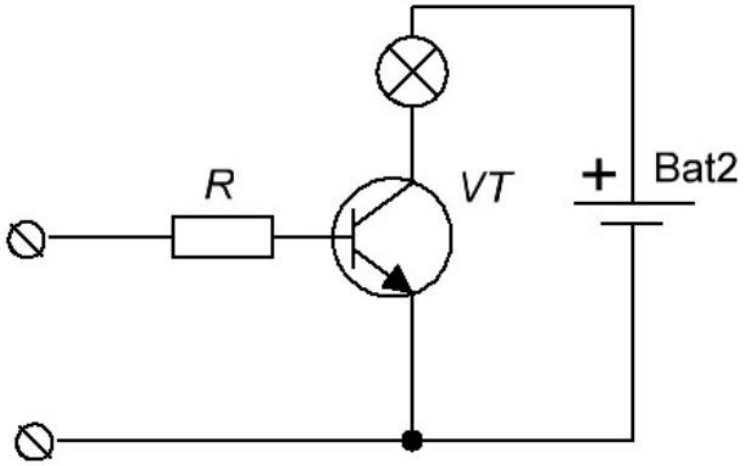

Рис. 17. Транзисторный ключ на биполярном транзисторе

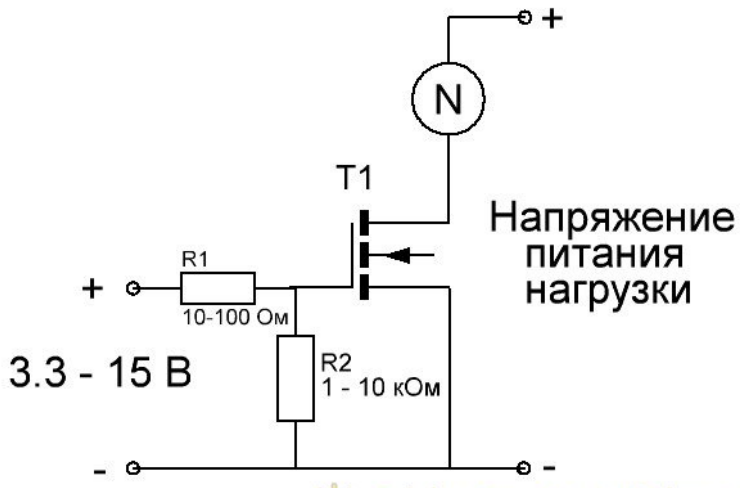

Рис. 18. Транзисторный ключ на полевом транзисторе

Ключ на полевом транзисторе обладает следующими основными преимуществами: низкие потери в открытом состоянии, управляются напряжением. Но при этом есть самый существенный недостаток: затвор обладает высокой ёмкостью (от 1 нФ до 20 нФ), следовательно, на больших частотах им довольно сложно управлять (требуется большой ток для зарядки этой ёмкости), соответственно требуется предварительный усилитель.

Ключ на биполярном транзисторе обладает более низкой эффективностью, так как требуется подавать ток на базу, а также сам транзистор имеет потери в районе 1 В на переходе коллектор-эмиттер. С другой стороны частота переключения при управлении малым током получается выше.

Теперь нужно собрать схему с вентилятором(будет осуществляться управление его электродвигателем) и взять код из предыдущего этапа:

```
uint8_t PWM = 85; //Экономим память и используем uint8_t вместо int
void setup() {
  // put your setup code here, to run once:
pinMode(9, OUTPUT);
Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
if(Serial.available()>1)//Если что-то пришло из терминала
{
  PWM = Serial.parseInt();//Парсим целое число (тип int)
  analogWrite(9,PWM);//Применяем изменения
}
}
```

Схема представлена на рис. 19.

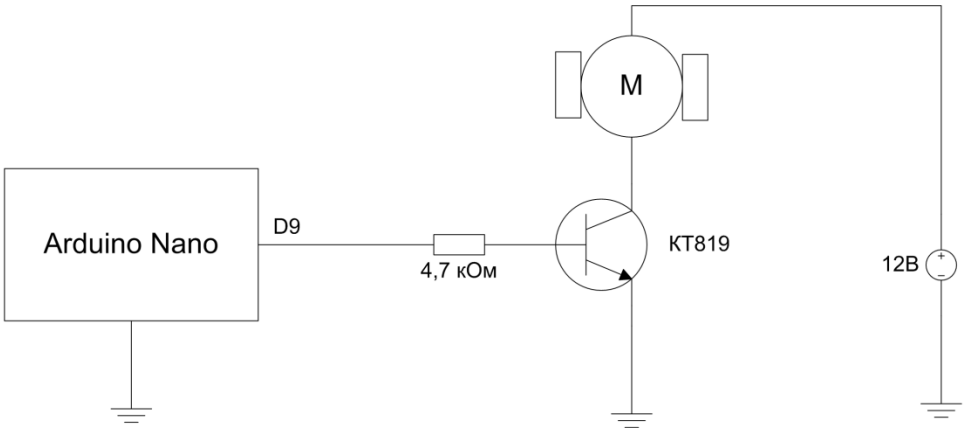

Рис. 19. Подключение электродвигателя к Arduino

В след за изменением скважности сигнала будет меняться скорость вращения двигателя вентилятора. Из-за низкой частоты ШИМ во время работы будет слышен писк. Также нужно понимать, что при работе с индуктивной нагрузкой нужно применять защитные диоды для транзистора, иначе высоковольтные выбросы (рис. 20) могут его повредить.

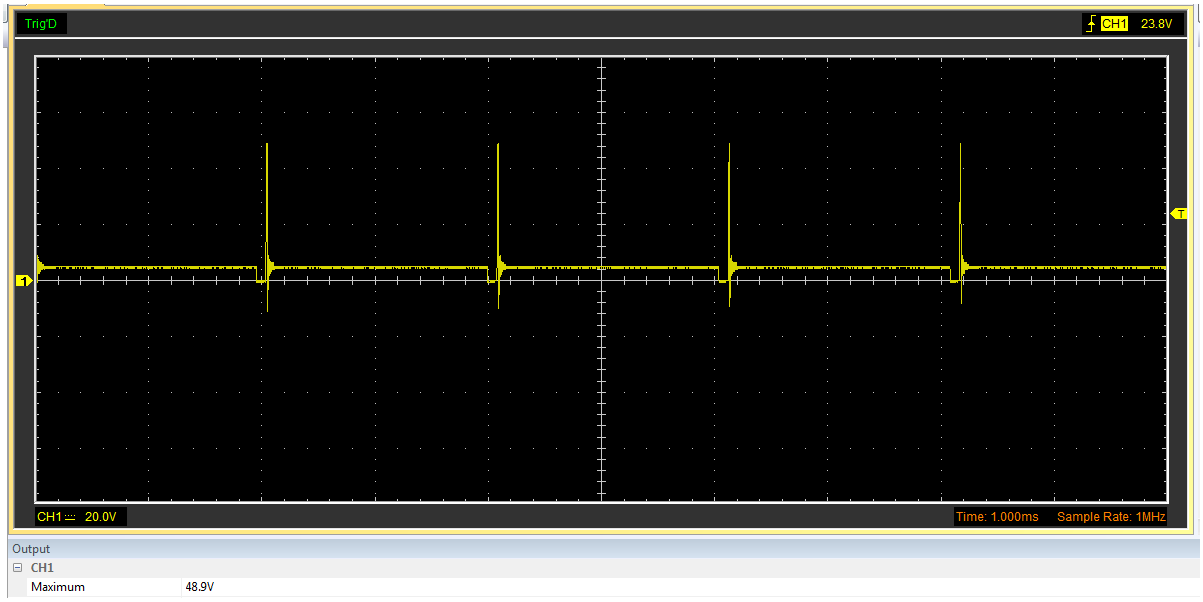

Рис. 20. Выбросы при коммутации индуктивной нагрузки

5. СВЯЗЬ ARDUINO С QT ЧЕРЕЗ COM ПОРТ

В рамках прошлого этапа мы научились регулировать и управлять Arduino Nano через встроенный COM терминал. Теперь научимся отправлять данные в COM порт посредством приложения, написанного с помощью Qt. Для этого нужно создать проект Qt Widgets, все настройки оставляем по умолчанию (рис. 21).

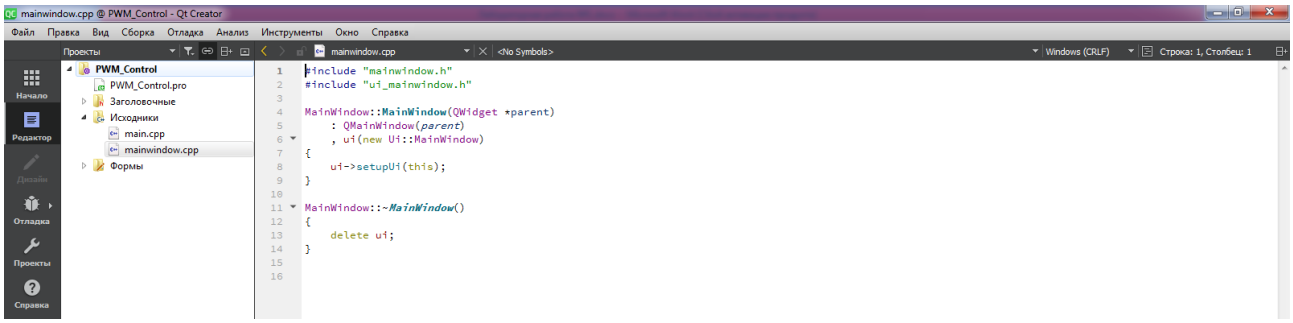

Рис. 21. Созданный шаблон приложения Qt Widgets

В качестве интерфейса создадим простой регулятор и индикатор (рис. 22).

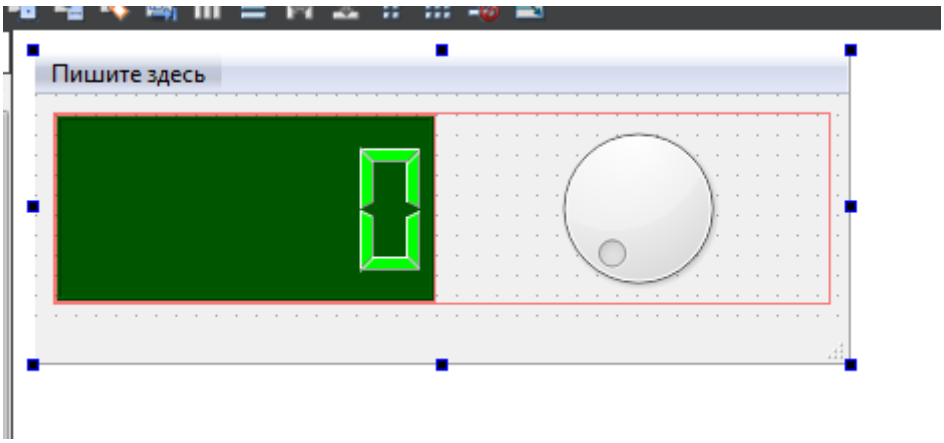

Рис. 22. Интерфейс программы

В качестве пределов для ручки нужно установить 0 и 255, в случае выхода за эти пределы – произойдёт переполнение переменной, что приведёт к непредсказуемым последствиям.

Далее нужно указать в .pro файле использование модуля SerialPort:

```
QT       += core gui serialport
```

Теперь в программе доступна библиотека QSerialPort, через которую и будет произведено взаимодействие.

Далее нужно создать класс для работы с командами COM порта (рис. 23).

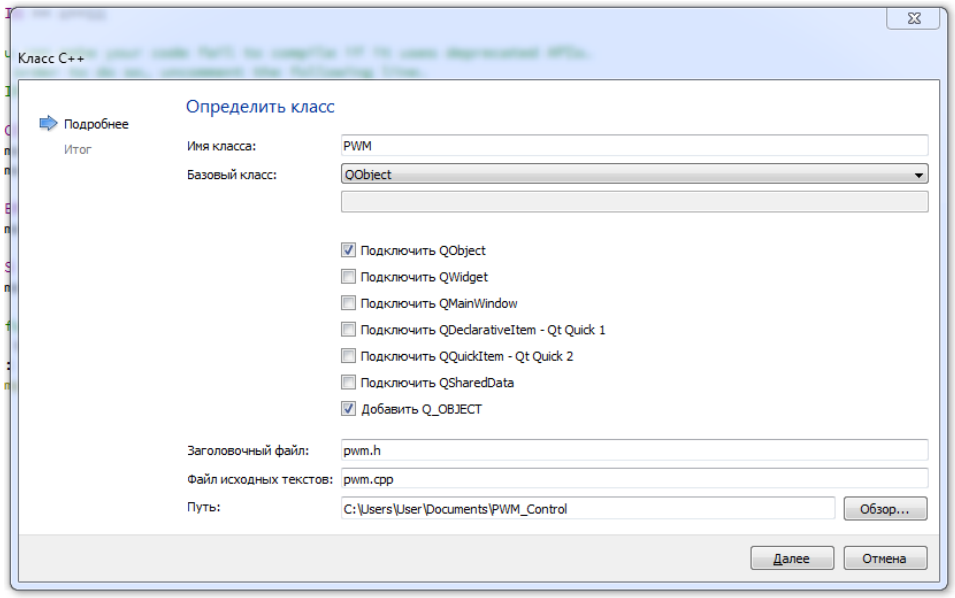

Рис. 23. Класс для работы с COM портом

В заголовочном файле нужно объявить 2 функции:

```
void setPWM(uint8_t);
bool SerialPortInit(QString PortName);
```

Далее нужно заполнить обе функции в файле pwm.cpp:

```
bool PWM::SerialPortInit(QString PortName)
{
   Port->setPortName(PortName);
   Port->close(); //Закрываем, если он открыт
   bool fail = false;
   Port->setBaudRate(QSerialPort::Baud9600); //Скорость, та что и в ардуине
   Port->setDataBits(QSerialPort::Data8);//Дальше по стандарту
   Port->setStopBits(QSerialPort::OneStop);
   Port->setParity(QSerialPort::NoParity);
   Port->setFlowControl(QSerialPort::NoFlowControl);
   if( Port->open(QSerialPort::ReadWrite))//Открываем на чтение и запись
   {
      fail = true;//Если получилось, то возвращаем true
      //Тут сделаем заготовку для connect, чтобы подключать чтение из порта
   }

   return (fail);
}
```

```
void PWM::setPWM(uint8_t pwm)
{
   QByteArray c;
   c.append(pwm);
   qDebug()<<c;
   Port->write(c);
   c.clear();
}
```

Полный листинг программы представлен ниже:

Файл pwm.h:

```
#ifndef PWM_H
#define PWM_H

#include <QObject>
#include <QSerialPort>
class PWM : public QObject
```

```
        {
           Q_OBJECT
        private:
           QSerialPort* Port;
        public:
           explicit PWM(QObject *parent = nullptr);
            bool SerialPortInit(QString PortName);
            void setPWM(uint8_t);
        signals:

        };

        #endif // PWM_H
```

Файл pwm.cpp:

```
        #include "pwm.h"
        #include <QtDebug>
        PWM::PWM(QObject *parent) : QObject(parent)
        {
           Port = new QSerialPort(this);
           Port->setReadBufferSize(64);
        }
        bool PWM::SerialPortInit(QString PortName)
        {
           Port->setPortName(PortName);
           Port->close(); //Закрываем, если он открыт
           bool fail = false;
           Port->setBaudRate(QSerialPort::Baud9600); //Скорость, та что и в
ардуине
           Port->setDataBits(QSerialPort::Data8);//Дальше по стандарту
           Port->setStopBits(QSerialPort::OneStop);
           Port->setParity(QSerialPort::NoParity);
           Port->setFlowControl(QSerialPort::NoFlowControl);
           if( Port->open(QSerialPort::ReadWrite))//Открываем на чтение и
запись
           {
              fail = true;//Если получилось, то возвращаем true
              //Тут сделаем заготовку для connect, чтобы подключать чтение из
порта
           }

           return (fail);
```

```
}
void PWM::setPWM(uint8_t pwm)
{
    QByteArray c;
    c.append(pwm);
    qDebug()<<c;
    Port->write(c);
    c.clear();
}
```

Файл mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "pwm.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    PWM* pwm;
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_dial_valueChanged(int value);

private:
    Ui::MainWindow *ui;

};
#endif // MAINWINDOW_H
```

Файл mainwindow.cpp:

```
#include "mainwindow.h"
```

```
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    pwm = new PWM();
    pwm->SerialPortInit("COM8");
}

MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::on_dial_valueChanged(int value)
{
    ui->lcdNumber->display(value);
    pwm->setPWM((uint8_t)value);
}
```

Результат работы программы показан на рис. 24.

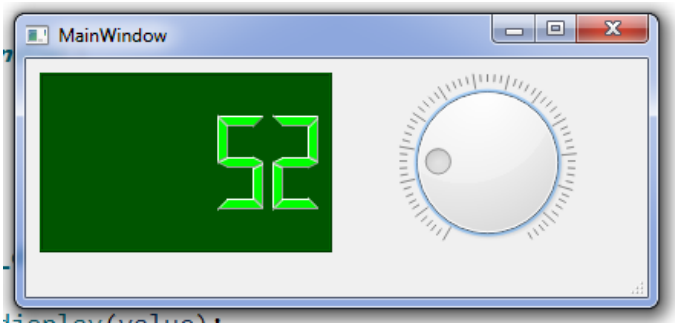

Рис. 24. Результат работы программы

Конечно, данный «протокол» передачи данных далеко не лучший, но для понимания работы его вполне достаточно. Для приёма данных нужно использовать слот **readyRead(),** в котором нужно вызывать обработчик сообщения. В частности, для примера, примем сообщение от Arduino и выведем в отладку. Для этого допишем тот самый connect, о котором шла речь в самом начале данного пункта:

```
        {
            fail = true;//Если получилось, то возвращаем true
             connect(Port,&QSerialPort::readyRead,this,&PWM::slotReadData);
        }
```

Слот, конечно же, необходимо объявить в .h файле и реализовать в .cpp. В самом же слоте просто будем выводить всё в отладчик:

```
 void PWM::slotReadData()
{
   qDebug()<<Port->readAll();
}
```

ЗАДАНИЕ. Написать программу для Ardunio, которая будет выводить в COM порт сообщение с номером **вашей** группы.

В результате работы программы вы должны получить в терминале следующее сообщение (только со своим номером группы) (рис. 25).

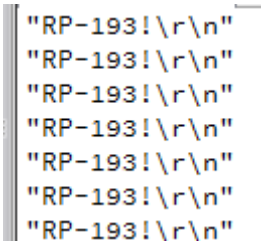

Рис. 25. Работа программы

6. ПРЕРЫВАНИЯ

После небольшого ознакомления со связью Qt и Arduino (она пригодится дальше) следует ознакомиться с асинхронной работой микроконтроллера. Конечно, можно многое сделать через if else, но иногда нужно очень быстро и точно отреагировать на то или иное воздействие. В частности, для измерения расстояния с помощью УЗ дальномера (рис. 26).

Рис. 26. Внешний вид УЗ дальномера

Также прерывания используются для пробуждения микроконтроллера из режима сна и т.д. Ещё прерывания можно повесить на таймер, что позволит повторять некоторое действие с заданной частотой (к примеру, опрашивать датчик температуры).

Данный механизм несколько похож на сигнально-слотовые соединения в Qt. Поэтому, поняв одно из этого, понять другое достаточно просто.

В качестве примера сделаем УЗ дальномер очень нерациональным, но очень наглядным методом, с использованием прерываний. УЗ дальномер выдаёт прямоугольный импульс пропорциональный дальности объекта (рис. 27). Наша задача максимально точно определить ширину данного импульса и перевести в расстояние. Конечно, так как частота небольшая, то можно воспользоваться и простой проверкой, но так как рассматривается работа прерываний, то воспользуемся именно ими.

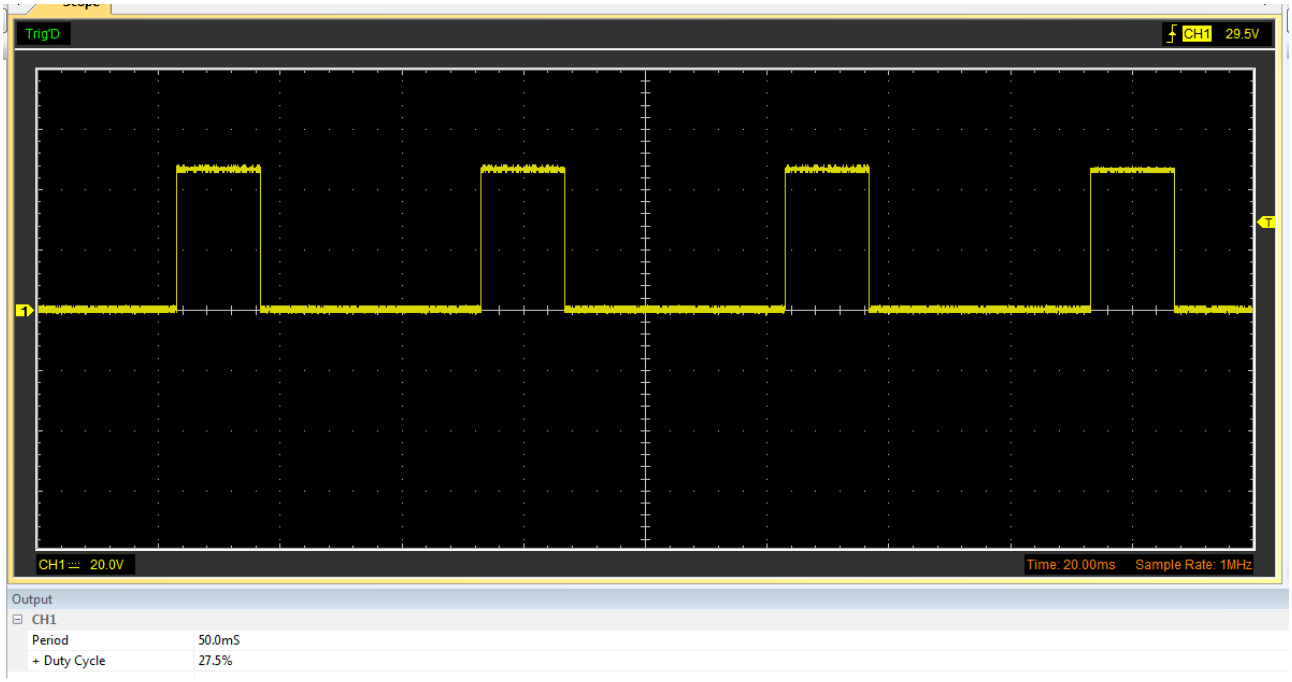

Рис. 27. Сигнал на выходе УЗ дальномера

Для перевода длительности в дальность нужно воспользоваться формулой:

$$L = \frac{(t_2 - t_1)}{58}[см]$$

Для работы с прерываниями в Arduino существует функция attachInterrupt():

```
unsigned long start = 0;
unsigned long end = 0;
void setup() {
  // put your setup code here, to run once:
  attachInterrupt(0,measure,RISING); //Прерывание на пин 2
  attachInterrupt(1,print,FALLING); //Прерывание на пин 3
//Оба пина подключены к выводу ECHO датчика
  Serial.begin(9600);
  pinMode(4,OUTPUT);//Пин для активации датчика (TRIG)
}
void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(4,HIGH);
  delayMicroseconds(10);
  digitalWrite(4,LOW);
  delay(50);
}
void measure()
{
  start = micros();
}
void print()
{
  end = micros();
  Serial.println((float)(end - start)/58);
}
```

Результат работы программы показан на рис. 28.

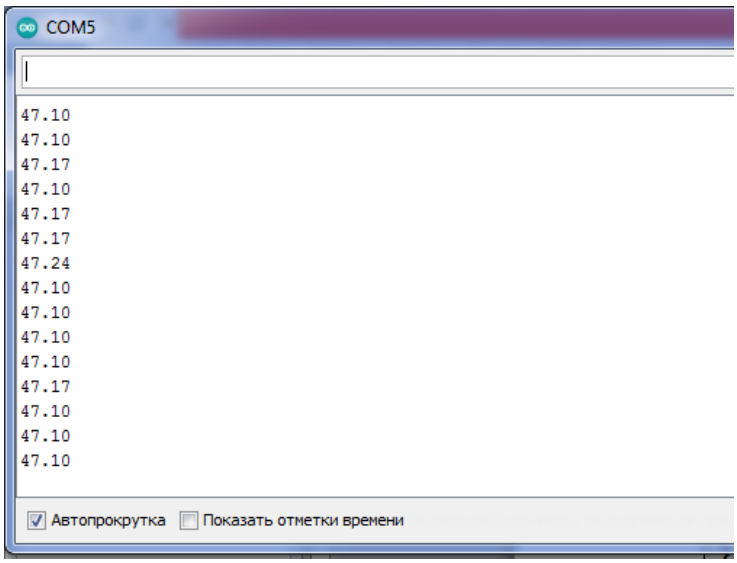

Рис. 28. Результаты измерений

Конечно, данный вариант измерений не самый лучший, и вообще есть готовые библиотеки, которые позволяют работать с датчиком без использования прерываний.

Прерывания по таймерам в библиотеке Arduino не описаны, для их вызова требуется прямое обращение к соответствующим регистрам микроконтроллера.

7. ПОЛЬЗОВАТЕЛЬСКИЕ БИБЛИОТЕКИ

Платформа Arduino стала популярна благодаря большому числу пользовательских библиотек. Их все можно найти и скачать из интернета. Эти библиотеки позволяют использовать различные устройства (экраны, цифровые генераторы, модули управления двигателями и т.д.), а также производить обработку тех или иных данных (к примеру, библиотека JSON). Библиотеки хранятся по пути: arduino/libraries. Путь к самой папке arduino зависит от места изначальной её установки. Также все библиотеки содержат примеры их использования, для ознакомления рекомендуется изучить работу этих примеров, а далее их код.

ВНИМАНИЕ! Пользовательские библиотеки не отличаются стабильностью работы и отсутствием багов, поэтому нужно использовать их крайне осторожно!

В качестве примера можно установить библиотеку для сканера отпечатков пальцев или LCD дисплея.

8. ЗАКЛЮЧИТЕЛЬНЫЙ ЭТАП

Заключительным этапом работы является написание отчёта по лабораторной работе.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бродин В.Б. Системы на микроконтроллерах и БИС программируемой логики / В.Б. Бродин, А.В. Калинин – М.: Издательство ЭКОМ, 2002. – 400 с.
2. Угрюмов Е.П. Цифровая схемотехника / Е.П. Угрюмов – 3-е изд. – СПб: БВХ-Петербург, 2010. – 816 с.
3. Партин А.С. Введение в цифровую схемотехнику / А.С. Партин, В.Г. Борисов – М.: Радио и связь, 1987. – 64 с.
4. Белов А.В. Конструирование устройств на микроконтроллерах / А.В. Белов – СПб: Наука и Техника, 2005. – 256 с.
5. Баранов В.Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы / В.Н. Баранов – 2-е изд. испр. – М.: Издательский дом «Додэка-XXI», 2006. – 288 с.
6. Голубцов М.С. Микроконтроллеры AVR: от простого к сложному / М.С. Голубцов – М.: СОЛОН-Пресс, 2003. – 288 с.

ОГЛАВЛЕНИЕ

Лабораторная работа №1. Основы разработки устройств с использованием микроконтроллера Atmega328p 3
1. Теоретическая часть 3
2. Подготовка программ и драйверов 4
3. Базовый функционал платформы Arduino 10
4. ШИМ 14
4.1. Разгон ШИМ 15
4.2. Использование ШИМ для управления нагрузкой 17
5. Связь Arduino с Qt через COM порт 20
6. Прерывания 26
7. Пользовательские библиотеки 29
8. Заключительный этап 29
Библиографический список 30

ОСНОВЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-АППАРАТНЫХ КОМПЛЕКСОВ И СИСТЕМ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторной работы №1
для студентов специальности 11.05.01
«Радиоэлектронные системы и комплексы»
очной формы обучения

Составитель
Сукачев Александр Игоревич

Издается в авторской редакции

Подписано к изданию 19.03.2024.
Уч.-изд. л. 1,7.

ФГБОУ ВО «Воронежский государственный технический университет»
394006 Воронеж, ул. 20-летия Октября, 84