

Приложение 11.5

ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Национальный исследовательский технологический университет «МИСиС»  
(НИТУ «МИСиС»)

УТВЕРЖДАЮ  
Проректор по образованию



\_\_\_\_\_/ А.А. Волков

« 31 » 12 2022 г.

Комплект материалов по курсу «Программирование микроконтроллеров»

Москва 2022 г.

№ 4384 МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ  
УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ МИСИС  
ЦЕНТР ПРОФЕССИОНАЛЬНОЙ НАВИГАЦИИ И ПРИЕМА  
Проект «IT-класс в московской школе»

М.Н. Давыдкин

# ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

Методические указания

Рекомендовано редакционно-издательским  
советом университета



Москва 2022

УДК 004.4'242  
Д13

Рецензенты:

канд. физ.-мат. наук, руководитель проектов Центра технологической поддержки образования ФГАОУ ВО «МФТИ (НИУ)» *Д.В. Савицкий*;  
канд. техн. наук, доц., заведующий учебной лабораторией АСУ ТП ФГБОУ ВО «НИУ «МЭИ» *А.А. Орлов*;  
старший преподаватель ФГАОУ ВО «НИУ «ВШЭ» *В.В. Куренков*;  
начальник отдела научно-технического творчества учащихся ФЦТТУ ФГБОУ ВО «МГТУ «СТАНКИН» *А.В. Петроченко*

**Давыдкин, Максим Николаевич.**

Д13 Программирование микроконтроллеров: метод. указания / М.Н. Давыдкин. – М. : Издательский дом НИТУ «МИСиС», 2022. – 176 с.

Современные устройства радиотехники и электроники спроектированы на базе цифровых и микропроцессорных систем. Поэтому для тех, кто занимается проектной деятельностью, очень актуальными являются знания в области цифровой техники и умение программировать встраиваемые микропроцессорные системы. Методическое пособие предназначено для знакомства с микроконтроллером семейства Atmel, встроенными периферийными модулями и принципами работы с ними. Пособие написано простым языком и дает представление о том, как программировать микроконтроллер, используя управляющие регистры, а также создание программ с использованием библиотек, рассмотрены базовые проекты.

Методическое пособие рекомендовано для учащихся системы среднего общего образования, учителей и педагогов, занимающихся проектной деятельностью, школьников, а также студентов.

004.4'242

© Давыдкин М.Н., 2022  
© НИТУ МИСИС, 2022

## Содержание

1 Классификация микроконтроллеров . . . . .	8
Классификация по разрядности шины данных ЦПУ . . . . .	8
Классификация по архитектуре вычислительной системы . . . . .	9
Классификация по фирменным платформам . . . . .	11
Классификация по выполняемым функциям . . . . .	12
Классификация по семействам 8-битных микроконтроллеров . . . . .	13
2 Структурная схема AVR-микроконтроллера . . . . .	15
Состав микроконтроллера AVR . . . . .	15
Функциональное назначение элементов микроконтроллера . . . . .	17
3 Тактовый генератор . . . . .	22
Тактовый генератор, устройство синхронизации . . . . .	22
Тактирование микроконтроллера . . . . .	23
4 Регистры портов общего назначения ввода/вывода GPIO . . . . .	26
Работа с портами ввода/вывода микроконтроллера . . . . .	27
Программирование микроконтроллера. Приоритеты операций . . . . .	30
5 Двоичная система счисления . . . . .	36
Использование двоичной системы в микроконтроллере . . . . .	36
Битовые операции . . . . .	37
Битовое И – логическое умножение . . . . .	37
Битовое ИЛИ . . . . .	40
Битовое НЕ . . . . .	42
Битовое исключающее ИЛИ . . . . .	43
Битовый сдвиг . . . . .	45

6 Управление нагрузкой на портах микроконтроллера . . . . .	48
Устанавливаем логическую «1» в требуемый бит порта . . . . .	48
Устанавливаем логическую «1» в требуемые пины порта . . . . .	51
Устанавливаем логический «0» в требуемый пин порта . . . . .	53
Проверка логических «1» или «0» в требуемом пине порта . . . . .	58
7 Подключение тактовой кнопки к микроконтроллеру . . . . .	62
Схемы подключения тактовых кнопок . . . . .	62
Дребезг контакта . . . . .	72
8 Таймер – работа со временем . . . . .	76
Функция millis() . . . . .	77
Таймеры микроконтроллера Atmega328 . . . . .	83
Регистры таймеров микроконтроллера Atmega328 . . . . .	84
Таймер/счетчик T0 (Timer/counter0) . . . . .	85
Регистр TSNT0 . . . . .	86
Регистр OCR0A . . . . .	86
Регистр OCR0B . . . . .	87
Регистр TCCR0A и TCCR0B . . . . .	87
Регистр TIMSK0 . . . . .	92
Регистр TIFR0 . . . . .	92
Таймер/счетчик T1 (Timer/counter1) . . . . .	93
Регистр OCR1A . . . . .	93
Регистр OCR1B . . . . .	94
Регистр TCCR1A и TCCR1B . . . . .	94
Регистр TIMSK1 . . . . .	99
Регистр TIFR1 . . . . .	99
Таймер/счетчик T2 (Timer/counter0) . . . . .	99
Регистр TCCR2B . . . . .	100
Регистр ASSR . . . . .	101
Регистр GTCCR . . . . .	102
Широтно-импульсная модуляция . . . . .	102
Настройка ШИМ управляющими регистрами . . . . .	105

9	Сторожевой таймер	108
	Регистр WDTCSR	108
	Библиотека avr/wdt	109
	void wdt_enable(timeout)	110
	void wdt_reset(void)	110
	void wdt_disable(void)	110
10	Аналоговые порты микроконтроллера	112
	Аналоговые сигналы и датчики	112
	Работа с аналоговыми входами	113
	analogRead(pin)	114
	analogReference(type)	114
	Настройка АЦП	
	управляющими регистрами	118
	Регистр <i>ADMUX</i>	118
	Регистр <i>ADCSRA</i>	120
	Регистр <i>ADCSRB</i>	122
11	Протокол передачи USART	125
	Характеристики модуля USART микроконтроллера	125
	Настройка USART управляющими регистрами	128
	Регистр <i>UDR0</i>	128
	Регистр <i>UCSR0A</i>	128
	Регистр <i>UCSR0B</i>	129
	Регистр <i>UCSR0C</i>	130
	Регистры <i>UBRR0H</i> , <i>UBRR0L</i>	131
12	Прерывание	137
	Регистр <i>MCUCR</i>	137
	Внешние прерывания	139
	Определение скорости двигателя с помощью энкодера	
	и прерывания	146
13	EEPROM – постоянная память данных	150
	Регистры EEPROM	150
	<i>EEARH</i> и <i>EEARL</i> – регистры адреса	151
	<i>EEDR</i> – регистр данных	151
	<i>EECR</i> – регистр управления	151
	Процедура записи в EEPROM	152
	Процедура чтения из EEPROM	153

14	Дисплей LCD1602	156
15	Создание игры с использованием дисплея LCD1602	164
	Разработка стратегии игры	164
	Реализация движения в игре	169
	Прыжок героя	169
	Движение препятствий	169
	Проверка столкновения	171
	Проверка удачного выполнения миссии и увеличение скорости	172
	Вывод героя, препятствия и счета на дисплей	172
	Обновление экрана	173

## ВВЕДЕНИЕ

Обычного обывателя не интересует, как работает современное цифровое устройство, на каких принципах и с помощью каких аппаратных средств реализована платформа модного гаджета или хайповой электронной игрушки. Вскрыв любое цифровое устройство, мы увидим различные платы, размещенные в определенном порядке. Среди таких плат с вероятностью 90 % будет микроконтроллер. Небольшой кусочек кремния, помещенный в пластиковый корпус, позволяет управлять технологическими процессами, техникой в космосе, автомобилями, бытовой техникой; участвовать в организации различных систем связи; обрабатывать аудио- и видеосигналы. Он может встраиваться в портативные электронные устройства.

Элективный курс «Программирование микроконтроллеров» позволяет познакомиться с процессом создания программного кода для микроконтроллеров, расширить знания в области настройки аппаратных модулей, интегрированных в микроконтроллер, способствует развитию собственного вектора профессиональных компетенций и созданию первых проектов с использованием программируемых микроконтроллеров.

Курс основан на самом распространенном 8-битном AVR-микроконтроллере фирмы Atmel семейства Atmega328P. Данный микроконтроллер представлен в оборудовании лабораторий проекта «ИТ-класс в московской школе». Получив знания и овладев навыками и умениями в области программирования данных типов микроконтроллера, обучающийся легко сможет перейти на изучение других типов микроконтроллеров.

## 1 Классификация микроконтроллеров

При выполнении проекта с использованием программируемого микроконтроллера важной задачей является выбор микроконтроллера. Разработчик сталкивается с выбором следующих параметров:

- достаточность периферийных устройств, входящих в состав микроконтроллера;
- цена микроконтроллера;
- скорость обработки информации микроконтроллером;
- память микроконтроллера.

Среди разработчиков данные вопросы вызывают жаркие споры, и нет единого правильного решения при выборе микроконтроллера. Рассмотрим классификацию микроконтроллеров, которая позволит вам лучше понять многообразие существующих микроконтроллеров на рынке.

### Классификация по разрядности шины данных ЦПУ

Микроконтроллеры по разрядности шины данных можно разделить следующим образом:

- 4-битные (Atmel MARC4, Winbond W742, NEC uPD75 и др.);
- 8-битные (Intel MCS-48, Intel MCS-51, Atmel ATtiny/Atmega/ATXmega, Microchip PIC12/16/18, Zilog Z86 и др.);
- 16-битные (Intel MCS-96, Texas Instruments MSP430, Motorola 68HC16, Fujitsu MB90, Infineon C16, Mitsubishi M16C, Microchip PIC24 и др.);
- 32-битные (Atmel ARM, Fujitsu MB91, NEC V850, NXP LPC2xxx и др.).

Разработчики-любители предпочтение отдают 8-битным микроконтроллерам. На сегодняшний день лидерами продаж среди производителей 8-битных микроконтроллеров являются фирмы Atmel и Microchip. Профессиональные раз-

работчик отдадут предпочтение микроконтроллерам с большей разрядностью.

## Классификация по архитектуре вычислительной системы

По архитектуре вычислительной системы микроконтроллеры можно разделить на:

- CISC (англ. complex instruction set computing);
- RISC (Reduced Instruction Set Computing).

Первые микроконтроллеры имели стандартную CISC-архитектуру. Особенности CISC: команды выполняются поочередно друг за другом и имеют разную длину и структуру. Выборка команды из памяти осуществляется побайтно и выполняется за несколько тактов. CISC-архитектуру имеют МК из семейства Motorola HC05/HC08, МК с ядром MCS-51, МК из семейства Infineon C500 и ряд других.

В начале 1980-х годов была разработана новая архитектура – RISC (англ. *reduced instruction set compute*) (аббревиатуру предложил Д. Паттерсон из Калифорнийского университета в г. Беркли, США). Основная идея заключается в замене сложных команд однотипными простыми и выполнении их единым потоком на параллельном конвейере. Все команды имеют фиксированную длину и в идеале должны выполняться за один, а не за несколько тактов, чем достигается повышенное быстродействие.

Одним из первых микроконтроллеров с архитектурой RISC стал PIC-контроллер 16C54 фирмы Microchip. Благодаря высокой производительности и трем десяткам легко запоминающихся команд PIC-контроллеры быстро завоевали популярность во всем мире. Вскоре примеру фирмы Microchip последовали разработчики из фирм Atmel, Scenix и др.

С точки зрения принципов конструирования вычислительных систем выделяют прынстонскую и гарвардскую архитектуры. Прынстонская архитектура была разработана Джоном фон Нейманом и независимо от него академиком

С.А. Лебедевым. В ней используется общая память для хранения программ и данных (рисунок 1.1). Основное преимущество заключается в упрощении схемотехники ЦПУ и в гибкости распределения ресурсов между областями памяти.

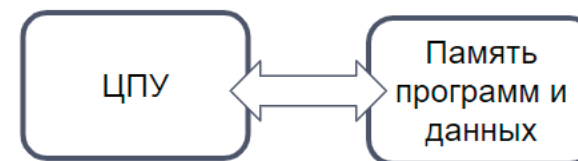


Рисунок 1.1 – Прынстонская архитектура

Особенностью гарвардской архитектуры является наличие отдельных адресных пространств для хранения команд и данных (рисунок 1.2). Эта архитектура почти не использовалась до конца 1970-х годов, пока разработчики микроконтроллера наконец-то поняли, что именно она дает им определенные преимущества. В частности, анализ реальных программ показывает, что объем памяти данных микроконтроллера, используемый для хранения промежуточных результатов, примерно на порядок меньше объема памяти, требуемого для хранения программ. Значит, можно сократить разрядность шины данных, уменьшить число транзисторов в микросхеме, а заодно и ускорить доступ к информации сразу в обоих «полушариях» памяти. Как следствие, сейчас большинство современных МК используют RISC-архитектуру гарвардского типа.



Рисунок 1.2 – Гарвардская архитектура

## Классификация по фирменным платформам

Каждая фирма – производитель микроконтроллеров поддерживает свою платформу, т.е. внутрифирменные стандарты, технологии, конструктивные особенности, запатентованные ноу-хау, разный подход к засекречиванию средств отладки и программирования. Поэтому микроконтроллеры разных фирм имеют различие в электрических параметрах, сферах применения и рыночной популярности. Согласно исследованиям, представленным в материале [1] и отображенным на рисунках 1.3 и 1.4, можно понять, какая фирма пользуется большей популярностью в России и что лидером рынка остается компания Microchip/Atmel.

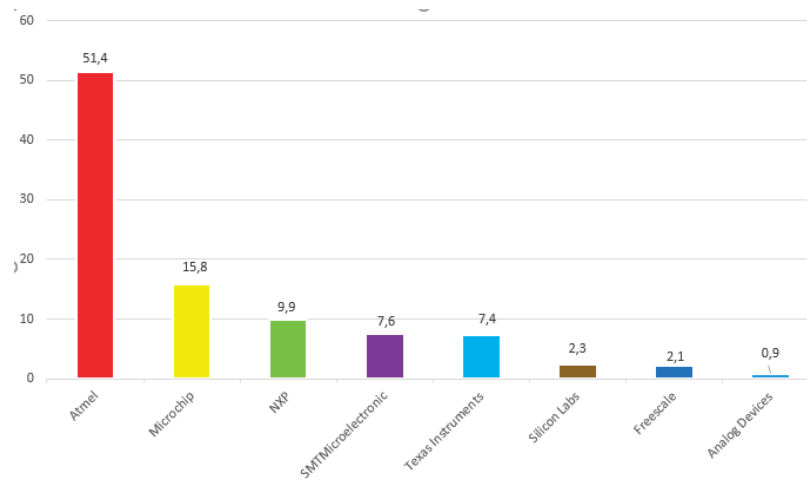


Рисунок 1.3 – Запрашиваемые производители микроконтроллеров

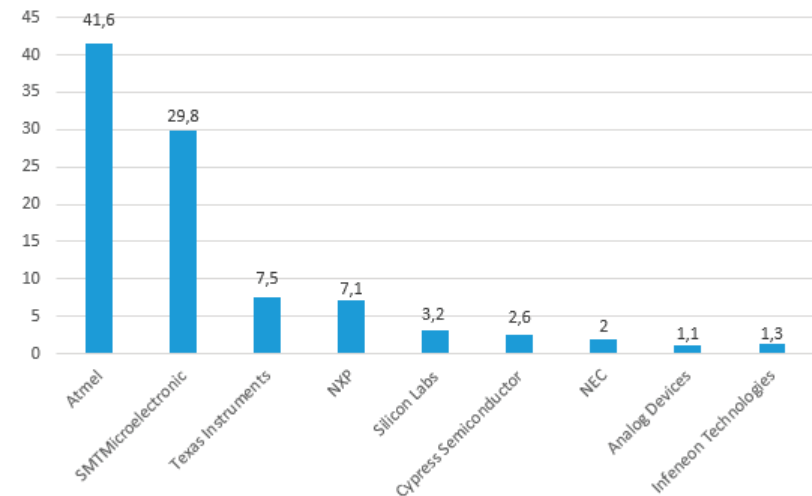


Рисунок 1.4 – Запрашиваемые производители микроконтроллеров в России

## Классификация по выполняемым функциям

Микроконтроллеры по выполняемым функциям можно разделить следующим образом:

- универсальные;
- специализированные.

Многие микроконтроллеры оснащены богатой периферией: порты ввода/вывода, таймеры/счетчики, канал последовательного доступа UART, аналоговый компаратор и т.д. Такие микроконтроллеры будем называть универсальными.

При эксплуатации таких микроконтроллеров были выявлено, что большинство периферийных устройств не используется, а на выполнение специфических задач микроконтроллер затрачивает много машинного времени. В связи с требованием рынка появились специализированные микро-

контроллеры, которые решают конкретную задачу, например драйверы шаговых двигателей, драйвер символического дисплея, Ethernet-адаптер, Bluetooth-микроконтроллер. Такие микроконтроллеры будем называть специализированными.

В проектах любителей удобно использовать универсальные микроконтроллеры, так как при развитии проекта может появиться желание интеграции функции устройства, которой ранее не предполагалось. При появлении опыта разработчик больше предпочтение отдает специализированным микроконтроллерам, и структура системы управления становится распределенной.

## Классификация по семействам 8-битных микроконтроллеров

Рассмотрим классификацию 8-битных микроконтроллеров. Среди разнообразия фирм можно выделить следующие семейства:

- ядро MCS-51 – AT89Cх051, AT89C5х, AT89S (Atmel), DS89 (Maxim/Dallas);
- ядро AVR – ATtiny, AT90S, Atmega, ATXmega (Atmel);
- ядро PIC – PIC10, PIC12, PIC16, PIC18 (Microchip);
- ядро SX – SXxxx (Uvicom, ранее Scenix);
- ядро 68HC – 68HC08, 68HC12 (Freescale Semiconductor, ранее Motorola);
- ядро ST – ST62, ST7 (STMicroelectronics, ранее SGS-THOMSON);
- ядро CIP-51 – C8051 (Silicon Laboratories, ранее Cygnal Integrated);
- ядро 8052 – W78E516 (Winbond);
- ядро «ТЕСЕЙ» – КР1878ВЕ1 («Ангстрем»).

Во многих источниках терминология, описывающая микроконтроллеры, трактуется по-разному, поэтому введем некоторые определения.

Ядро – базовое устройство внутренней вычислительной системы. Ядро определяет систему команд, шинный интер-

фейс, архитектуру памяти, т.е. коренные отличия «вычислителей» друг от друга. Процессорное ядро может быть одинаковым, а фирмы-изготовители – разными.

Семейство – группа микросхем, имеющих одно ядро, у которых примерно одинаковый набор программных и периферийных функций. Семейство может разбиваться на более мелкие подсемейства.

Серия, линейка – это фирменный бренд или рекламный слоган, например серия «Classic», серия «MEGA», линейка «MegaPIC».

Модель – несколько микросхем одного семейства, различающиеся между собой второстепенными цифрами (буквами) в названии, что определяет разный температурный диапазон, тактовую частоту, вариант корпуса, питание.

Рассмотрим на примере микроконтроллера AVR фирмы Atmel 8-битный микроконтроллер Atmega328P-PU, расшифровка его обозначения представлена на рисунке 1.5. Данный микроконтроллер в методическом пособии будет играть важную роль, так изучение программирования микроконтроллера начнем именно с него.

АТ	MEGA	32	8	Р	–	PU
Фирма – производитель микроконтроллера Atmel	Семейство микроконтроллера	32КВ FLASH	Модификация 8	Корпус DIP		Диапазон температур – от –40 до +85 °С

Рисунок 1.5 – Маркировка микроконтроллера Atmega328P-PU



## 2 Структурная схема AVR-микроконтроллера

### Состав микроконтроллера AVR

Микроконтроллер, который мы рассмотрим в данном пособии, относится к 8-битным AVR-микроконтроллерам фирмы Atmel семейства Atmega. Внешний вид микроконтроллера представлена на рисунке 2.1, а его структурная схема – на рисунке 2.2.

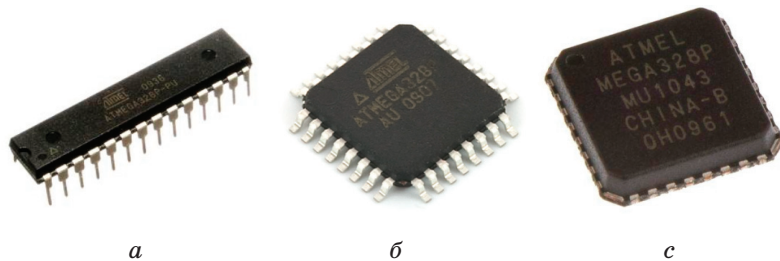


Рисунок 2.1 – Внешний вид микроконтроллера Atmel Atmega328 в различных корпусах:  
 а – DIP; б – TQFP; в – VQFN

Главная часть микроконтроллера – центральный процессор ЦПУ (CPU). Основная функция ядра ЦП заключается в обеспечении правильного выполнения программы. Таким образом, ЦП должен иметь доступ к памяти, выполнять вычисления, управлять периферийными устройствами и обрабатывать прерывания.

В состав ядра входит:

- АЛУ – арифметическое логическое устройство (ALU – Arithmetic Logic Unit);
- оперативная память (RAM – Random Access Memory);
- регистр команд;
- декодер команд;

- 32 регистра общего назначения;
- счетчик команд;
- блок внутренней отладки (OCD – On-Chip Debugger).

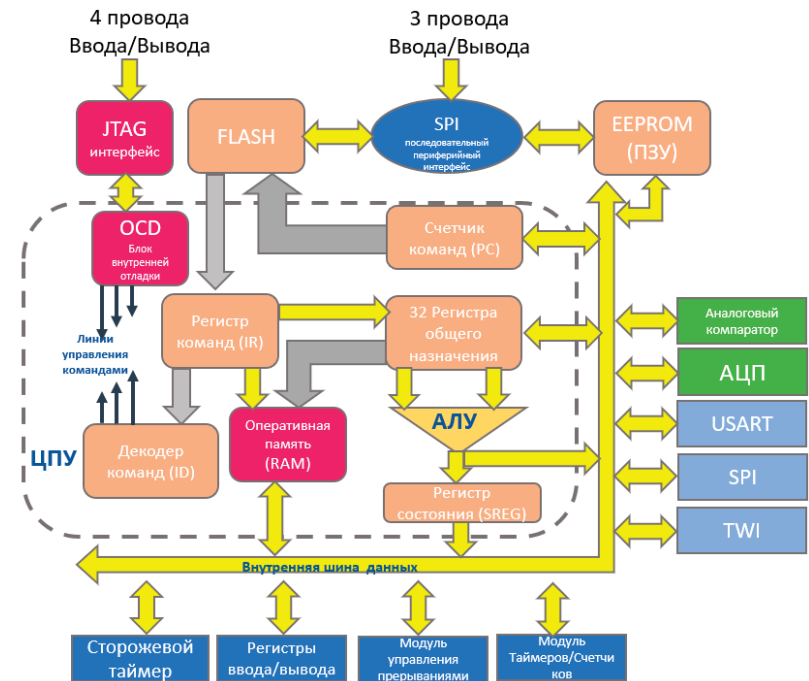


Рисунок 2.2 – Структурная схема микроконтроллера AVR

- В состав микроконтроллера также входят:
- ПЗУ – перепрограммируемое запоминающее устройство (EEPROM – Electrically Erasable Programmable Read-Only Memory), энергонезависимая память;
  - FLASH – перепрограммируемая память для сохранения программы;
  - блок сравнения аналоговых сигналов, аналоговый компаратор;
  - аналого-цифровой преобразователь;

- модуль управления ICD дисплеем;
- универсальный асинхронный приемопередатчик (US-ART – Universal Asynchronous Receiver-Transmitter);
- последовательный интерфейс с двухпроводным подключением (TWI – Two-Wire serial Interface);
- сторожевой таймер;
- порты ввода/вывода;
- блок управления прерываниями;
- модуль таймеров и счетчиков;
- четырехпроводной интерфейс внутренней отладки (JTAG – Joint Test Action Group);
- последовательный периферийный интерфейс (SPI – Serial Peripheral Interface).

Чтобы максимизировать производительность и параллельную работу, AVR использует гарвардскую архитектуру – с отдельными памятью и шинами для программы и данных. Инструкции в памяти программы выполняются с одноуровневой конвейерной обработкой. Пока одна инструкция выполняется, следующая инструкция предварительно выбирается из памяти программы. Эта концепция позволяет инструкции выполняться в каждом такте. Взаимосвязь модулей и блоков внутри микроконтроллера осуществляется по внутренней 8-разрядной шине данных.

## **Функциональное назначение элементов микроконтроллера**

**Основная функция ядра ЦП (CPU)** заключается в обеспечении правильного выполнения программы. Таким образом, ЦП должен иметь доступ к памяти, выполнять вычисления, управлять периферийными устройствами и обрабатывать прерывания.

**АЛУ – арифметическое логическое устройство** – синхронно с тактовым сигналом и основываясь на состоянии счетчика команд (Program Counter) выбирает из памяти программ (FLASH) очередную команду и производит ее выполнение.

Тактовый сигнал для микроконтроллера вырабатывается тактовым генератором и может быть подан из нескольких доступных источников на выбор:

- встроенный кварцевый генератор с подключаемым внешним резонатором;
- керамический или кварцевый резонатор с конденсаторами;
- внешний тактовый сигнал.

Установка источника тактовых импульсов производится при помощи FUSE-битов.

**FUSES** (от англ. плавление, пробка, предохранитель) – специальные 4 байта ( $4 \times 8 = 32$  бит) данных, которые настраивают некоторые глобальные параметры микроконтроллера в процессе прошивки. После прошивки данные биты нельзя изменить через внутреннюю программу, что записана в МК.

Данной конфигурацией бит мы указываем микроконтроллеру следующее:

- какой задающий генератор использовать (внешний или внутренний);
- делить частоту генератора на коэффициент или нет;
- использовать ножку сброса (RESET) для сброса или же как дополнительный пин ввода/вывода;
- количество памяти для загрузчика;
- другие настройки в зависимости от используемого микроконтроллера.

## **EEPROM**

**EEPROM** (англ. Electrically Erasable Programmable Read-Only Memory) – электрически стираемое перепрограммируемое ПЗУ – энергонезависимая память данных, в которой данные будут храниться даже при отключении питания микроконтроллера. В данной памяти можно хранить настройки выполнения программы, собранные данные для статистики работы устройства и другую полезную информацию. К примеру, собрав маленькую метеостанцию на микроконтроллере, в EEPROM на каждый день можно сохранять данные о температуре воздуха, давлении, силе ветра, а потом

в любой момент считать эти собранные данные и провести статистические исследования.

Для EEPROM выделено отдельное адресное пространство, которое отличается от адресного пространства RAM и FLASH. Память EEPROM микроконтроллера – очень ценный ресурс, поскольку ее, как правило, очень мало – от 0,5 до нескольких килобайт на чип. Количество перезаписей для данного типа памяти составляет порядка 100 000, что в **10 раз больше, чем ресурс FLASH-памяти.**

### **Analog Comparator**

**Analog Comparator** – блок, который сравнивает между собой два уровня сигнала и запоминает результат сравнения в определенном регистре, после чего можно данный результат проанализировать и выполнить необходимые действия. Для примера: можно использовать этот блок как АЦП (аналогово-цифровой преобразователь) и измерять напряжение батареи питания, в случае если напряжение батареи достигло низкого уровня – произвести некоторые действия, помигать красным светодиодом и т.п. Также данный модуль можно применять для измерения длительности аналоговых сигналов, считывания установленных режимов работы устройства при помощи потенциометра и т.п.

### **A/D Converter**

**A/D Converter** – блок, который преобразовывает аналоговое значение напряжения в цифровое значение, с которым можно работать в программе и на основе которого можно выполнять определенные действия. Как правило, диапазон напряжений, что подаются на вход АЦП в AVR-микроконтроллере, находится в пределах 0–5,5 В. Для данного блока очень важно, чтобы микроконтроллер питался от стабильного и качественного источника питания. Во многих AVR-микроконтроллерах есть специальный отдельный вывод для подачи стабильного питания на схему АЦП.

## **USART**

**USART** – последовательный асинхронный интерфейс для обмена данными с другими устройствами. Есть поддержка протокола RS-232, благодаря чему микроконтроллер можно соединить для обмена данными с компьютером.

Для подобной связи МК с COM-портом компьютера нужен конвертер логических уровней напряжения (+12 В – для COM; +5 В – для микроконтроллера), или же просто RS232-TTL. Для подобных целей используют микросхемы MAX232 и им подобные.

Для подключения микроконтроллера к компьютеру через USB используя UART-интерфейс, можно использовать специализированную микросхему FT232RL. Таким образом, на новых компьютерах и ноутбуках можно, не имея физического COM-порта, привязать микроконтроллер, используя USB-порт через USART-интерфейс.

## **TWI**

**TWI** – интерфейс для обмена данными по двухпроводной шине. К такой шине данных можно подключить до 128 различных устройств, используя две линии данных: тактовый сигнал (SCL) и сигнал данных (SDA). Интерфейс TWI является аналогом базовой версии интерфейса I2C.

В отличие от SPI-интерфейса (один мастер и один/несколько ведомых) интерфейс TWI – двунаправленный, что позволяет организовать между несколькими микроконтроллерами небольшую внутреннюю сеть.

## **Watchdog Timer**

**Watchdog Timer** представляет собою систему контроля зависания устройства с последующим его перезапуском. Это как автоматическая кнопка RESET для старенького компьютера.

## I/O Ports, GPIO

**I/O Ports, GPIO** – это набор блоков портов ввода/вывода, к пинам которых можно подключить разнообразные датчики, исполняющие устройства и цепи. Количество пинов вход/выход, что идут от портов в микроконтроллере, может быть от 3 до 86.

Выходные драйверы в портах AVR-микроконтроллера позволяют напрямую подключать нагрузку с потребляемым током 20 мА (максимум 40 мА) при напряжении питания 5В. Общий нагрузочный ток для одного порта не должен превышать значение в 80 мА (например, на 4 пина для одного из портов повесить по светодиоду с током 15–20 мА).

## Interrupts

**Interrupts** – это блок, который отвечает за реакцию и запуск на выполнение определенных функций при поступлении сигнала на определенные входы микроконтроллера или же по какому-то внутреннему событию (например, тиканью таймера). Под каждое прерывание разрабатывается и записывается в память отдельная подпрограмма.

Почему этот блок называется блоком прерываний? Потому что при возникновении определенного для прерывания события выполнение основной программы прерывается и происходит приоритетное выполнение подпрограммы, которая написана для текущего прерывания. По завершении выполнения подпрограммы происходит возвращение к выполнению основной программы с того момента, где она была прервана.

## Timers/Counters

**Timers/Counters** – набор таймеров и счетчиков. Микроконтроллер, как правило, содержит от одного до четырех таймеров и счетчиков. Они могут применяться для подсчета количества внешних событий, формирования сигналов определенной длительности, выработки запросов на прерывания и т.п. Разрядность таймеров и счетчиков составляет 8 и 16 бит (смотреть в даташите для чипа).

## 3 Тактовый генератор

Тактовый генератор – это сердце микроконтроллера. По каждому такту или импульсу тактового генератора происходит какая-нибудь операция, передаются данные по шинам и регистрам, работают таймеры, переключаются порты ввода/вывода. Чем больше тактовая частота, тем больше энергии нужно микроконтроллеру.

Импульсы формируются тактовым генератором с определенной скоростью (частотой). Сам генератор может быть как внутренний, так и внешний. Все это гибко настраивается.

### Тактовый генератор, устройство синхронизации

Упрощенная схема устройства синхронизации представлена на рисунке 3.1. Опираясь на основной тактовый сигнал, поступающий в микроконтроллер, формируются дополнительные сигналы, используемые для тактирования и различных модулей, и блоков микроконтроллера:

- `clkCPU` – тактовый сигнал центрального процессора, используется для тактирования блоков микроконтроллера, отвечающих за работу с ядром микроконтроллера (регистровый файл, память данных и т.п.). При выключении этого сигнала ЦПУ останавливается, все вычисления прекращаются;

- `clkI/O` – тактовый сигнал подсистемы ввода/вывода, используется большинством периферийных устройств, таких как таймеры/счетчики и интерфейсные модули. Этот сигнал используется также подсистемой внешних прерываний, однако некоторые внешние прерывания могут генерироваться и при его отсутствии;

- `clkFLASH` – тактовый сигнал для управления FLASH-памятью программ. Как правило, этот сигнал формируется одновременно с тактовым сигналом центрального процессора;

- `clkASY` – тактовый сигнал асинхронного таймера/счетчика. Тактирование осуществляется непосредственно от внешнего кварцевого резонатора (32 768 Гц). Наличие это-

го сигнала позволяет использовать соответствующий таймер/счетчик в качестве часов реального времени даже при нахождении микроконтроллера в «спящем» режиме;

- clkADC – тактовый сигнал модуля АЦП. Наличие этого тактового сигнала позволяет осуществлять преобразования при остановленных ЦПУ и подсистеме ввода/вывода. При этом значительно уменьшается уровень помех, генерируемых микроконтроллером, точность преобразования увеличивается.

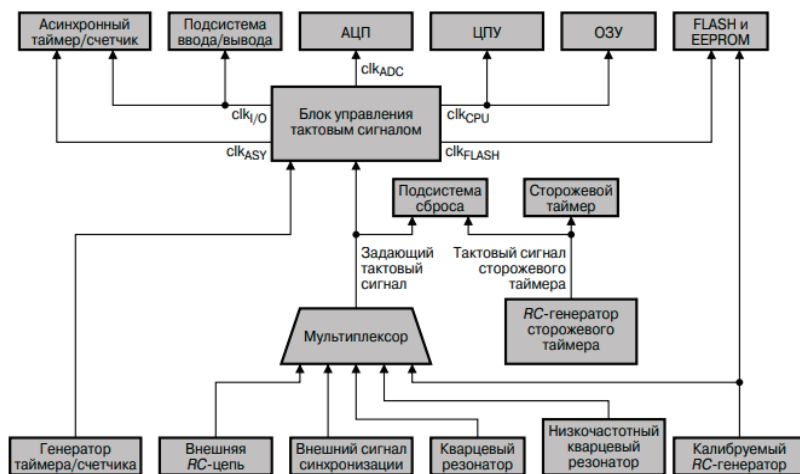


Рисунок 3.1 – Структурная схема устройства синхронизации

## Тактирование микроконтроллера

Тактирование микроконтроллера Atmega328 может осуществляться от четырех источников тактового сигнала [2], а именно:

- внутренний генератор с внутренней задающей RC-цепочкой. При таком тактировании никакой обвязки не нужно. К выводам XTAL1 и XTAL2 можно ничего не подключать,

их можно использовать как обычные порты ввода/вывода. Внутренний RC-генератор можно настроить на четыре значения частоты;

- внутренний генератор с внешней задающей RC-цепочкой (рисунок 3.2, а). Тактирование аналогично предыдущему способу, только вот задающая RC-цепочка находится не внутри МК, а снаружи, такая схема позволяет изменять частоту прямо на ходу. Изменение задающей частоты происходит путем изменения значения сопротивления;

- внутренний генератор с внешним задающим кварцевым резонатором (рисунок 3.2, б). В этом случае снаружи МК цепляют кварцевый резонатор с небольшой обвязкой кварца из двух конденсаторов. Если используется кварц (резонатор) с частотой менее 1 МГц, то конденсаторы можно и не ставить;

- внешний генератор (рисунок 3.2, в). В этом случае импульсы поступают на вход МК от внешнего генератора. Такое тактирование применяют, когда нужно, чтобы несколько независимых микроконтроллеров работали синхронно от одного генератора.

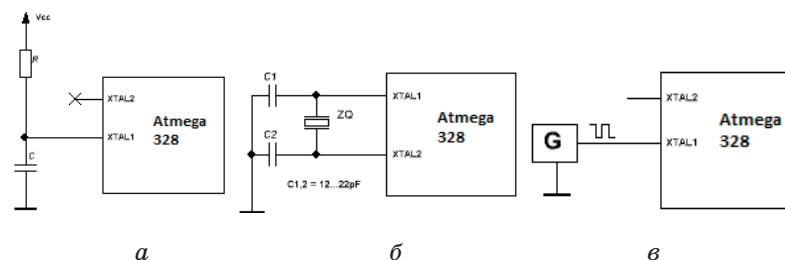


Рисунок 3.2 – Тактирование микроконтроллера

Во всех проектах, которые будут описаны в данном пособии, мы используем плату разработчика Arduino UNO (далее – плата разработчика), в которой тактирование сигнала осуществляется с помощью внутреннего генератора с внешним задающим кварцевым резонатором, схема подключения представлена на рисунке 3.3.

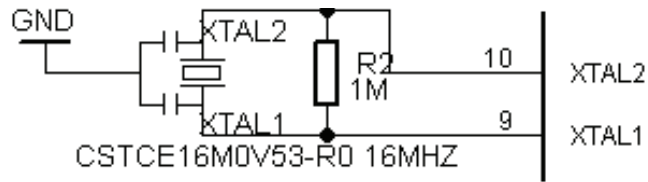


Рисунок 3.3 – Схема подключения резонатора в плате разработчика Arduino Uno

## 4 Регистры портов общего назначения ввода/вывода GPIO

В цифровом мире минимальной единицей памяти является ячейка, содержащая 0 или 1. Данная ячейка называется битом. Atmega328 является 8-битным микроконтроллером. Максимальный размер регистров и память микроконтроллера разбита таким образом, что максимально оперируют 8 битами. Группа из 8 битов называется байтом. Каждый регистр имеет размерность, кратную 8 битам. Структура 8-битного регистра представлена на рисунке 4.1. Нумерация в регистре осуществляется с младшего бита и начинается с 0.



Рисунок 4.1 – Структура регистра

Умение работать с битами в микроконтроллере позволит вам:

- 1) работать напрямую с регистрами микроконтроллера;
- 2) работать с внешними микросхемами и осуществлять быстрый парсинг сигнала из двоичной системы;
- 3) эффективно организовывать хранение данных: упаковывать информацию с цифровых выводов в одну переменную и распаковывать обратно;
- 4) создавать символы и другую информацию для матричных дисплеев;
- 5) понимать чужой код;
- 6) осуществлять быстрые вычисления.

## Работа с портами ввода/вывода микроконтроллера

В микроконтроллере Atmega328 порты ввода/вывода состоят из нескольких выводов (пины), через них микроконтроллер может получать и отправлять цифровые сигналы. Направление передачи данных осуществляется программно и может быть изменено в любой момент времени.

Микроконтроллер Atmega328 имеет четыре 8-разрядных порта ввода/вывода (порты A, B, C, D).

Для управления портами общего назначения существуют следующие регистры управления:

- **DDRX** (Data Direction Register) – регистр направления порта;
- **PORTX** (Data Register) – регистр данных;
- **PINX** (Input Pins Address) – регистр чтения.

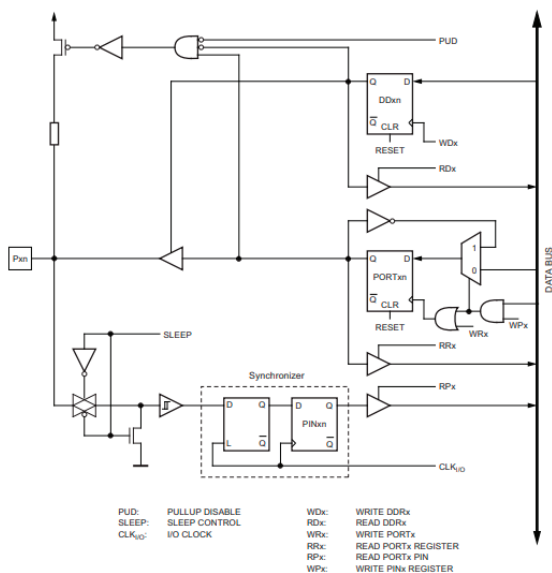


Рисунок 4.2 – Функциональная схема порта общего назначения

Вместо значения x прописывается требуемый порт микроконтроллера (B, C, D).

Внутренняя схема подключения портом микроконтроллера Atmega328 представлена на рисунке 4.2. Порты являются двунаправленными портами ввода/вывода с дополнительным внутренним подтягивающим резистором.

На рисунке 4.2 WRx, WPx, WDx, RRx, RPx и RDx являются общими для всех контактов одного и того же порта, CLK<sub>I/O</sub>, SLEEP и PUD – общими для всех портов.

### Регистр DDRx

Регистр направления порта (Data Direction Register) представляет собой 8-битный регистр, представленный на рисунке 4.3, биты которого формируют режим работы порта. Если в бит регистра установлен лог. «0», то вывод работает как ВХОД, если в бит регистра установлена лог. «1», то вывод работает как ВЫХОД.

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	<b>DDB7</b>	<b>DDB6</b>	<b>DDB5</b>	<b>DDB4</b>	<b>DDB3</b>	<b>DDB2</b>	<b>DDB1</b>	<b>DDB0</b>	<b>DDRB</b>
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	–	<b>DDC6</b>	<b>DDC5</b>	<b>DDC4</b>	<b>DDC3</b>	<b>DDC2</b>	<b>DDC1</b>	<b>DDC0</b>	<b>DDRC</b>
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	<b>DDD7</b>	<b>DDD6</b>	<b>DDD5</b>	<b>DDD4</b>	<b>DDD3</b>	<b>DDD2</b>	<b>DDD1</b>	<b>DDD0</b>	<b>DDRD</b>
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 4.3 – Регистры DDRx микроконтроллера Atmega328

## Регистр PORTx

Регистр данных (Data Register) режим управления состоянием вывода представляет собой 8-битный регистр, представленный на рисунке 4.4, биты которого в зависимости от выбранного режима в регистре DDRx устанавливают состояния выхода порта:

- когда ножка настроена на выход, значение соответствующего бита в регистре PORTx определяет состояние вывода. Если  $PORTx_u=1$  то на выводе лог. «1», если  $PORTx_u=0$ , то на выводе лог. «0». Когда ножка настроена на вход, если  $PORTx_u=0$ , то вывод в режиме Hi-Z. Если  $PORTx_u=1$ , то вывод в режиме PullUp с подтяжкой резистором в 100к до питания;

- **Вход Hi-Z** – режим высокоимпендансного входа;
- **Вход PullUp** – вход с подтяжкой.

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x08 (0x28)	-	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 4.4 – Регистры PORTx микроконтроллера Atmega328

## Регистр PINx

Регистр чтения (Input Pins Address) представляет собой 8-битный регистр, представленный на рисунке 4.5,

из него можно только читать. В регистре PINx содержится информация о реальном текущем логическом уровне на выводах порта.

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	-	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Рисунок 4.5 – Регистры PINx микроконтроллера Atmega328

## Программирование микроконтроллера. Приоритеты операций

Любое выражение, написанное на языке программирования, состоит из операндов (переменных, констант и т.д.), соединенных знаками операций. Операции выполняются в строгой последовательности. Величина, определяющая преимущественное право на выполнение той или иной операции, называется приоритетом. Приоритеты операций, используемых в языках программирования C++ и Arduino, представлены на рисунке 4.6. Порядок выполнения операций может регулироваться с помощью круглых скобок.



Приоритет	Оператор	Описание
1	::	Разрешение области видимости
2	a++ a--	Суффиксный/постфиксный инкремент и декремент
	тип() тип{}	Функциональный оператор приведения типов
	a()	Вызов функции
	a[]	Индексация
3	.	Доступ к элементу
	++a --a	Префиксный инкремент и декремент
	+a -a	Унарные плюс и минус
	! ~	Логическое НЕ и побитовое НЕ
	(тип)	Приведение типов в стиле C
	*a	Косвенное обращение (разыменование)
	&a	Взятие адреса
	sizeof	Размер в байтах
	co_await	Выражение await (C++20)
	new new[]	Динамическое распределение памяти
delete delete[]	Динамическое освобождение памяти	
4	.* ->*	Указатель на элемент
5	a*b a/b a%b	Умножение, деление и остаток от деления
6	a+b a-b	Сложение и вычитание
7	<< >>	Побитовый сдвиг влево и сдвиг вправо
8	<=>	Оператор трёхстороннего сравнения (C++20)
9	< <= > >=	Для операторов отношения < и ≤ и > и ≥ соответственно
10	== !=	Операторы равенства = и ≠ соответственно
11	&	Побитовое И
12	^	Побитовый XOR (исключающее или)
13		Побитовое ИЛИ (включающее или)
14	&&	Логическое И
15		Логическое ИЛИ
16	a?b:c	Тернарный условный оператор
	throw	Оператор throw
	co_yield	Выражение yield (C++20)
	=	Прямое присваивание
	+= -=	Присваивание с сложением и вычитанием
	*= /= %=	Присваивание с умножением, делением и остатком
	<<= >>=	Присваивание с побитовым сдвигом влево и вправо
&= ^=  =	Присваивание с побитовым И, XOR и ИЛИ	

Рисунок 4.6 – Приоритеты операций в языке программирования

Рассмотрим различные режимы работы с портами общего назначения на примере порта D. Схема подключения для работы с портом D представлена на рисунке 4.7. В плате разработчика порты D соответствуют цифровым выводам 0–7 (см. рисунок 4.7 с принципиальной схемой платы разработчика).

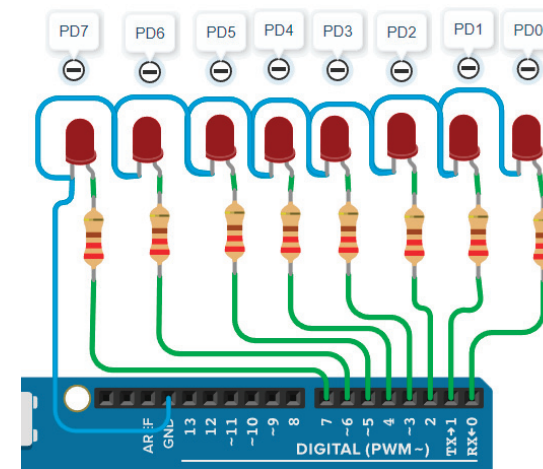


Рисунок 4.7 – Схема для анализа режима работы «ВЫХОД» пинов порта D

Рассмотрим программу, написанную языком верхнего уровня Arduino. Язык Arduino похож на язык программирования C++, но им не является. Многие конструкции языка программирования C++ в языке программирования Arduino были сохранены. Целью данного методического пособия является изучение не языка программирования, а программирования микроконтроллера. В связи с этим некоторые моменты, связанные с конструкциями языка программирования Arduino и C++, излагаются в том объеме, чтобы пользователь смог понять, как выполняется код программы. Рекомендуется дополнительно изучать язык программирования C++. Любая программа, даже пустая, должна содержать две функции:

- void setup()
- void loop()

Функция **void setup()** выполняется однократно в момент получения питания микроконтроллером. Обычно в данной функции производится предустановка микроконтроллера (инициализируются протоколы, устанавливаются режимы работы портов микроконтроллера и т.д.).

Функция **void loop()** выполняется после функции **setup()** и выполняется многократно, пока микроконтроллер не обесточится или не выйдет из строя.

Листинг программы на языке Arduino, выполняющий алгоритм мигания светодиода на 2-м цифровом выводе порта с периодом 500 мс, представлен ниже:

```
1 void setup()
2 {
3   pinMode(2, OUTPUT); //установка режима работы вывода
   2 как выход
4 }
5 void loop()
6 {
7   digitalWrite(2, 1); //установка на вывод 2 логиче-
   ской единицы
8   delay(500); // задержка 500 миллисекунд
9   digitalWrite(2, 0); //установка на вывод 2 логиче-
   ского нуля
10  delay(500); // задержка 500 миллисекунд
11 }
```

Рассмотрим подробнее, как выполняется код. Выполнение кода осуществляется последовательно с первой строчки и по порядку. В первой строке формируется функция **setup ()**, тело данной функции заключено в фигурные скобки:

```
{ – начало тела функции;
} – конец тела функции.
```

В строке три используется функция **pinMode ()**, которая устанавливает режим работы на конкретный пин порта:

```
INPUT – вход;
OUTPUT – выход.
```

Запись **pinMode(2, OUTPUT);** формирует второй порт платы разработчика на выход.

После выполнения команды необходимо ставить знак «;».

После функции **setup ()** вызывается функция **void loop ()** (5-я строка), тело данной функции находится на строках 6–11.

В строке 7 вызывается функция **digitalWrite(2, 1)** которая устанавливает лог. «1» на 2-й цифровой вывод платы разработчика. В строке 8 вызывается функция **delay(500)**, которая задерживает выполнение кода микроконтроллера на указанное время в скобках – 500 мс. Использование функции **delay ()** сильно замедляет работу микроконтроллера, так как в этот момент микроконтроллер ничего не выполняет, он только ждет, пока закончится время. В разделах данного методического пособия мы разберем решение, при котором избавимся от функции **delay()**. На первом этапе знакомства пока в программе оставим данную функцию.

Далее программа попадает на 9-ю строку **digitalWrite(2, 0)**, которая устанавливает лог. «0» на 2-й цифровой вывод платы разработчика. В строке 10 вызывается функция **delay(500)**, которая задерживает выполнение кода микроконтроллера на указанное время в скобках – 500 мс. Далее процесс повторяется, и мы как исследователи видим, что на 2-м порте D микроконтроллера происходит мигание светодиода.

Как видно, в листинге программы даны комментарии по ходу выполнения команды, прочитав которые, мы начинаем понимать, что происходит в каждой строчке. Комментарии могут быть строчными, они обозначаются знаком **//**, т.е. все, что написано после данных знаков, компилятор не воспринимает как код программы. Очень часто разработчику приходится комментировать целые абзацы, для этого используются абзацные комментарии, их помещают в следующие символы:

```
/* – начало комментария;
*/ – конец комментария.
```

Все что находится между **/\* \*/**, компилятор не воспринимает как код программы.

При компиляции данная программа занимает 924 байт.

Рассмотрим пример выполнения того же самого алгоритма, но теперь управление портом D будем осуществлять через непосредственное управление регистрами. Листинг программы на языке Arduino с прямым обращением к регистрам порта представлен ниже:

```

1 void setup()
2 {
3   DDRD = 0b00000100; // установка 2-го пина порта D
   в режим вывода
4 }
5 void loop()
6 {
7   PORTD = 0b00000100; // установка логической 1
   во 2-й пин порта D
8   delay(500); // задержка 500 миллисекунд
9   PORTD = 0b00000000; // установка логического 0
   во 2-й пин порта D
10  delay(500); // задержка 500 миллисекунд
11 }

```

В строке 3 записью `DDRD = 0b00000100` настраивается 2-й вывод порта D в режим «ВЫХОД».

Запись в виде `0b00000100` – это форма бинарной записи числа, т.е. в двоичной системе счисления. Так как нумерация осуществляется с младшего бита числа, то 2-й вывод порта D управляется 3-м битом регистра `DDRD`. В который устанавливается лог. «1». В 7-й строке в регистр данных `PORTD` заносится число `0b00000100`, таким образом 3-й бит числа показывает, что во 2-й вывод порта D устанавливается лог. «1». В этот момент на выводе 2 появляется напряжение высокого уровня, соответствующего 5 В. В строке 8 формируется задержка на 500 мс, далее в 9-й строке в регистр данных `PORTD` заносится число `0b00000000`, таким образом в 3-й бит числа устанавливается лог. «0», что во 2-м выводе порта D устанавливает лог. «0» – низкое напряжение, и в 10-й строке формируется задержка на 500 мс. Далее выполнение кода повторяется.

При компиляции данная программа занимает 646 байт. Видим, что при написании кода программы и непосредственном управлении регистрами объем программы получается меньше при том же алгоритме работы.

## 5 Двоичная система счисления

### Использование двоичной системы в микроконтроллере

Как видим в приведенных выше программах, для непосредственного управления состоянием регистров микроконтроллера необходимо обладать знаниями двоичной системы счисления. В данном методическом пособии дается общее представление о двоичной системе счисления. Рекомендуется математические операции в двоичной системе счисления изучить более подробно. Двоичная система представлена на рисунке 5.1.

Двоичная	Десятичная
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
...	...
10000	16

Рисунок 5.1 Представление числа в двоичной системе счисления

Обратите внимание на последовательность и чередование 0 и 1 в двоичные системы счисления. Вы заметили важность степени двойки? Именной с ней связано абсолютно все.

Рассмотрим степень двойки в разных системах счисления в таблице 5.1.

Таблица 5.1 – Степень в двоичной и десятичной системе счисления

$2^n$	Десятичная (DEC)	Двоичная (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

При работе с регистрами порта микроконтроллера будем использовать логические операции, речь о которых пойдет ниже.

## Битовые операции

### Битовое И – логическое умножение

Логическое умножение выполняется оператором & или and. Результаты работы логического умножения представлены в таблице истинности (рисунок 5.2).

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

Рисунок 5.2 – Таблица истинности логического умножения

Основное применение операции И – битовая маска, которая позволяет взять из байта только указанные биты.

Например, к цифровым портам микроконтроллера необходимо подключить цифровые датчики и считать в данный момент работы программы информацию с данных датчиков. Пример работы электрической схемы представлен на рисунке 5.3.

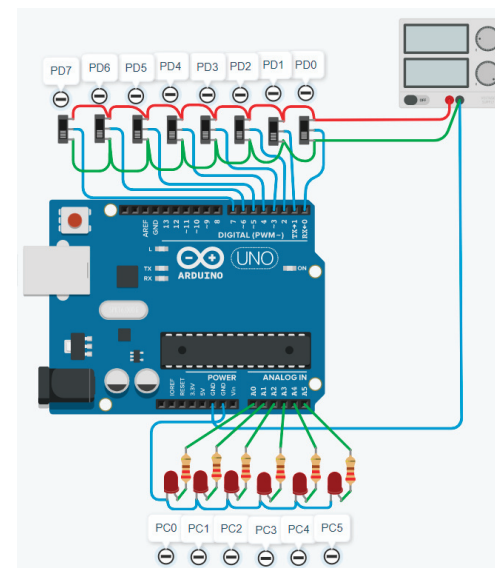


Рисунок 5.3 – Схема для работы с портами микроконтроллера

Электрическая схема состоит из платы разработчика, в которой к каждому цифровому порту (0–7) подключен средний вывод ползункового переключателя, крайние выводы переключателя подключены к потенциалам +5 В и 0 соответственно. Изменяя положение переключателя, можно изменять уровень напряжения на цифровых портах микроконтроллера. Результат битовых операций выводим на порт С микроконтроллера. Для этого к аналоговому выводу А0–А5 платы разработчика подключены светодиоды с резисторами. При подаче логической единицы на пин порта С (высокого потенциала) засвечивается светодиод, подключенный к соответствующему выводу порта.

Листинг программы:

```
1 byte a = 0;
2 void setup() {
3   DDRD = 0b00000000; // установка порта D в режим ввода
4   PORTD = 0b11111111; // устанавливаем пины порта D
   в режим PullUp
5   DDRC = 0b1111111; // установка порта C в режим вывода
6 }
7 void loop() {
8   a = PIND & 0b100001; // наносим битовую маску на 0
   и 5-й бит порта D
9   PORTC = a; // выводим полученный результат
   в PORTC
10 }
```

Результат работы программы представлен на рисунке 5.4.

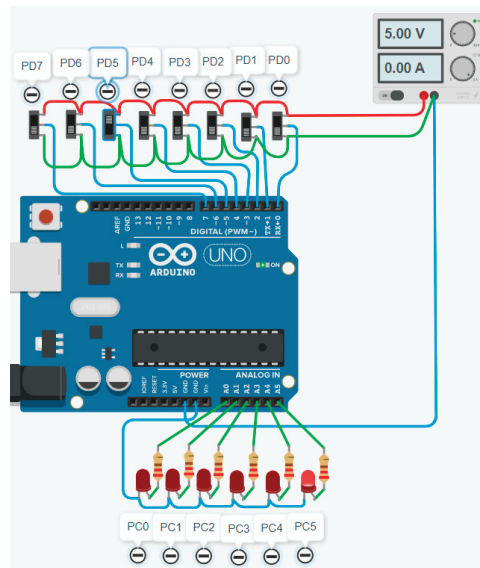


Рисунок 5.4 – Результат работы битовой маски

На выход 5 порта D микроконтроллера подается логическая единица, в программе, представленной на листинге выше, накладывается битовая маска и данные выводятся в порт C. В результате работы программы засвечивается светодиод, подключенный к выводу 5 порта C.

### Битовое ИЛИ

Логическое сложение выполняется оператором `|` или `or`. Результаты работы логического сложения представлены в таблице истинности, приведенной на рисунке 5.5.

0		0	==	0
0		1	==	1
1		0	==	1
1		1	==	1

Рисунок 5.5 – Таблица истинности логического сложения

Основное применение операции И – установка бита в байте.

Например, микроконтроллер управляет силовой нагрузкой, которую необходимо либо включить, либо отключить. Электрическая схема представлена на рисунке 5.6.

Листинг программы:

```
byte a = 0;
void setup() {
  DDRD = 0b00000000; // установка порта D в режим ввода
  PORTD = 0b11111111; // устанавливаем пины порта D в режим PullUp
  DDRC = 0b1111111; // установка порта C в режим вывода
}
void loop() {
  a = PIND | 0b000010; // устанавливаем единицу в 1-й разряд порта C
```

```
PORTC = a;    // выводим полученный результат в PORTC
}
```

Результат логического сложения  $PIND \mid 0b000010$  представлен на рисунке 5.6. При изменении параметров первого пина порта D индикация светодиода, подключенного к 1-му пину порта C, изменяться не будет. Это подтверждает правильность алгоритма работы битового сложения.

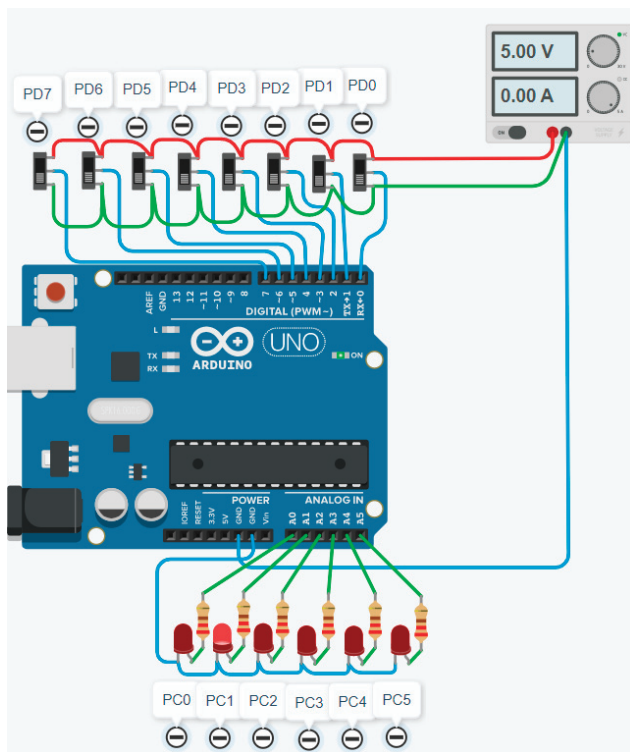


Рисунок 5.6 – Результат работы логического сложения

## Битовое НЕ

Операция логического отрицания (битовое НЕ) выполняется оператором  $\sim$ . Результаты работы логического отрицания представлены в таблице истинности на рисунке 5.7.

$\sim 0$	==	1
$\sim 1$	==	0

Рисунок 5.7 – Таблица истинности логического отрицания

Результат логического отрицания  $\sim PIND$  представлен на рисунке 5.8. На порт D подаем двоичное число  $0b101110$ , на порт C выводим инверсное значение порта D. Правильность работы подтверждается светящимися светодиодами, подключенными к выводам 0 и 4 порта C (рисунок 5.8).

Листинг программы:

```
1 byte a = 0;
2 void setup() {
3   DDRD = 0b00000000; // установка порта D в режим ввода
4   PORTD = 0b11111111; // устанавливаем пины порта D
   в режим PullUp
5   DDRC = 0b1111111; // установка порта C в режим вывода
6 }
7 void loop() {
8   a = ~PIND; // инвертируем данные, поступающие
   с порта D
9   PORTC = a; // выводим полученный результат
   в PORTC
10 }
```

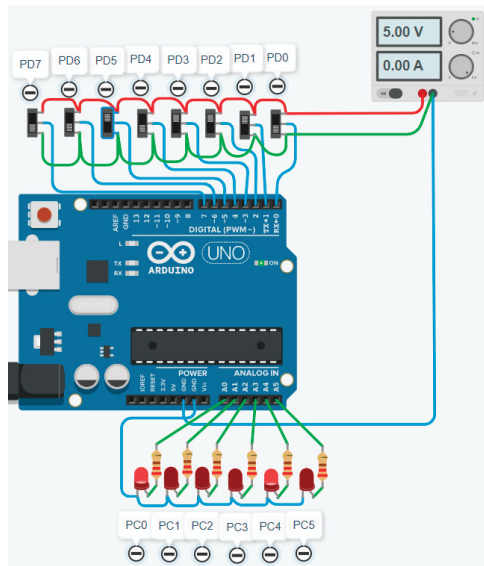


Рисунок 5.8 – Результат работы логического отрицания

### Битовое исключающее ИЛИ

Исключающее ИЛИ (XOR) выполняется оператором  $\wedge$  или хог. Результаты работы исключающего ИЛИ представлены в таблице истинности, приведенной на рисунке 5.9.

0	$\wedge$	0	==	0
0	$\wedge$	1	==	1
1	$\wedge$	0	==	1
1	$\wedge$	1	==	0

Рисунок 5.9 – Таблица истинности исключающего ИЛИ

Основное применение операции исключающего ИЛИ – для инвертирования состояния отдельного бита в байте.

Листинг программы:

```
byte a = 0;
void setup() {
  DDRD = 0b00000000; // установка порта D в режим ввода
  PORTD = 0b11111111; // устанавливаем пины порта D в режим PullUp
  DDRC = 0b11111111; // установка порта C в режим вывода
}
void loop() {
  a = PIND^=0b0000100; // инвертируем данные на 2-м
  // пине порта D
  PORTC = a; // выводим полученный результат в PORTC
}
```

Результат работы программы представлен на рисунке 5.10.

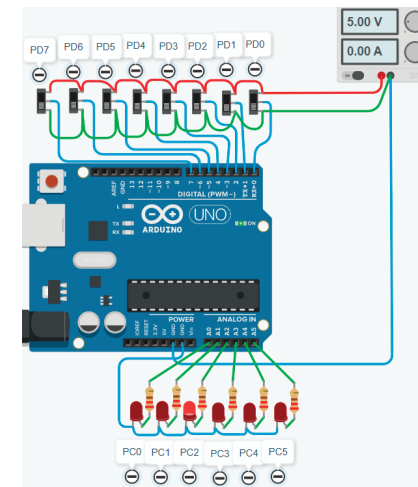


Рисунок 5.10 – Результат работы исключающего ИЛИ

## Битовый сдвиг

Битовый сдвиг позволяет двигать биты вправо или влево, используя операторы `<<` или `>>` соответственно или составные операторы `>>=` и `<<=`. Если при сдвиге биты выйдут за пределы байта, они теряются.

Например, в некоторую переменную занесем двоичное число `0b000011` и осуществим над ним битовый сдвиг влево на 2 позиции. Результат данного сдвига представлен на рисунке 5.11. Таким образом, видно, что при сдвиге влево происходит умножение исходного числа на 2 в степени 2.

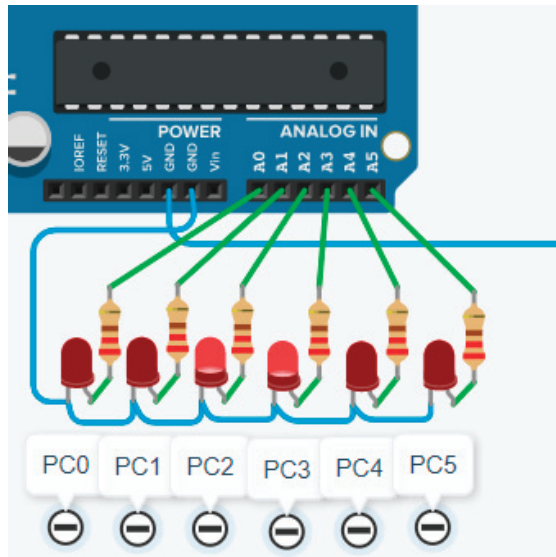


Рисунок 5.11 – Результат работы программы битового сдвига

Таблица 5.2 – Пример битового сдвига влево

Двоичная	Десятичная
<code>0b000011</code>	3
<code>0b000011 &lt;&lt; 2 == 1100</code>	$3 * 4 = 12$

Листинг программы:

```
byte a = 0;
void setup() {
  DDRC = 0b111111; // установка порта C в режим вывода
  a = 0b000011;
  a = a << 2;
  PORTC = a;
}
void loop() {
}
```

Если же сделать битовый сдвиг вправо на 2 единицы числа `0b000011`, то результатом будет число `0b000000` (рисунок 5.12). Смещение битов произошло за пределы байта, поэтому светодиоды не светятся. Для хранения такого значения необходим уже другой тип данных переменной.

Листинг программы:

```
byte a = 0;
void setup() {
  DDRC = 0b111111; // установка порта C в режим вывода
  a = 0b000011;
  a = a >> 2;
  PORTC = a;
}
void loop() {
}
```

Таблица 5.3 – Пример битового сдвига вправо

Двоичная	Десятичная
<code>0b000011</code>	3
<code>0b000011 &gt;&gt; 2 == 0000</code>	$3 / 4 = 0.75$



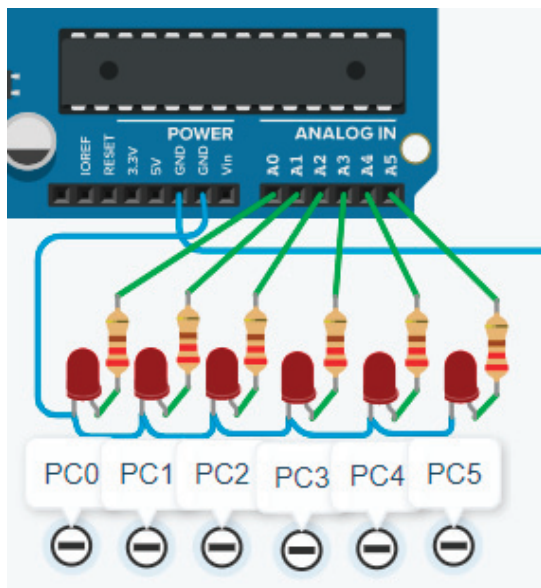


Рисунок 5.12 – Битовый сдвиг вправо

## 6 Управление нагрузкой на портах микроконтроллера

### Устанавливаем логическую «1» в требуемый бит порта

При работе микропроцессора очень часто возникают задачи смены логического сигнала на каком-либо цифровом выходе микроконтроллера. Данный выход может управлять силовой нагрузкой или участвовать в формировании сигнала управления в сложной цепи. Возникает задача, используя битовые операции, установить логическую единицу в требуемый пин порта микроконтроллера.

Если необходимо установить «1» в требуемый бит порта D микроконтроллера, то воспользуемся программой, представленной ниже.

```

1 void setup() {
2   DDRD = 0b11111111; // установка порта D в режим вы-
   вода
3   PORTD = 0b00000000; // устанавливаем «0» в пины порта D
4   PORTD = 0b0000100; // устанавливаем «1» на 2-м пине
   порта D
5 }
6 void loop()
7 {
8 }

```

При компиляции код занимает 454 байт.

В 1-й строке регистром DDRD (регистр направления порта D) выставляем все пины порта D в режим вывода. В 3-й строке выставляем все пины порта в логический «0». В результате этого все светодиоды, подключенные к порту D, будут отключены. В 4-й строке программы в порт D устанавливаем во 2-й пин порта логическую «1». В результате выполнения данной программы светодиод, подключенный ко 2-му пину порта, засветится (рисунок 6.1).

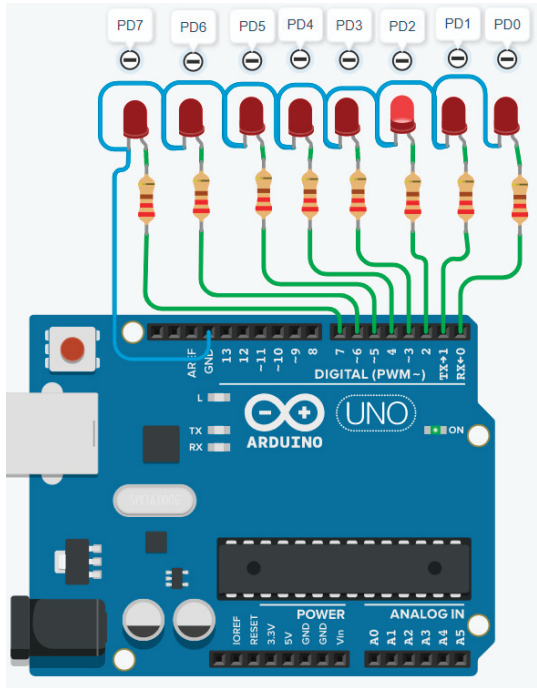


Рисунок 6.1 – Результат работы программы с портом D

В данном случае выполняется прямая запись каждого бита в порт D.

Воспользуемся битовым ИЛИ или логическим сложением для установки нужного бита в требуемый бит порта D. Результаты работы программы представлены на рисунке 6.1.

```

1 void setup() {
2   DDRD = 0b11111111; // установка порта D в режим вы-
   вода
3   PORTD = 0b00000000; // устанавливаем «0» в пины порта D
4   PORTD |= (1 << 2); // устанавливаем «1» на 2-м пине
   порта D
5 }

```

```

6 void loop()
7 {
8 }

```

При компиляции код занимает 452 байт.

В чем преимущество команды `PORTD = 0b0000100` перед `PORTD |= (1 << 2)?` Очевидно, что запись `PORTD |= (1 << 2)` позволяет адресно установить в конкретный пин порта логическую «1». В записи `PORTD = 0b0000100` можно ошибиться и записать логическую «1» не в тот пин. Визуально запись `PORTD |= (1 << 2)` позволяет быстрее определить при чтении программы, к какому пину передается информация. При компиляции двух представленных программ существует разница в объемах занимаемой памяти микроконтроллера: 1-я программа занимает 454 байта, 2-я программа занимает 452 байта. Следовательно, предпочтительнее использовать программу с логическим сложением.

Есть еще один способ установки лог. «1» на требуемом пине порта D. Так как микроконтроллер Atmega328 является основой популярной платы разработчиков Arduino Uno, воспользуемся языком Arduino.

```

1 void setup() {
2   pinMode (2, OUTPUT); // установка 2-го пина порта D
   в режим вывода
3   digitalWrite(2,1); // устанавливаем лог. «1» на 2-м пине
   порта D
4 }
5 void loop()
6 {
7 }

```

При компиляции код занимает 712 байт.

В строке 2 устанавливаем режим работы пина порта 2 на вывод, в строке 3 устанавливаем лог. «1» на 2-м пине порта. Результат работы программы представлен на рисунке 6.1. Если посмотреть на листинг программы, то программа, написанная данным языком, проста в понимании, но при компиляции программа занимает 712 байт. Это по объему в 1,5 раза

больше, чем программа, которая работает напрямую с регистрами. Но в чем же преимущества написания кода языком Arduino? Например, при смене платформы не нужно будет переписывать обращение к регистрам, достаточно будет поменять платформу перед компиляцией программы (например, сменить Arduino Uno на Arduino Mega).

## Устанавливаем логическую «1» в требуемые пины порта

Микроконтроллер часто управляет несколькими нагрузками. Рассмотрим вариант установки лог. «1» в нескольких пинах порта D микроконтроллера.

```

1 void setup() {
2   DDRD = 0b11111111; // установка порта D в режим вы-
   вода
3   PORTD = 0b00000000; // устанавливаем «0» в пины пор-
   та D
4   PORTD = 0b10000101; // устанавливаем «1» на 0-м, 2-м и
   7-м пинах порта D
5 }
6 void loop()
7 {
8 }
```

При компиляции код занимает 454 байта.

В строке 4 листинга программы устанавливается лог. «1» на 0-м, 2-м и 7-м пинах порта D. Объем занимаемой памяти – 454 байта. Результат работы программы представлен на рисунке 6.2.

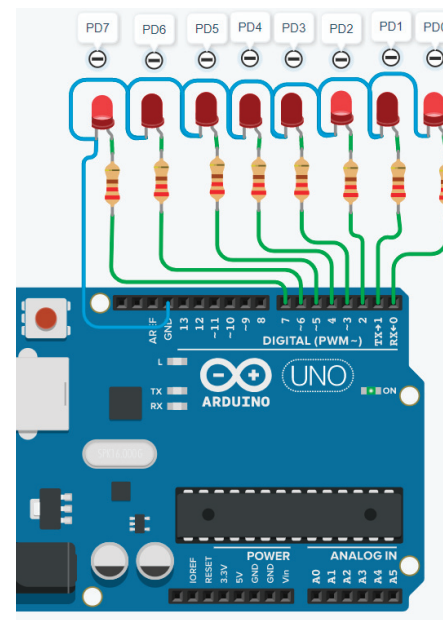


Рисунок 6.2 – Результат работы программы с портом D

Воспользуемся операцией логического сложения, для этого рассмотрим программу, представленную ниже:

```

1 void setup() {
2   DDRD = 0b11111111; // установка порта D в режим вы-
   вода
3   PORTD = 0b00000000; // устанавливаем «0» в пины пор-
   та D
4   PORTD |= (1 << 0) | (1 << 2) | (1 << 7); // устанавливаем
   «1» на 0-м, 2-м и 7-м пинах порта D
5 }
6 void loop()
7 {
8 }
```

При компиляции код занимает 456 байт.

Результат работы программы представлен на рисунке 6.2. В данном варианте написания сохраняется легкость определения номера пина порта D установки лог. «1». Но объемом памяти данная программа занимает на 2 байта больше.

Рассмотрим реализацию поставленной задачи языком Arduino. Листинг программы представлен ниже:

```

1 void setup() {
2   pinMode (0, OUTPUT); // установка 0-го пина порта D
   в режим вывода
3   pinMode (2, OUTPUT); // установка 2-го пина порта D
   в режим вывода
4   pinMode (7, OUTPUT); // установка 7-го пина порта D
   в режим вывода
5   digitalWrite(0,1); // устанавливаем лог. «1» на 0-м пине
   порта D
6   digitalWrite(2,1); // устанавливаем лог. «1» на 2-м пине
   порта D
7   digitalWrite(7,1); // устанавливаем лог. «1» на 7-м пине
   порта D
8 }
9 void loop()
10 {
11 }
```

Необходимо каждому пину порта D прописать режим работы, а также каждый требуемый пин (0, 2, 7) выставить в лог. «1». Программа занимает 764 байта.

### Устанавливаем логический «0» в требуемый пин порта

Естественно, нагрузку необходимо отключить или логический сигнал необходимо перевести в лог. «0». В этом нам помогут операции логического умножения  $\&=$  и логического отрицания  $\sim$ .

Если необходимо установить «0» в требуемый бит порта D микроконтроллера, для этого используем программу, представленную ниже:

```

1 void setup() {
2   DDRD = 0b11111111; // установка порта D в режим вы-
   вода
3   PORTD = 0b11111111; // устанавливаем «1» в пины пор-
   та D
4   PORTD = 0b11111011; // устанавливаем «0» на 2-м пине
   порта D
5 }
6 void loop()
7 {
8 }
```

При компиляции код занимает 454 байта.

Во 2-й строке регистром DDRD (регистр направления порта D) выставляем все пины порта D в режим вывода. В 3-й строке выставляем все пины порта D в лог. «1». В результате этого все светодиоды, подключенные к порту D, будут включены. В 4-й строке программы в порт D устанавливаем во 2-й пин порта логический «0». В результате выполнения данной программы светодиод, подключенный ко 2-му пину порта, потухнет (рисунок 6.3).

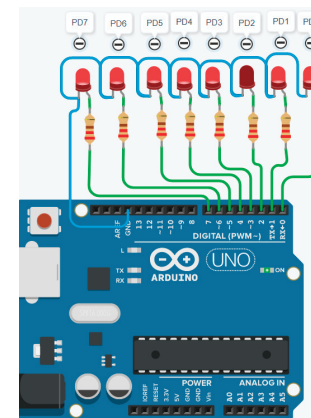


Рисунок 6.3 – Результат работы программы с портом D

Воспользуемся битовым ИЛИ или логическим сложением для установки нужного бита в требуемый бит порта D. Результаты работы программы представлены на рисунке 6.3.

Листинг программы:

```
1 void setup() {
2   DDRD = 0b11111111; // установка порта D в режим вы-
   вода
3   PORTD = 0b11111111; // устанавливаем «1» в пины пор-
   та D
4   PORTD = ~(1 << 2); // устанавливаем «0» на 2-м пине
   порта D
5 }
6 void loop()
7 {
8 }
```

При компиляции код занимает 454 байта.

Запись `PORTD = ~(1 << 2)` позволяет адресно установить в конкретный пин порта логический «0». При компиляции двух представленных программ разница в объемах занимаемой памяти микроконтроллера отсутствует, обе программы занимают 454 байта FLASH-памяти. Результат работы программы представлен на рисунке 6.3.

Устанавливаем логический «0» в требуемые пины порта

Микроконтроллеру необходимо при управлении силовой нагрузкой параллельно отключать сразу несколько пинов порта. Реализация данной задачи языком Arduino представлена в листинге программы:

```
1 void setup() {
2   pinMode (0, OUTPUT); // установка 0-го пина порта D
   в режим вывода
3   pinMode (1, OUTPUT); // установка 1-го пина порта D
   в режим вывода
4   pinMode (2, OUTPUT); // установка 2-го пина порта D
   в режим вывода
```

```
5   pinMode (3, OUTPUT); // установка 3-го пина порта D
   в режим вывода
6   pinMode (4, OUTPUT); // установка 4-го пина порта D
   в режим вывода
7   pinMode (5, OUTPUT); // установка 5-го пина порта D
   в режим вывода
8   pinMode (6, OUTPUT); // установка 6-го пина порта D
   в режим вывода
9   pinMode (7, OUTPUT); // установка 7-го пина порта D
   в режим вывода
10  digitalWrite(0,0); // устанавливаем лог. «0»
   на 0-м пине порта D
11  digitalWrite(1,1); // устанавливаем лог. «1»
   на 1-м пине порта D
12  digitalWrite(2,0); // устанавливаем лог. «0»
   на 2-м пине порта D
13  digitalWrite(3,1); // устанавливаем лог. «1»
   на 3-м пине порта D
14  digitalWrite(4,1); // устанавливаем лог. «1»
   на 4-м пине порта D
15  digitalWrite(5,1); // устанавливаем лог. «1»
   на 5-м пине порта D
16  digitalWrite(6,1); // устанавливаем лог. «1»
   на 6-м пине порта D
17  digitalWrite(7,0); // устанавливаем лог. «0»
   на 7-м пине порта D
19 }
20 void loop()
21 {
22 }
```

В строках 2–9 настраиваем пины порта в режим выхода (OUTPUT) и затем строками 10–17 устанавливаем необходимый пин порта в требуемое логическое состояние. Таким образом, на пинах 0, 2 и 7 будет установлен лог. «0», светодиоды, подключенные к данным портам, светиться не будут. Результат работы программы представлен на рисунке 6.4.

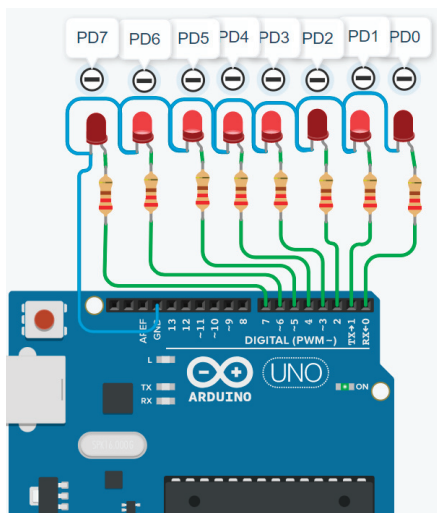


Рисунок 6.4 – Результат работы программы с портом D

Для реализации данной задачи для прямого управления портом через регистры воспользуемся несколькими битовыми операциями: сложение, отрицание и битовый сдвиг влево.

Листинг программы прямого управления регистрами порта:

```

1 void setup() {
2   DDRD = 0b11111111; // установка порта D в режим вы-
   вода
3   PORTD = 0b11111111; // устанавливаем лог. «1» в пины
   порта D
4   PORTD &= ~( (1 << 0) | (1 << 5) ); // устанавливаем лог.
   «0» на 2-м пине порта D
5 }
6 void loop()
7 {
8 }
```

Во 2-й строчке программы настраиваем порт D микроконтроллера Atmega 328 на выход, установив лог. «1» во все биты регистра DDRD. Команда `PORTD &= ~( (1 << 0) | (1 << 5) )` выполняется последовательно согласно приоритетам логических операций. Сначала выполняется то, что в скобках, затем выполняется операция логического сложения и затем – операция инверсии. В результате выполнения в пины 0 и 5 порта D устанавливается лог. «0». Результат выполнения команды представлена на рисунке 6.4.

### Проверка логических «1» или «0» в требуемом пине порта

Микроконтроллер не только управляет силовой нагрузкой, он также контролирует параметры с помощью цифровых датчиков, которые могут быть подключены к его цифровым портам. Таким образом, микроконтроллеру необходимо уметь проверять наличие логического «0» или «1» на выводах его порта.

Воспользуемся битовыми операциями на микроконтроллере, а именно битового сдвига влево (<<) и логического умножения (&).

Допустим, считав состояние в цифровой порт D, получили число 0b01010101. Необходимо проверить 2-й бит регистра на наличие в нем логической «1». Для этого создаем маску на порт с помощью битового сдвига (1<<2), таким образом формируется число 0b00000100. Затем накладываем данную битовую маску на данные, которые считали с порта, используя при этом логическое умножение (&):

```

0b01010101
&
0b00000100
-----
0b00000100
```

Получим число 0b00000100, в десятичной системе это 4. В двоичной системе для операции сравнения все, что больше 0, – это 1. Таким образом, уверенно можем сказать, что на 2-м выходе порта – логическая единица. Следовательно, для фиксации на 2-м выводе порта D микроконтроллера Atmega328 будем использовать следующее выражение:

$$\text{PIND} \& (1 \ll 2)$$

Если, считав с порта D, получили комбинацию цифр, в которой на 2-м бите находился логический «0», то при наложении маски:

```

0b01010001
&
0b00000100
-----
0b00000000

```

получим логический «0». Таким образом, используя битовые операции, можно детектировать состояния необходимого бита порта.

Листинг программы представлен ниже:

```

1 void setup() {
2   DDRD = 0b00000000; // установка 0–7-го пинов порта D
  в режим ввода
3   DDRC = 0b1111111; // установка 0–5-го пинов порта C
  в режим вывода
4 }
5 void loop() {
6   if (PIND & (1 << 2)) {
7     PORTC |= (1 << 0);
8   } else {
9     PORTC &= !(1 << 0);
10  }
11 }

```

Результат работы программы представлен на рисунке 6.5.

При проверке наличия логического 0 на выводе порта используем операцию логической инверсии ! при проверке логической «1», т.е. !(PIND & (1 << 2)).

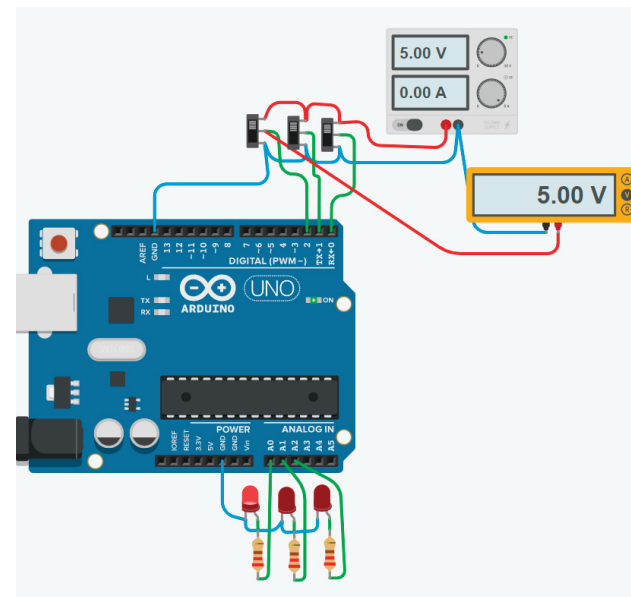


Рисунок 6.5 – Схема экспериментальной установки

Листинг программы представлен ниже:

```

1 void setup() {
2   DDRD = 0b00000000; // установка 0–7-го пинов порта D
  в режим ввода
3   DDRC = 0b1111111; // установка 0–5-го пинов порта C
  в режим вывода
4 }
5 void loop() {
6   if (!(PIND & (1 << 2))) {
7     PORTC |= (1 << 0);
8   } else {

```

```

9  PORTC &= !(1 << 0);
10 }
11 }

```

Результат работы программы представлен на рисунке 6.6.

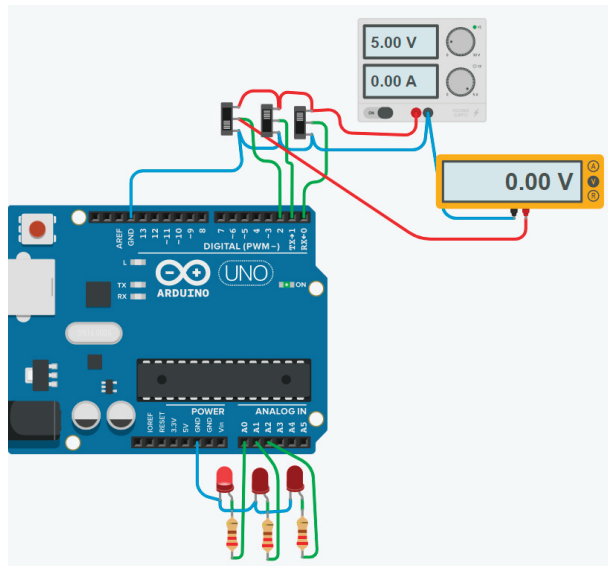


Рисунок 6.6 – Схема экспериментальной установки

## 7 Подключение тактовой кнопки к микроконтроллеру

### Схемы подключения тактовых кнопок

В предыдущих разделах рассмотрели, как считать с пина порта логический «0» или «1». Рассмотрим примеры, как подключить замыкающие контакты (ключи) к микроконтроллеру и как настроить его конфигурацию. Существует несколько способов подключения ключей к микроконтроллеру, которые представлены на рисунке 7.1. Схема на рисунке 7.1, а потребляет питание только в том случае, когда контакт К1 замкнут, при разомкнутом контакте схема не потребляет энергию, в отличие от схемы, представленной на рисунке 7.1, б.

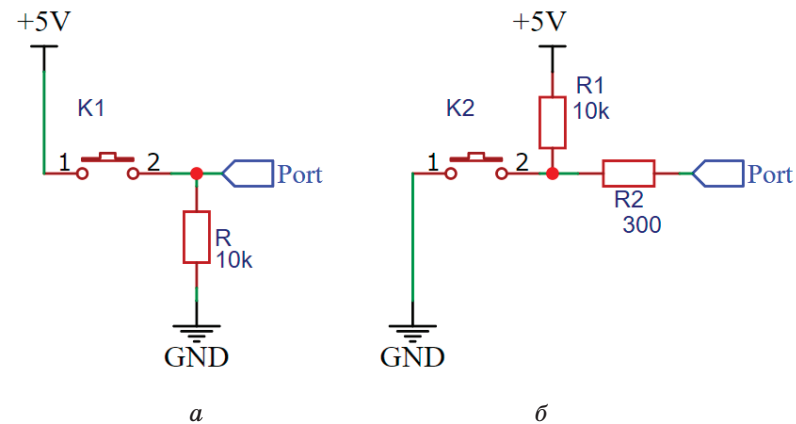


Рисунок 7.1 – Схемы подключения замыкающего контакта

Схема, приведенная на рисунке 7.1, б, содержит больше компонентов, так как несмотря на то, что в современных микроконтроллерах есть внутренние подтягивающие резисторы, необходимо предусмотреть резистор R2. Он будет служить



токоограничивающим резистором при нажатой кнопке К2. Величина сопротивления резистора R2 определяется токовым ограничением порта микроконтроллера.

Рассмотрим схему, представленную на рисунке 7.1, а. Когда контакт К1 не замкнут, вывод Port подключен через резистор R к низкому потенциалу GND и на выводе Port находится логический «0». Если замкнуть контакт К1, то ток будет протекать через него и попадать в вывод Port, который подключается к цифровому порту микроконтроллера. Если порт настроен на вход, то величина входного сопротивления намного меньше, чем у резистора R. Таким образом, на выводе Port появится напряжение, соответствующее уровню логической «1» (рисунок 7.2).

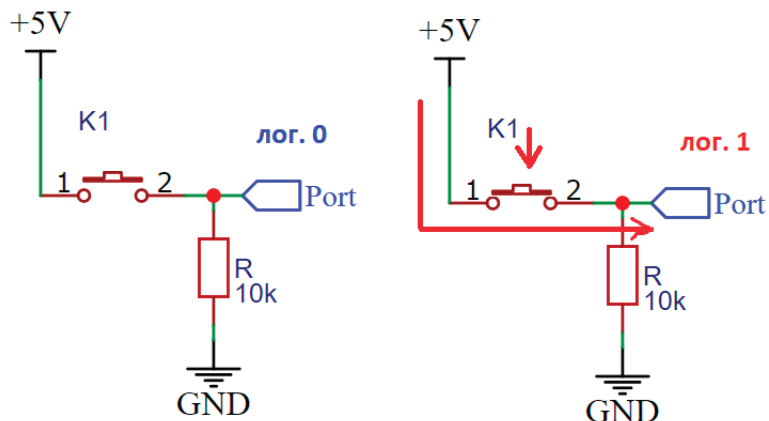


Рисунок 7.2 – Принцип образования логического уровня сигнала для схемы А

Рассмотрим схему, представленную на рисунке 7.1, б. Когда контакт К2 не замкнут, вывод Port подключен через резистор R к высокому потенциалу +5 и на выводе Port находится логическая «1». Если замкнуть контакт К2, то на выводе 2 контакта К2 появится низкий уровень потенциала, таким образом, ток не будет протекать через вывод Port, таким образом появится логический «0» (рисунок 7.3).

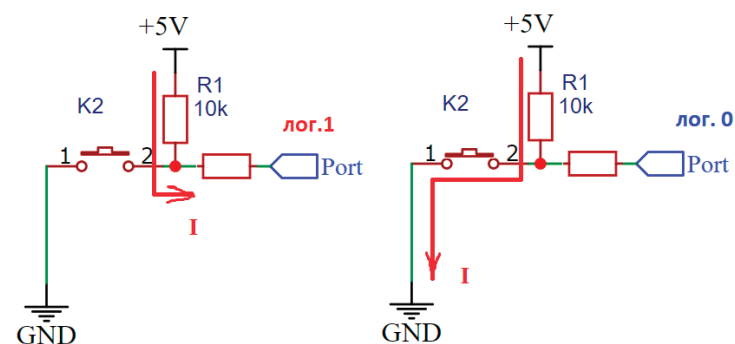


Рисунок 7.3 – Принцип образования логического уровня сигнала для схемы Б

Протестируем данные схемы подключения на микроконтроллере. Для этого используем листинг программы, представленный ниже:

```

1 void setup() {
2   DDRD = 0b00000000; // установка 0–7-го пинов порта D
   в режим ввода
3   DDRC = 0b111111; // установка 0–5-го пинов порта C
   в режим вывода
4 }
5 void loop() {
6   if (!(PIND & (1 << 4))) {
7     PORTC |= (1 << 0);
8   } else {
9     PORTC &= !(1 << 0);
10  }
11 }

```

При появлении низкого уровня сигнала на пине 4 порта D будет подаваться логическая «1» на 0-й порт C, светодиод в данном случае будет светиться. При появлении логической «1» на 4-м пине порта D светодиод, подключенный к 0-му порту C, не светится. Результат работы программы представлен на рисунке 7.4.

В качестве переключателя используем кнопку. Схема и внешний вид кнопки представлен на рисунке 7.5. Пары контактов 1 и 2, 3 и 4 кнопки соединены между собой, будьте внимательны при подключении.

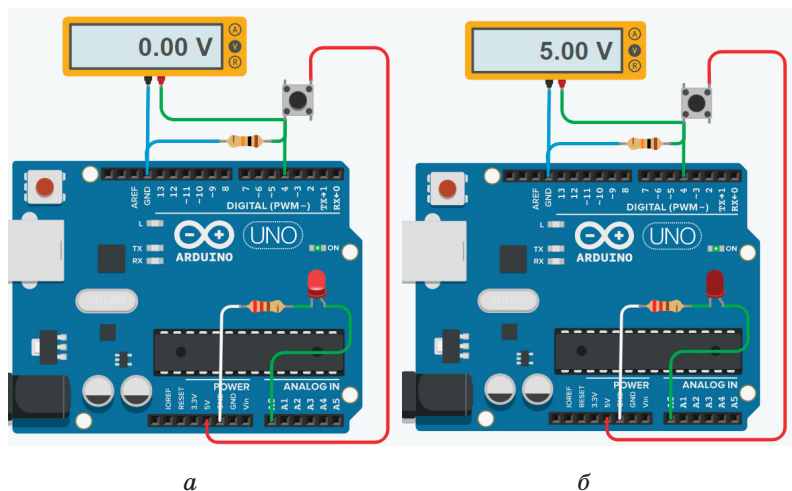


Рисунок 7.4 – Результат работы программы (схема см. рисунок 7.1, а): а – кнопка не нажата; б – кнопка нажата

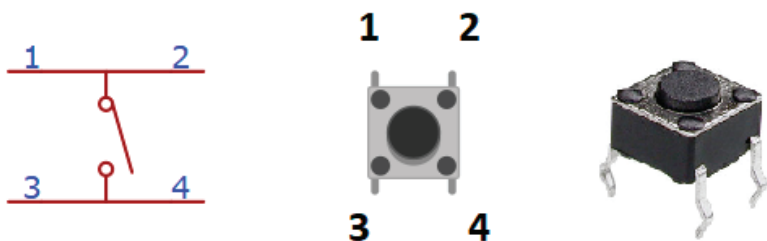


Рисунок 7.5 – Схема и внешний вид тактовой кнопки

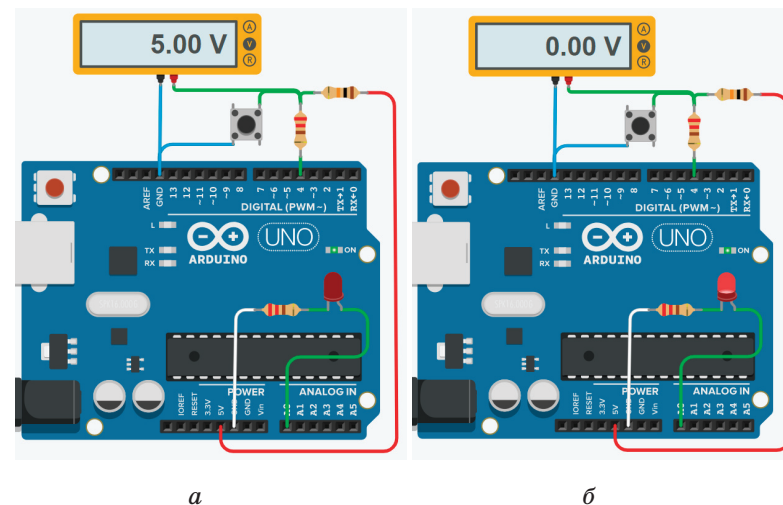


Рисунок 7.6 – Результат работы программы (схема см. рисунок 7.1, б): а – кнопка не нажата; б – кнопка нажата

Однократное нажатие кнопки или изменение состояния сигнала управляемого порта реализуется очень просто, но если усложнить задачу? Например, необходимо при нажатии 3 раза на кнопку засветить светодиодом, а если нажать еще 3 раза на кнопку, то погасить его. Реализуем данный алгоритм, используя схему, представленную на рисунке 7.4. Представим алгоритм в виде блок-схемы на рисунке 7.7.

Вводим переменную *but*, отвечающую за количество нажатий на кнопку, и обнуляем ее. Проверяем, нажата ли кнопка, если нажата, то увеличиваем переменную *but* на единицу. Далее проверяем: переменная *but* = 3, т.е. было ли совершено 3 нажатия на кнопку, если да, то включаем светодиод, переходим к проверке переменной *but*, если она равна 6, то выключаем светодиод, обнуляем переменную *but* и повторяем алгоритм заново, если нет, возвращаемся к проверке нажатия кнопки.

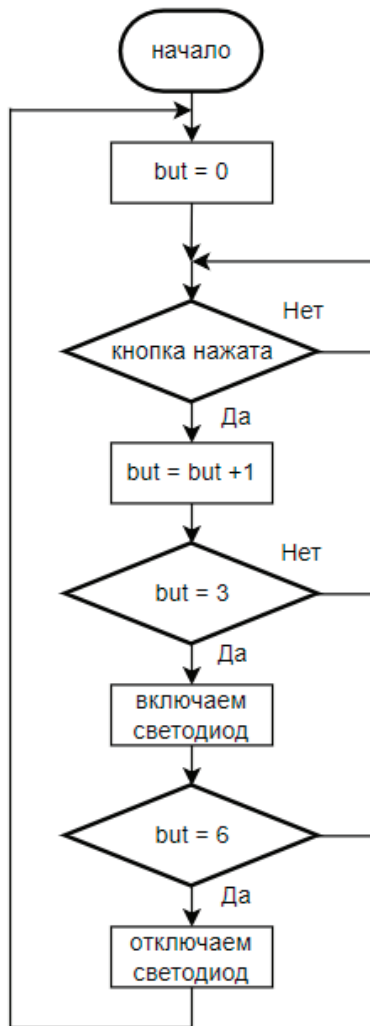


Рисунок 7.7 – Блок-схема алгоритма

Листинг программы, реализующей описанный выше алгоритм, написанный языком Arduino, представлен ниже:

```

1 byte but = 0;
2 void setup()
3 {
4   pinMode(4, INPUT); // установка 4-го пина порта D
   в режим ввода
5   pinMode(14, OUTPUT); // установка 0-го пина порта C
   в режим вывода
6 }
7 void loop()
8 {
9   if (digitalRead(4))
10  {
11    but++;
12    if (but == 3)
13    {
14      digitalWrite(14,1);
15    }
16    if (but == 6)
17    {
18      digitalWrite(14,0);
19      but = 0;
20    }
21  }
22 }
  
```

Загрузив данный код в микроконтроллер и нажав на кнопку, мы поймем, что программа работает некорректно, хотя алгоритм правильный. В этом можно убедиться, если интегрировать в код вывод в монитор последовательного порта информацию о состоянии переменной but. Листинг программы представлен ниже:

```

1 byte but = 0;
2 void setup()
3 {
4   Serial.begin(9600);
5   pinMode(4, INPUT); // установка 4-го пина порта D в ре-
   жим ввода
6   pinMode(14, OUTPUT); // установка 0-го пина порта C
   в режим вывода
  
```

```

7  }
8  void loop()
9  {
10 if (digitalRead(4))
11 {
12   but++;
13   Serial.println(but);
14   if (but == 3)
15   {
16     digitalWrite(14,1);
17   }
18   if (but == 6)
19   {
20     digitalWrite(14,0);
21     but =0;
22   }
23 }
24 }

```

Один клик на кнопку, и в мониторе последовательного порта выводятся значения количества нажатий на тактовую кнопку, результаты выгрузки монитора последовательного порта представлены на рисунке 7.8.

На рисунке видно, что из-за быстрого действия микроконтроллера программа успевает многократно считать нажатие кнопки, хотя для нас произошел один клик. Решить поставленную задачу поможет задержка после инкрементирования переменной but в 13-й строке, листинг программы представлен ниже:

```

1  byte but = 0;
2  void setup()
3  {
4  Serial.begin(9600);
5  pinMode(4, INPUT); // установка 4-го пина порта D
   в режим ввода
6  pinMode(14, OUTPUT); // установка 0-го пина порта C
   в режим вывода
7  }

```

```

8  void loop()
9  {
10 if (digitalRead(4))
11 {
12   but++;
13   delay(500);
14   Serial.println(but);
15   if (but == 3)
16   {
17     digitalWrite(14,1);
18   }
19   if (but == 6)
20   {
21     digitalWrite(14,0);
22     but =0;
23   }
24 }
25 }

```

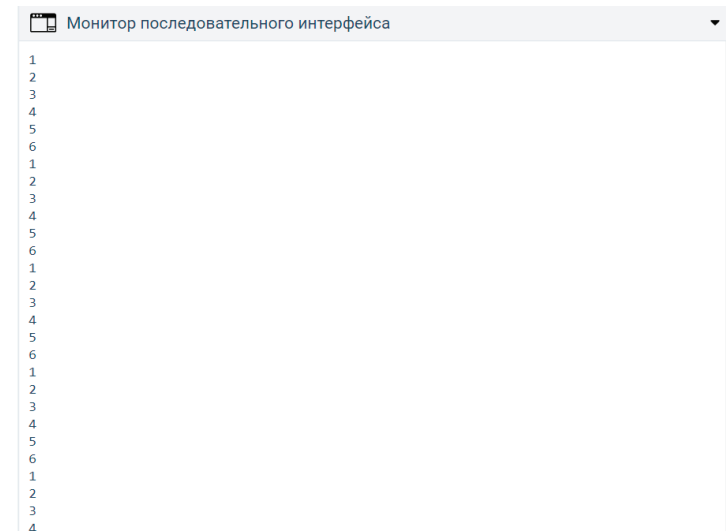


Рисунок 7.8 – Изменения переменной but при однократном клике на кнопку

Использование функции delay() является не лучшим решением, особенно если микроконтроллер работает над несколькими параллельно протекающими процессами. Как избавиться от функции delay(), рассмотрим в разделе с таймерами.

Для ввода информации используют клавиатуру, например представленную на рисунке 7.9. После рассмотрения способов опроса кнопки, описанных выше, возникает желание использовать 12 портов микроконтроллера, но данная идея очень расточительна по отношению к ресурсу микроконтроллера. Рассмотрев внутреннюю схему клавиатуры, представленной на рисунке 7.10, можно сократить количество используемых портов до 7.

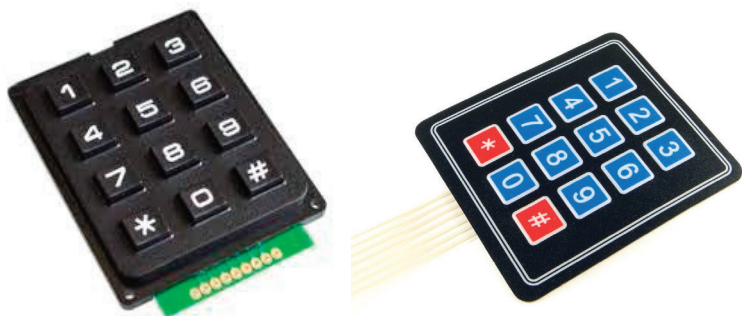


Рисунок 7.9 – Внешний вид клавиатур на 12 кнопок

Видим, что кнопки имеют общую шину по строкам и по столбцам, и это можно использовать для детектирования, какая кнопка нажата в данный момент. Например, если подавать питание поочередно на столбцы, то при замыкании какой-либо кнопки будет получать питание строчка, в результате чего на выходе будем получать цифровой код нажатой кнопки. Все вышесказанное легко представить в виде таблицы работы клавиатуры (таблица 7.1)

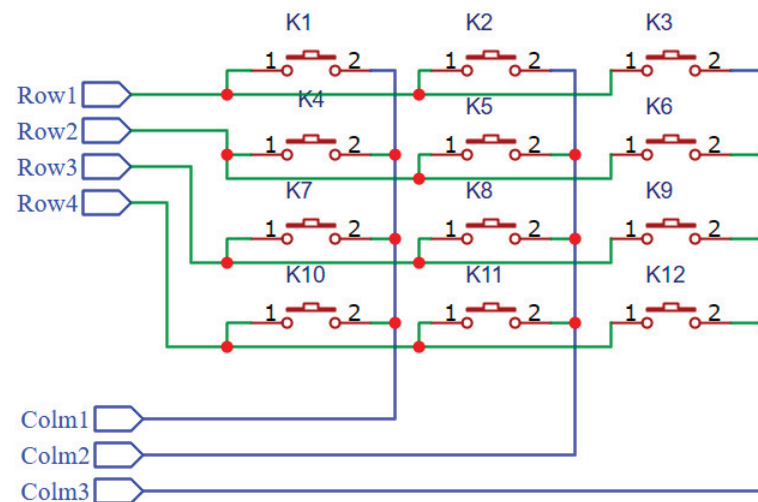


Рисунок 7.10 – Схема клавиатуры

Таблица 7.1 – Таблица работы клавиатуры

Нажата кнопка	K1	K4	K7	K10
Подаем питание на Colm1	1000	0100	0010	0001
	K2	K5	K8	K11
Подаем питание на Colm2	1000	0100	0010	0001
	K3	K6	K9	K12
Подаем питание на Colm3	1000	0100	0010	0001

Таким образом, мы сможем идентифицировать нажатую кнопку на матричной клавиатуре. В этом примере нельзя одновременно нажимать несколько кнопок, так как это приведет к неправильному считыванию [3].

## Дребезг контакта

Это явление в электромеханических коммутационных устройствах, наблюдаемое в момент замыкания контактов,

когда происходят многократные неконтролируемые замыкания и размыкания контактов за счет упругости материалов и деталей контактной системы – некоторое время контакты отскакивают друг от друга при соударениях, размыкая и замыкая электрическую цепь [4]. Идеальная коммутация нажатия кнопки представлена на рисунке 7.11, а. Данный рисунок получен в результате коммутации кнопки в виртуальной среде Tinkercad. Видно четкий фронт сигнала без каких-либо помех, причем фронт один. Осциллограмма коммутации реальной кнопки представлена на рисунке 7.11, б. Здесь мы видим многократное срабатывание кнопки, наличие помех, что, естественно, может негативно сказаться на процессе анализа нажатия кнопки микроконтроллером.

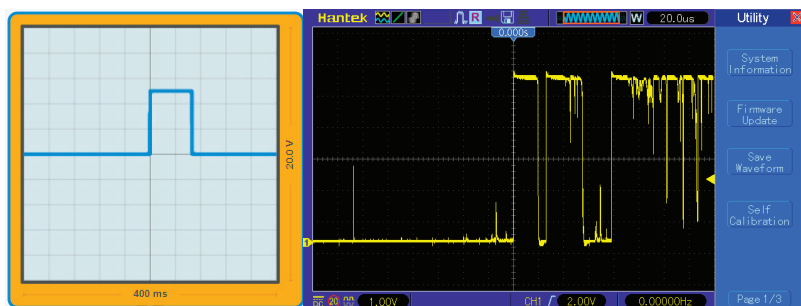


Рисунок 7.11 – Осциллограммы коммутации:  
а – идеальной; б – реальной

Избежать отрицательного воздействия дребезга контакта можно различными способами. Можно выделить два: аппаратный и программный способ. Аппаратный способ заключается в том, чтобы посредством подключения внешних цепей, например RC-цепочки, воздействовать на коммутационный режим, уменьшая помехи, т.е. использовать как фильтр. Программное решение заключается в том, чтобы создать небольшую задержку при нажатии кнопки – не более 100 миллисекунд. Листинг программы с программным решением антидребезга представлен ниже:

```

1 uint32_t butt_Timer = 0;
2 void setup() {
3   Serial.begin(9600);
4   PORTD &= ~(1 << 4); // устанавливаем 4-й пин порта D
   на выход
5 }
6 byte flag = 0; //обнуляем флаги
7 byte state = 0; //обнуляем состояния кнопок
8 void loop() {
9   state = ((PIND & (1 << 4)) >> 4); // записали
   состояние в 0-й бит state
10  if (((state & (1 << 0)) & !(flag & (1 << 0))) && (millis() -
   butt_Timer > 100)) // обработчик нажатия
11  {
12    flag |= (1 << 0); // устанавливаем flag 1
13    butt_Timer = millis();
14    Serial.println("PUSH DOWN");
15  }
16  if (!(((state & (1 << 0)) & (flag & (1 << 0)))) && (millis() -
   butt_Timer > 100)) { // обработчик отпускания
17    flag &= ~(1 << 0); // устанавливаем 0-й
   бит flag лог. «0»
18    butt_Timer = millis();
19    // Serial.println("PUSH UP");
20  }
21 }

```

Работа с таймерами и работа с протоколами передачи данных будет описана в следующих разделах.

В 1-й строке программы создаем переменную butt\_Timer для отслеживания переполнения таймера, в 3-й строке активируем протокол передачи данных UART для вывода информации в монитор порта. Использование монитора порта позволит протестировать программы на интересующем нас этапе работы программы. В 4-й строчке активизируем пин порта 4 работать на получение информации извне. В 6-й строке вводим переменную flag типа byte, которая будет хранить флаги состояния кнопок. Текущее состояние кнопок будет храниться в байтах переменной state. В 9-й строке считываем значе-

ние с пина 4 и записываем его посредством битового сдвига в 0-й бит числа state. В 10-й строке сверяем текущее состояние  $((state \& (1 \ll 0)) \& !(flag \& (1 \ll 0)))$  0-го бита числа state с 0-м битом числа flag, которое хранит предыдущее состояние кнопки. Кроме того, проверяется, переполнен ли таймер нажатия кнопки  $(millis() - butt\_Timer > 100)$  для задержки срабатывания нажатия. Это делается для того, чтобы отсеять помехи при срабатывании и принять сигнал, когда кнопка точно нажата. При выполнении двух условий заходим в обработчик нажатия, в 12-й строке флаг меняем на противоположный и в 13-й строке записываем время, когда произошло срабатывание кнопки, в переменную butt\_Timer для отслеживания следующего срабатывания таймера. В 14-й строке выводим в монитор порта фразу PUSH DOWN. В 16-й строке проверяем по описанному выше алгоритму, отжата ли кнопка, и если она отжата, в 17-й строке меняем состояние флага. Таким образом мы избегаем дребезга контакта и в процессе эксплуатации.

## 8 Таймер – работа со временем

В описанных выше программах и алгоритмах использовали функцию delay(), которая останавливает выполнение работы микроконтроллера на время, указанное внутри функции. Данная задержка по времени во многих решениях неприемлема, особенно если микроконтроллер выполняет параллельно несколько задач или отслеживает данные с разных датчиков. В этом разделе решим задачу с многозадачностью микроконтроллера и рассмотрим возможность замены функции delay() другими решениями.

Алгоритм работы таймера можно представить в виде простой схемы, представленной на рисунке 8.1. Пусть нам из пункта А необходимо добраться до пункта Б за 5 минут. Находясь в пункте А, фиксируем время и следуем по маршруту, мы проверяем часы и отнимаем от текущего времени то, которое запомнили в начале пути. Таким образом, перемещаясь по маршруту, будем понимать, какое время мы в дороге. При этом человеку, который двигается из пункта А в пункт Б, ничего не мешает выполнять другие функции, например пить воду и съесть бутерброд. Так и микроконтроллер при своей работе может, контролируя время, выполнять несколько задач.



Рисунок 8.1 – Алгоритм таймера на millis()

Выполнить данный алгоритм в языке программирования Arduino поможет функция millis().

## Функция millis()

Функция возвращает количество микросекунд с момента запуска текущей программы на плате Arduino. Это число переполнится (вернется к нулю) примерно через 70 минут. Тип данных: unsigned long.

Реализуем мигание светодиода с использованием функции millis() по алгоритму, представленному на рисунке 8.1. Листинг программы представлен ниже:

```
1 uint32_t timer =0;
2 #define T_PERIOD 500 //период переключения
3 void setup()
4 {
5   DDRD |= (1<<7); // установили 7-й пин порта D на выход
6 }
7 void loop ()
8 {
9   if ((millis()-timer)>= T_PERIOD)
10  {
11    timer = millis(); //сброс
12    if ( PIND&(1<<7))
13    {
14      PORTD &= ~(1<<7);
15    }
16    else
17    {
18      PORTD |= (1<<7);
19    }
20 }
21 }
```

В 1-й и 2-й строках программы мы объявляем и обнуляем переменную timer1 и timer2 типа uint32\_t – это беззнаковый int, который занимает 32 бита, для хранения данных возвращаемой функцией millis(). В 3-й и 4-й строках формируем с помощью функции #define автозамену слова T\_PERIOD1 на число 100 и T\_PERIOD2 на число 500 при компиляции программы. Данные числа отвечают за

длительность импульса. В 6-й строке DDRD |= (1 << 7)|(1 << 4) устанавливаем режим вывода на 7-й и 4-й пины порта D.

В бесконечном цикле loop() вызываются пользовательские функции mig\_led1(), а затем mig\_led2(). Принцип их аналогичен, поэтому привожу описание только для одной пользовательской функции mig\_led1().

При вызове данной функции программа переходит в 16-ю строку, и начинает выполняться подпрограмма, которая описана в 17-й и 24-й строках. В 17-й строке проверяем условие if ((millis() – timer1) >= T\_PERIOD1), т.е. если разница между текущим временем и зафиксированное временем будет больше T\_PERIOD1, значит произошло событие, т.е. скобка стала равна true, тогда попадаем в тело условия if. В строке 18 сбрасываем зафиксированное время timer1 = millis(), это делается для следующей обработки. В 19-й строке считываем состояние порта D PIND & (1 << 7)), если состояние порта будет равно 1, то меняем его на противоположный в 20-й строке PORTD &= ~(1<<7), если же состояние порта равно 0, то в 22-й строке меняем его на противоположный PORTD |= (1<<7), выставляем лог. «1». Далее операции повторяются заново. Для функции mig\_led2() алгоритм работы аналогичен, исключением является лишь настройка на другие порты и таймер.

Реализация же поставленной задачи с использованием функции millis() представлена в ответе в виде листинга программы.

Листинг программы:

```
1 uint32_t timer1 = 0;
2 uint32_t timer2 = 0;
3 #define T_PERIOD1 100 //период переключения
4 #define T_PERIOD2 500 //период переключения
5 void setup() {
6   DDRD |= (1 << 7)|(1 << 4); // установили 7-й, 4-й пины
   порта D на выход
7 }
8 void loop() {
9   //*****мигаем 7-м портом
   *****
```



```

10 mig_led1();
11 //*****
12 //*****мигаем 4-м портом
13 mig_led2();
14 //*****
15 }
16 void mig_led1() {
17 if ((millis() - timer1) >= T_PERIOD1) {
18 timer1 = millis(); //сброс
19 if (PIND & (1 << 7)) {
20 PORTD &= ~(1 << 7);
21 } else {
22 PORTD |= (1 << 7);
23 }
24 }
25 }
26 void mig_led2() {
27 if ((millis() - timer2) >= T_PERIOD2) {
28 timer2 = millis(); //сброс
29 if (PIND & (1 << 4)) {
30 PORTD &= ~(1 << 4);
31 } else {
32 PORTD |= (1 << 4);
33 }
34 }
35 }

```

Чтобы реализовать алгоритм, нам потребуется две переменные для хранения времени – timer1 и timer2. Периоды мигания запоминаем с помощью функции автозамены define T\_PERIOD1 и T\_PERIOD2 соответственно 100 и 500 мс. В 6-й строчке программы устанавливаем режим работы пинов порта 4 и 7 на выход. Для удобства понимания основного кода программы мигание отдельных светодиодов реализовали с помощью подпрограмм: mig\_led1() и mig\_led2(). Подпрограммы выполняют алгоритм, описанный для мигания единичного

светодиода, с той лишь разницей, что время и период мигания настраиваются для каждого светодиода индивидуально.

Усложним задачу для микроконтроллера, пусть он также мигает светодиодами с разной частотой, но теперь добавим опрос кнопки и при нажатии на кнопку будем засвечивать третий светодиод, который подключим к выводу A0 (0-й пин порта C). Схема представлена на рисунке 8.2.

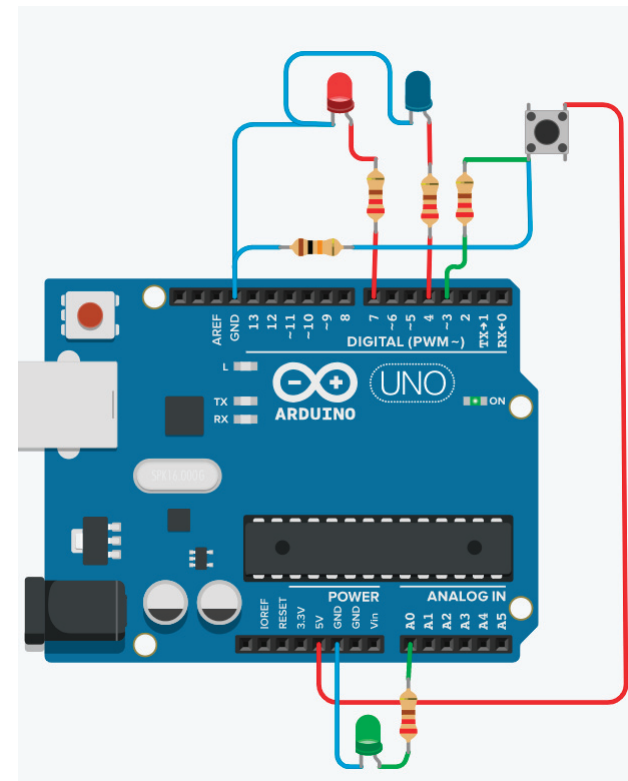


Рисунок 8.2 – Схема для реализации многозадачности

Алгоритм реализован в листинге программы, представленном ниже:

```

1  uint32_t timer1 = 0;
2  uint32_t timer2 = 0;
3  #define T_PERIOD1 100 //период переключения
4  #define T_PERIOD2 500 //период переключения
5  void setup() {
6   DDRD |= (1 << 7) | (1 << 4); // установили 7-й, 4-й пины
   порта D на выход
7   DDRD &= ~(1 << 3); // 2-й пин порта D установили
   на ввод информации
8   DDRC |= (1 << 0); // 0-й пин порта C установили
   на вывод
9  }
10 void loop() {
11  //*****мигаем 7-м портом
   *****
12  mig_led1();
13  //*****
   *****
14  //*****мигаем 4-м портом
   *****
15  mig_led2();
16  //*****
   *****
17  //*****Опрос кнопки *****
18  button(3);
19  //*****
   *****
20  }
21 void button(byte x) {
22  if ((PIND & (1 << x))) {
23   // включаем светодиод
24   PORTC |= (1 << 0);
25  } else {
26   // выключаем светодиод
27   PORTC &= ~(1 << 0);
28  }

```

```

29  }
30 void mig_led1() {
31  if ((millis() - timer1) >= T_PERIOD1) {
32   timer1 = millis(); //сброс
33   if (PIND & (1 << 7)) {
34    PORTD &= ~(1 << 7);
35   } else {
36    PORTD |= (1 << 7);
37   }
38  }
39  }
40 void mig_led2() {
41  if ((millis() - timer2) >= T_PERIOD2) {
42   timer2 = millis(); //сброс
43   if (PIND & (1 << 4)) {
44    PORTD &= ~(1 << 4);
45   } else {
46    PORTD |= (1 << 4);
47   }
48  }
49  }

```

В основную программу, листинг который мы рассмотрели выше, добавили функцию опроса кнопки `button()`, которая реализована в строках с 21-й по 29-ю программного кода. В функцию `button()` передаем номер порта, с которого снимаем информацию о кнопке. Далее вызываем функцию `button()` и переходим в 21-ю строку. Номер пина порта, переданный в функцию `button()`, хранится в локальной переменной `X` типа `byte`. В 22-й строке `if ((PIND & (1 << x)))` считываем состояние порта D и проверяем, есть ли лог. «1» на указанном порте X. Если лог. «1» установлена, то включаем светодиод в строке `PORTC |= (1 << 0)`, если нет, то в строке `PORTC &= ~(1 << 0)`.

Таким образом, основой код программы будет состоять из реализации трех подпрограмм: мигание 1-м светодиодом – строка 12, мигание 2-м светодиодом – строка 15 и опрос кнопки – строка 18. Использование подпрограмм позволяет

уменьшить тело основного кода, сделать его удобочитаемым и понятным.

```

1 void loop() {
2  //*****мигаем 7-м портом *****
3  mig_led1();
4  //*****
5  //*****мигаем 4-м портом *****
6  mig_led2();
7  //*****
8  //*****Опрос кнопки *****
9  button(3);
10 //*****
11 }

```

При реализации алгоритма мы решили задачу многозадачности для микроконтроллера, т.е. параллельного выполнения микроконтроллером нескольких задач.

## Таймеры микроконтроллера Atmega328

Работа функции millis() основана на модуле микроконтроллера Atmega328 Timer [6]. Микроконтроллер тактируется встроенными часами (осциллятор), на рисунке 8.3 выходы микроконтроллера XTAL1 и XTAL2 подключены к внешнему кварцевому генератору с частотой 16 МГц (в плате разработчика Arduino Uno микроконтроллер также питается от внешнего осциллятора).

Частота влияет на скорость работы микроконтроллера. Чем выше частота, тем выше скорость работы. Таймер использует счетчик, который считает с определенной скоростью, зависящей от частоты осциллятора (кварцевого генератора). При осцилляторе на 16 МГц состояние счетчика увеличивается на 1 каждые 62 наносекунды, т.е. 1/16000000 секунды. Фактически это время, за которое плата микроконтроллера

переходит от одной инструкции к другой (т.е. выполняет одну инструкцию).

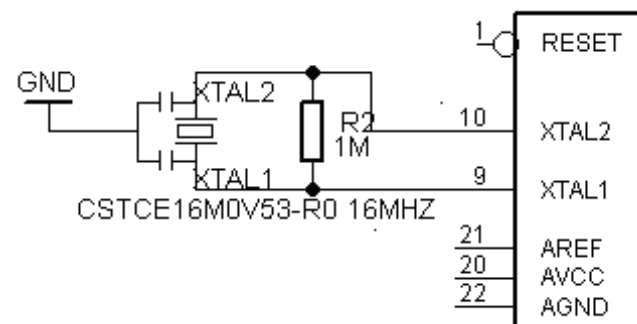


Рисунок 8.3 – Тактирующий внешний осциллятор

## Регистры таймеров микроконтроллера Atmega328

В микроконтроллере Atmega328 используется три таймера:

- **Timer0**: 8-битный таймер;
- **Timer1**: 16-битный таймер;
- **Timer2**: 8-битный таймер.

На рисунке 8.4 представлены выводы микроконтроллера, связанные с таймерами. Для изменения конфигурации таймеров в плате разработчика используются регистры управления, описание которых представлено ниже.

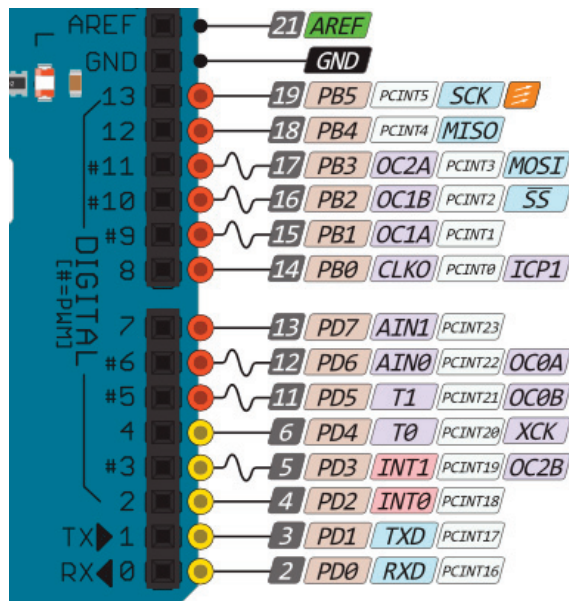


Рисунок 8.4 – Выводы таймеров микроконтроллера на плате разработчика

## Таймер/счетчик T0 (Timer/counter0)

Таймер/счетчик T0 представляет собой 8-битный модуль, который настраивается и контролируется следующими регистрами:

- TCNT0 – счетный регистр таймера/счетчика T0;
- OCR0A – регистр сравнения A;
- OCR0B – регистр сравнения B;
- TMSK0 – регистр маски прерываний для таймера/счетчика T0;
- TIFR0 – регистр флагов прерываний для таймера/счетчика T0;
- TCCR0A – регистр управления A;
- TCCR0B – регистр управления B.

Таймер/счетчик T0 обладает следующими функциями:

- сброс по совпадению;
- два независимых входа по совпадению;
- изменяемый период ШИМ-сигнала;
- фазовый корректор ШИМ-сигнала;
- различные способы тактирования;
- три независимых источника прерывания.

## Регистр TSNT0

8-битный регистр таймера/счетчика, возможен прямой доступ к чтению/записи.

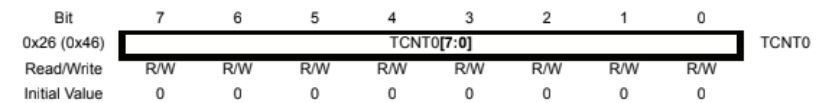


Рисунок 8.5 – Структура регистра TSNT0

## Регистр OCR0A

Регистр сравнения A содержит 8-битное значение, которое постоянно сравнивается со значением счетчика (TCNT0).

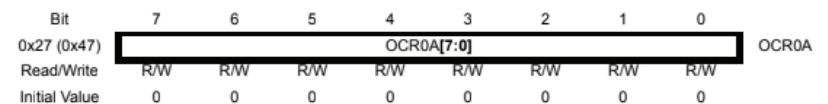


Рисунок 8.6 – Структура регистра OCR0A

Может использоваться для генерации прерывания сравнения выхода или для генерации выходного сигнала на выводе OCR0A (12-я ножка микроконтроллера или 6-я ножка платы разработчика).

## Регистр OCR0B

Регистр сравнения В содержит 8-битное значение, которое постоянно сравнивается со значением счетчика (TCNT0).

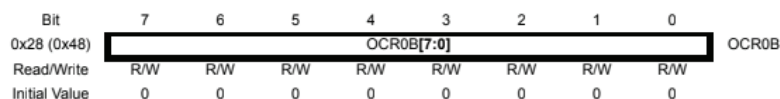


Рисунок 8.7 – Структура регистра OCR0B

Может использоваться для генерации прерывания сравнения выхода или для генерации выходного сигнала на выводе OC0B (11-я ножка микроконтроллера или 5-я ножка платы разработчика).

## Регистр TCCR0A и TCCR0B

### Регистр TCCR0A

Режим работы таймера/счетчика T0 устанавливается регистрами TCCR0A и TCCR0B, структура которого представлена на рисунке 8.8.

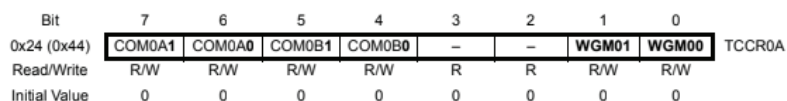


Рисунок 8.8 – Структура регистра TCCR0A

Биты 7:6 – COM0A1:0: сравнение режима выхода А, данные биты устанавливают то, какой сигнал появится на выводе OC0A (12-я ножка микроконтроллера или 6-й выход платы разработчика) при совпадении с А (совпадение значения счетного регистра TCNT0 со значением регистра сравнения OCR0A).

Таблица 8.1 – Режим таймера обычный

COM0A1	COM0A0	Состояние
0	0	Вывод OC0A не функционирует
0	1	Изменение состояния вывода OC0A на противоположное при совпадении с А
1	0	Сброс вывода OC0A в 0 при совпадении с А
1	1	Установка вывода OC0A в 1 при совпадении с А

Таблица 8.2 – Режим таймера ШИМ

COM0A1	COM0A0	Состояние
0	0	Вывод OC0A не функционирует
0	1	Если бит WGM02 регистра TCCR0B установлен в 0, вывод OC0A не функционирует. Если бит WGM02 регистра TCCR0B установлен в 1, изменение состояния вывода OC0A на противоположное при совпадении с А
1	0	Сброс вывода OC0A в 0 при совпадении с А, установка вывода OC0A в 1, если регистр TCNT0 принимает значение 0x00 (неинверсный режим)
1	1	Установка вывода OC0A в 1 при совпадении с А, установка вывода OC0A в 0, если регистр TCNT0 принимает значение 0x00 (инверсный режим)

Частота генерируемого сигнала  $f_{OCn} = f_{clk\_I/O}/256N$ , где N – коэффициент деления предделителя.

Таблица 8.3 – Режим коррекции фазы ШИМ

COM0A1	COM0A0	Состояние
0	0	Вывод OC0A не функционирует
0	1	Если бит WGM02 регистра TCCR0B установлен в 0, вывод OC0A не функционирует. Если бит WGM02 регистра TCCR0B установлен в 0, изменение состояния вывода OC0A на противоположное
1	0	Сброс вывода OC0A в 0 при совпадении с А во время увеличения значения счетчика, установка вывода OC0A в 1 при совпадении с А во время уменьшения значения счетчика

Окончание таблицы 8.3

COM0A1	COM0A0	Состояние
1	1	Установка вывода ОС0А в 1 при совпадении с А во время увеличения значения счетчика, сброс вывода ОС0А в 0 при совпадении с А во время уменьшения значения счетчика

Таблица 8.4 – Режим таймера при совпадении

COM0A1	COM0A0	Состояние
0	0	Вывод ОС0А не функционирует
0	1	Состояние вывода меняется на противоположное
1	0	Вывод сбрасывается в 0
1	1	Вывод сбрасывается в 1

Биты 5:4 – COM0B1:0: сравнение режима выхода В устанавливают то, какой сигнал появится на выводе ОС0В (11-я ножка микроконтроллера или 5-я ножка платы разработчика) при совпадении с В (совпадение значения счетного регистра TCNT0 со значением регистра сравнения OCR0B).

Таблица 8.5 – Режим таймера обычный

COM0B1	COM0B0	Состояние
0	0	Вывод ОС0В не функционирует
0	1	Изменение состояния вывода ОС0В на противоположное при совпадении с В
1	0	Сброс вывода ОС0В в 0 при совпадении с В
1	1	Установка вывода ОС0В в 1 при совпадении с В

Таблица 8.6 – Режим таймера ШИМ

COM0B1	COM0B0	Состояние
0	0	Вывод ОС0В не функционирует
0	1	Резерв
1	0	Сброс вывода ОС0В в 0 при совпадении с В, установка вывода ОС0В в 1, если регистр TCNT0 принимает значение 0x00 (прямой режим)
1	1	Установка вывода ОС0В в 1 при совпадении с В, установка вывода ОС0В в 0, если регистр TCNT0 принимает значение 0x00 (инверсный режим)

Таблица 8.7 – Режим коррекции фазы ШИМ

COM0B1	COM0B0	Состояние
0	0	Вывод ОС0В не функционирует
0	1	Резерв
1	0	Сброс вывода ОС0В в 0 при совпадении с В во время увеличения значения счетчика, установка вывода ОС0В в 1 при совпадении с В во время уменьшения значения счетчика
1	1	Установка вывода ОС0В в 1 при совпадении с В во время увеличения значения счетчика, сброс вывода ОС0В в 0 при совпадении с В во время уменьшения значения счетчика

Таблица 8.8 – Режим таймера при совпадении

COM0B1	COM0B0	Состояние
0	0	Вывод ОС0В не функционирует
0	1	Состояние вывода меняется на противоположное
1	0	Вывод сбрасывается в 0
1	1	Вывод сбрасывается в 1

Биты 1:0 – WGM01:0: сравнение режима выхода А, регистра TCCR0A устанавливают режим работы таймера/счетчика T0.

Таблица 8.9 – Выбор режима таймера

WGM02	WGM01	WGM00	Состояние
0	0	0	Обычный режим
0	0	1	Режим коррекции фазы ШИМ
0	1	0	Режим подсчета импульсов (сброс при совпадении)
0	1	1	Режим ШИМ

**Частота генерируемого сигнала** будет определяться выражением:

– при режиме сброс при совпадении:

$$fOCn = fclk\_I/O/2N(1 + OCRn),$$

где N – коэффициент деления делителя [4];

– при режиме ШИМ:

$$fOCn = fclk\_I/O/256N,$$

где N – коэффициент деления предделителя;

– при режиме ШИМ с коррекцией фазы:

$$fOCn = fclk\_I/O/512N,$$

где N – коэффициент деления предделителя.

### Регистр TCCR0B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 8.9 – Структура регистра TCCR0B

**Биты 7:6 – FOC0A:FOC0B** принудительно устанавливают значение на выводах OC0A и OC0B.

**Биты 2:0 – CS02:00** устанавливают режим тактирования и предделителя тактовой частоты таймера/счетчика T0.

Таблица 8.10 – Биты выбора предделителя

CS02	CS01	CS00	Описание
0	0	0	Таймер остановлен
0	0	1	Предделитель равен 1, т.е. выключен. Таймер считает тактовые импульсы
0	1	0	Предделитель равен 8, тактовая частота делится на 8
0	1	1	Предделитель равен 64, тактовая частота делится на 64
1	0	0	Предделитель равен 256, тактовая частота делится на 256
1	0	1	Предделитель равен 1024, тактовая частота делится на 1024
1	1	0	Тактовые импульсы идут от ножки T0 (6-я ножка микроконтроллера или 4-я ножка платы разработчика) на переходе с 1 на 0. Предделитель равен 256, тактовая частота делится на 256

Окончание таблицы 8.10

CS02	CS01	CS00	Описание
1	1	1	Тактовые импульсы идут от ножки T0 (6-я ножка микроконтроллера или 4-я ножка платы разработчика) на переходе с 0 на 1. Предделитель равен 1024, тактовая частота делится на 1024

### Регистр TIMSK0

Регистр TIMSK0 управляет прерываниями от таймера.

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 8.10 – Структура регистра TIMSK0

**Биты 2:1 – OCIE0B:OCIE0A** разрешают прерывания при совпадении с A и B при установке 1. Если в эти биты записать 0, прерывания от таймера/счетчика будут запрещены.

**Бит 0 – TOIE0** разрешает прерывание по переполнению при установке 1. Если в этот бит записать 0, прерывания от таймера/счетчика по переполнению будут запрещены.

### Регистр TIFR0

Регистр флагов прерываний для таймера/счетчика T0

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 8.11 – Структура регистра TIFR0

Биты OCF0B (2), OCF0A (1) и TOV0 (0) устанавливаются в 1 в зависимости от того, какое прерывание поступило – совпадение с A, B или переполнение.

## Таймер/счетчик T1 (Timer/counter1)

Представляет собой 16-битный модуль, который настраивается и контролируется следующими регистрами:

- **TCNT1** – счетный регистр таймера/счетчика T1;
- **OCR1A** – регистр сравнения A (16 бит);
- **OCR1B** – регистр сравнения B (16 бит);
- **TIMSK1** – регистр маски прерываний для таймера/счетчика T1;
- **TIFR1** – регистр флагов прерываний для таймера/счетчика T1;
- **TCCR1A** – регистр управления A;
- **TCCR1B** – регистр управления B;
- **TCCR1C** – регистр управления C;
- **ICR1** – регистр захвата (16 бит).

**Таймер/счетчик T1** обладает следующими функциями:

- сброс по совпадению;
- два независимых входа по совпадению;
- один вход захвата;
- блок шумоподавления входа захвата;
- изменяемый период ШИМ-сигнала;
- фазовый корректор ШИМ-сигнала;
- различные способы тактирования;
- три независимых источника прерывания.

### Регистр OCR1A

Регистр сравнения A содержит 16-битное значение, которое постоянно сравнивается со значением счетчика (TCNT1).

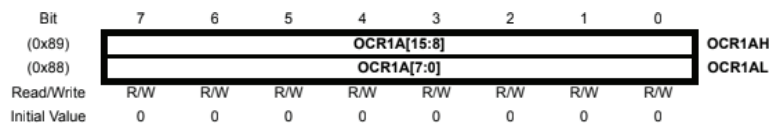


Рисунок 8.12 – Структура регистра OCR1A

Может использоваться для генерации прерывания сравнения выхода или для генерации выходного сигнала на выводе

OC1A (15-я ножка микроконтроллера или 9-я ножка платы разработчика).

### Регистр OCR1B

Регистр сравнения B содержит 16-битное значение, которое постоянно сравнивается со значением счетчика (TCNT1).

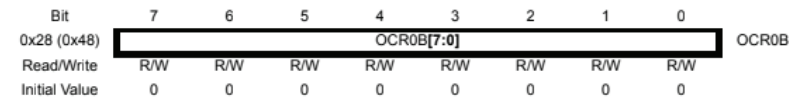


Рисунок 8.13 – Структура регистра OCR1B

Может использоваться для генерации прерывания сравнения выхода или для генерации выходного сигнала на выводе OC1B (16-я ножка микроконтроллера или 10-я ножка платы разработчика).

### Регистр TCCR1A и TCCR1B

#### Регистр TCCR1A

Режим работы таймера/счетчика T1 устанавливается регистрами TCCR1A и TCCR1B, структура которого представлена на рисунке 8.14.

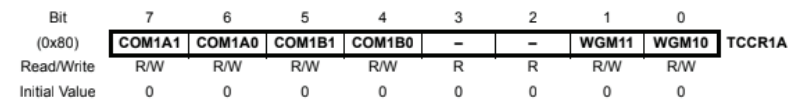


Рисунок 8.14 – Структура регистра TCCR1A

**Биты 7:6 – COM1A1:0:** сравнение режима выхода A, данные биты устанавливают то, какой сигнал появится на выводе OC1A (15-я ножка микроконтроллера или 9-й порт платы разработчика) при совпадении с A (совпадение значения счетного регистра TCNT1 со значением регистра сравнения OCR1A).



Таблица 8.11 – Режим таймера обычный

COM1A1	COM1A0	Состояние
0	0	Вывод OC0A не функционирует
0	1	Изменение состояния вывода OC1A на противоположное при совпадении с А
1	0	Сброс вывода OC1A в 0 при совпадении с А
1	1	Установка вывода OC1A в 1 при совпадении с А

Таблица 8.12 – Режим таймера ШИМ

COM1A1	COM1A0	Состояние
0	0	Вывод OC1A не функционирует
0	1	Если биты WGM13 – WGM10 установлены в (0000 – 1101), вывод OC1A не функционирует. Если биты WGM13 – WGM10 установлены в 1110 или 1111, изменение состояния вывода OC0A на противоположное при совпадении с А
1	0	Сброс вывода OC1A в 0 при совпадении с А, установка вывода OC1A в 1, если регистр TCNT1 принимает значение 0x00 (прямой режим)
1	1	Установка вывода OC1A в 1 при совпадении с А, установка вывода OC1A в 0, если регистр TCNT1 принимает значение 0x00 (инверсный режим)

Таблица 8.13 – Режим коррекции фазы ШИМ

COM1A1	COM1A0	Состояние
0	0	Вывод OC1A не функционирует
0	1	Если биты WGM13 – WGM10 установлены в (0000 – 1100, 1010, 1100 – 1111), вывод OC1A не функционирует. Если биты WGM13 – WGM10 установлены в 1101 или 1011, изменение состояния вывода OC1A на противоположное при совпадении с А
1	0	Сброс вывода OC1A в 0 при совпадении с А во время увеличения значения счетчика, установка вывода OC1A в 1 при совпадении с А во время уменьшения значения счетчика
1	1	Установка вывода OC1A в 1 при совпадении с А во время увеличения значения счетчика, сброс вывода OC1A в 0 при совпадении с А во время уменьшения значения счетчика

**Биты 5:4 – COM1B1:0:** сравнение режима выхода В, устанавливая то, какой сигнал появится на выводе OC1B (16-я ножка микроконтроллера или 10-я ножка платы разработчика) при совпадении с В (совпадение значения счетного регистра TCNT1 со значением регистра сравнения OCR1B).

Таблица 8.14 – Режим таймера обычный

COM1B1	COM1B0	Состояние
0	0	Вывод OC1B не функционирует
0	1	Изменение состояния вывода OC1B на противоположное при совпадении с В
1	0	Сброс вывода OC1B в 0 при совпадении с В
1	1	Установка вывода OC1B в 1 при совпадении с В

Таблица 8.15 – Режим таймера ШИМ

COM1B1	COM1B0	Состояние
0	0	Вывод OC1B не функционирует
0	1	Вывод OC1B не функционирует
1	0	Сброс вывода OC1B в 0 при совпадении с В, установка вывода OC1B в 1, если регистр TCNT1 принимает значение 0x00 (прямой режим)
1	1	Установка вывода OC1B в 1 при совпадении с В, установка вывода OC1B в 0, если регистр TCNT1 принимает значение 0x00 (инверсный режим)

Таблица 8.16 – Режим коррекции фазы ШИМ

COM1B1	COM1B0	Состояние
0	0	Вывод OC1B не функционирует
0	1	Вывод OC1B не функционирует
1	0	Сброс вывода OC1B в 0 при совпадении с В во время увеличения значения счетчика, установка вывода OC1B в 1 при совпадении с В во время уменьшения значения счетчика
1	1	Установка вывода OC1B в 1 при совпадении с В во время увеличения значения счетчика, сброс вывода OC1B в 0 при совпадении с В во время уменьшения значения счетчика

**Биты 4:3 WGM13:12** регистра TCCR1B и биты 1:0 – WGM11:10 регистра TCCR1A, сравнение режима выхода А,

регистра TCCR0A, устанавливают режим работы таймера/счетчика T1.

Таблица 8.17 – Режимы работы таймера

WGM13	WGM12	WGM11	WGM10	Состояние
0	0	0	0	Обычный режим
0	0	0	1	Коррекция фазы ШИМ, 8 бит
0	0	1	0	Коррекция фазы ШИМ, 9 бит
0	0	1	1	Коррекция фазы ШИМ, 10 бит
0	1	0	0	Режим счета импульсов (OCR1A) (сброс при совпадении)
0	1	0	1	ШИМ, 8 бит
0	1	1	0	ШИМ, 9 бит
0	1	1	1	ШИМ, 10 бит
1	0	0	0	Коррекция фазы и частоты ШИМ (ICR1)
1	0	0	1	Коррекция фазы и частоты ШИМ (OCR1A)
1	0	1	0	Коррекция фазы ШИМ (ICR1)
1	0	1	1	Коррекция фазы и частоты ШИМ (OCR1A)
1	1	0	0	Режим счета импульсов (ICR1) (сброс при совпадении)
1	1	0	1	Резерв
1	1	1	0	ШИМ (ICR1)
1	1	1	1	ШИМ (OCR1A)

Регистр TCCR1B

Bit (0x81)	7	6	5	4	3	2	1	0	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 8.15 – Структура регистра TCCR1B

**Биты 7 – ICNC1** управляет схемой подавления помех блока захвата (0 – выключена / 1 – включена).

**Биты 6 – ICES1** выбирает активный фронт регистра захвата (0 – по спадающему фронту сигнала / 1 – по нарастающему фронту сигнала).

**Биты 2:0 – CS12:10** устанавливают режим тактирования и предделителя тактовой частоты таймера/счетчика T1.

Таблица 8.18 – Выбор режима тактирования и предделителя таймера

CS12	CS11	CS10	Описание
0	0	0	Таймер остановлен
0	0	1	Предделитель равен 1, т.е. выключен, таймер считает тактовые импульсы
0	1	0	Предделитель равен 8, тактовая частота делится на 8
0	1	1	Предделитель равен 64, тактовая частота делится на 64
1	0	0	Предделитель равен 256, тактовая частота делится на 256
1	0	1	Предделитель равен 1024, тактовая частота делится на 1024
1	1	0	Тактовые импульсы идут от ножки T1 (11-я ножка микроконтроллера или 5-й пин платы разработчика) на переходе с 1 на 0
1	1	1	Тактовые импульсы идут от ножки T1 (11-я ножка микроконтроллера или 5-й пин платы разработчика) на переходе с 0 на 1

Регистр TCCR1C

Bit (0x82)	7	6	5	4	3	2	1	0	TCCR1C
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 8.16 – Структура регистра TCCR1C

**Биты 7:6 – FOC1A и FOC1B** регистра TCCR1C принудительно устанавливают значение на выводах OC1A и OC1B.

## Регистр TIMSK1

Регистр TIMSK1 управляет прерываниями от таймера.

Bit	7	6	5	4	3	2	1	0	
(0x6F)	-	-	ICIE1	-	-	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 8.17 – Структура регистра TIMSK1

**Бит 5 – ICIE1** разрешает прерывание по захвату.

**Биты 2:1 – OCIE1B:OCIE1A** разрешают прерывания при совпадении с А и В при установке 1. Если в эти биты записать 0, прерывания от таймера/счетчика будут запрещены.

**Бит 0 – TOIE1** разрешает прерывание по переполнению при установке 1. Если в этот бит записать 0, прерывания от таймера/счетчика по переполнению будут запрещены.

## Регистр TIFR1

Регистр флагов прерываний для таймера/счетчика T1.

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	-	-	ICF1	-	-	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 8.18 – Структура регистра TIFR1

Биты ICF1 (5), OCF1B (2), OCF1A (1) и TOV1 (0) устанавливаются в 1 в зависимости от того, какое прерывание поступило – захват, совпадение с А, В или переполнение.

## Таймер/счетчик T2 (Timer/counter0)

Представляет собой 8-битный модуль, который настраивается и контролируется следующими регистрами:

- TCNT2 – счетный регистр таймера/счетчика T2;

- OCR2A – регистр сравнения А;
- OCR2B – регистр сравнения В;
- TIMSK2 – регистр маски прерываний для таймера/счетчика T2;
- TIFR2 – регистр флагов прерываний для таймера/счетчика T2;
- TCCR2A – регистр управления А;
- TCCR2B – регистр управления В;
- ASSR – регистр асинхронного режима;
- GTCCR – главный регистр всех таймеров/счетчиков.

**Таймер/счетчик T2** обладает следующими функциями:

- сброс по совпадению;
- два независимых входа по совпадению;
- изменяемый период ШИМ-сигнала;
- фазовый корректор ШИМ-сигнала;
- различные способы тактирования;
- три независимых источника прерывания;
- возможность работы от независимого внешнего часового тактового генератора 32 кГц;
- асинхронный режим;
- делитель частоты 10 бит.

Режим работы таймера/счетчика T2 устанавливается регистрами TCCR2A и TCCR2B аналогично таймеру/счетчику T0.

## Регистр TCCR2B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 8.19 – Структура регистра TCCR2B

**Биты 7:6 – FOC2A:FOC2B** принудительно устанавливают значение на выводах OC2A и OC2B.

**Биты 2:0 – CS02:00** устанавливают режим тактирования и делителя тактовой частоты таймера/счетчика T2.

Таблица 8.19 – Установка делителя таймера

CS02	CS01	CS00	Описание
0	0	0	Таймер остановлен
0	0	1	Делитель равен 1, т.е. выключен. Таймер считает тактовые импульсы
0	1	0	Делитель равен 8, тактовая частота делится на 8
0	1	1	Делитель равен 32, тактовая частота делится на 32
1	0	0	Делитель равен 64, тактовая частота делится на 64
1	0	1	Делитель равен 128, тактовая частота делится на 128
1	1	0	Делитель равен 256, тактовая частота делится на 256
1	1	1	Делитель равен 1024, тактовая частота делится на 1024

### Регистр ASSR

Регистр управления асинхронным режимом работы таймера/счетчика 2.

Bit (0xB6)	7	6	5	4	3	2	1	0	ASSR
Read/Write	R	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 8.20 – Структура регистра ASSR

**Бит 6 EXCLK** разрешает использование внешнего тактового сигнала от кварцевого резонатора 32 кГц при записи в него 1.

**Бит 5 AS2** (управляет способом тактирования (1 – от внешнего резонатора 32 кГц, подключенного к TOSC1 (9-я ножка), / 0 – от внутреннего генератора CLK)

**Бит 4 TCN2UB** показывает, доступен ли для записи регистр TCNT2 (1 – недоступен / 0 – доступен).

**Бит 3 OCR2AUB** показывает, доступен ли для записи регистр OCR2A (1 – недоступен / 0 – доступен).

**Бит 2 OCR2BUB** показывает, доступен ли для записи регистр OCR2B (1 – недоступен / 0 – доступен).

**Бит 1 TCR2AUB** показывает, доступен ли для записи регистр TCCR2A (1 – недоступен / 0 – доступен).

**Бит 0 TCR2BUB** показывает, доступен ли для записи регистр TCCR2B (1 – недоступен / 0 – доступен).

### Регистр GTCCR

Главный регистр всех таймеров/счетчиков.

Bit	7	6	5	4	3	2	1	0	GTCCR
0x23 (0x43)	TSM	-	-	-	-	-	PSRASYS	PSRSYNCS	
Read/Write	R/W	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 8.21 – Структура регистра GTCCR

**Бит 7 TSM** запрещает автоматический сброс битов PSRASYS и PSRSYNCS.

**Бит 1 PSRASYS** сбрасывает делитель таймера/счетчика T2 если установить в 1, после этого бит сбрасывается в 0 автоматически.

**Бит 0 PSRSYNCS** сбрасывает делитель таймера/счетчика T0 и T1, если установить в 1, после этого бит сбрасывается в 0 автоматически.

### Широтно-импульсная модуляция

Широтно-импульсная модуляция (PWM – pulse-width modulation), или ШИМ, – это метод преобразования сигнала, при котором изменяется длительность импульса, а частота остается неизменной. Вид такого сигнала представлен на рисунке 8.22. Широтно-импульсная модуляция используется в различных электронных устройствах:

- в модулях подсветки экранов ЖК-дисплеев;
- импульсных преобразователях, которые входят в состав современных блоков питания;
- сварочных инверторах;
- импульсных источниках преобразования энергии и многих других.

В микроконтроллере ШИМ-сигнал могут генерировать выходы, связанные с таймерами.

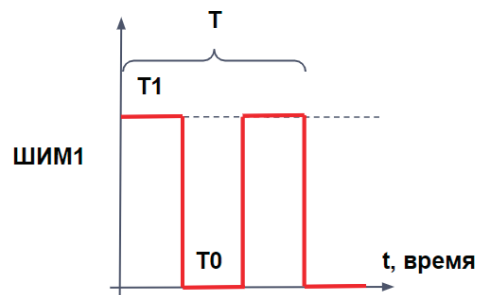


Рисунок 8.22 – Широтно-импульсный сигнал

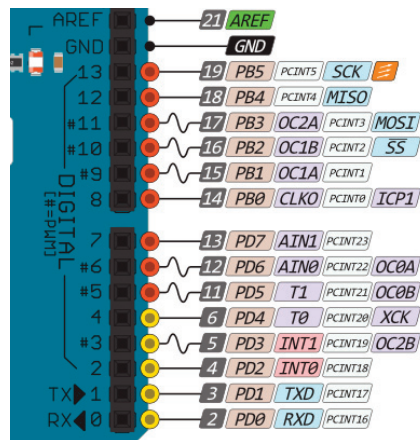


Рисунок 8.23 – Назначение портов платы разработчика

Задание ШИМ-сигнала на требуемом порту микроконтроллера при программировании на языке Arduino используется следующая команда:

`analogWrite(порт, величина).`

Порт – номер порта платы разработчика (3, 5, 6, 9, 10 и 11) (рисунок 8.23).

Величина – параметр, который отвечает за ширину импульса, его можно изменять от 0 до 255.

Рассмотрим пример формирования ШИМ-сигнала на порту 11 платы разработчика. Используем листинг программы, представленный ниже:

```

1 void setup()
2 { analogWrite(11, 125); }
3 void loop() { }

```

На осциллограмме, представленной на рисунке 8.24, можно рассчитать частоту сигнала:

$$f=1/T=1/(4*0.5*10^{-3})= 500 \text{ Гц.}$$

Данная частота для многих задач импульсного управления в электронике слишком маленькая, поэтому воспользуемся знаниями, полученными в разделах про таймер, и увеличим частоту сигнала на 11-м выходе платы разработчика.

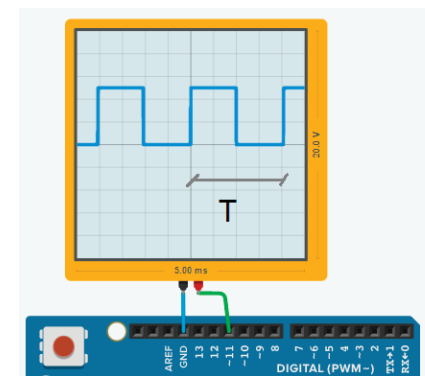


Рисунок 8.24 – Результат эксперимента

## Настройка ШИМ управляющими регистрами

Используем различные режимы работы таймера для формирования требуемого сигнала. Рассмотрим 3-й порт платы разработчика, видим, что данный порт связан с таймером 2 (OC2B). Настроим ШИМ-сигнал таймера 2 и зададим различную величину делителя таймера.

Настройка таймера осуществляется регистрами микроконтроллера TCCR2A и TCCR2B. Для этого с помощью битов CS22, CS21, CS20 устанавливаем число 110 в регистр TCCR2B (листинг программы, строки 12–13). Режим работы таймера устанавливаем с помощью битов WGM21, WGM20 регистра TCCR2A, устанавливаем работу таймера как ШИМ с коррекцией фазы, т.е. записав число 11 в соответствующие биты (листинг программы, строки 15–16). Активацию сигнала осуществляем в 17-й строке командой analogWrite(11, 125), где 11 – пин порта платы разработчика, 125 – ширина импульса ШИМ-сигнала (может быть 0–125).

```
1 void setup() {
2   // Устанавливаем делитель частоты
3   // 001 делитель 1
4   // 010 делитель 8
5   // 011 делитель 32
6   // 100 делитель 64
7   // 101 делитель 128
8   // 110 делитель 256
9   // 111 делитель 1024
10  // Устанавливаем делитель частоты 110,
    что соответствует 256
11  TCCR2B |= (1 << CS22);
12  TCCR2B |= (1 << CS21);
13  TCCR2B &= ~(1 << CS20);
14  // устанавливаем режим работы таймера на 11-м выходе
    как ШИМ
15  TCCR2A &= ~(1 << WGM21);
16  TCCR2A |= (1 << WGM20);
```

```
17 analogWrite(11, 125);
18 }
19 void loop() {}
```

Результат работы кода представлен на рисунке 8.25, где частота сигнала равна 122 Гц, таким образом можно изменять величину делителя и получать сигналы различной частоты.

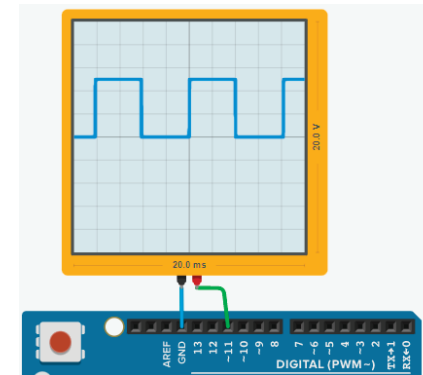


Рисунок 8.25 – Результат эксперимента  
f сигнала = 122 Гц (делитель 256)

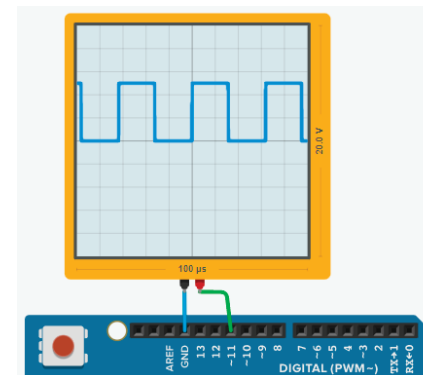


Рисунок 8.26 – Результат эксперимента  
f сигнала = 31.25 кГц (делитель 1)

Результат экспериментов с делителем таймера представлен в виде таблицы. Проверить результаты эксперимента легко, воспользовавшись листингом программы, представленным выше, при этом необходимо изменять делитель в строках 11–12. Эксперимент проводился при частоте осциллятора 16 МГц.

Таблица 8.20 – Результаты эксперимента с делителем

Делитель	Частота сигнала на 11-м пине (экспериментальная)	$f = f_{clk\_I/O}/512N$ , где N – коэффициент деления делителя (расчетная)
1	31.25 кГц	$F = (16 \cdot 10^6)/512 \cdot 1 = 31.25 \text{ кГц}$
8	3.9 кГц	$F = (16 \cdot 10^6)/512 \cdot 8 = 3.9 \text{ кГц}$
32	980 Гц	$F = (16 \cdot 10^6)/512 \cdot 32 = 980 \text{ Гц}$
64	490 Гц	$F = (16 \cdot 10^6)/512 \cdot 64 = 490 \text{ Гц}$
128	245 Гц	$F = (16 \cdot 10^6)/512 \cdot 128 = 245 \text{ Гц}$
256	122 Гц	$F = (16 \cdot 10^6)/512 \cdot 256 = 122 \text{ Гц}$
1024	30 Гц	$F = (16 \cdot 10^6)/512 \cdot 1024 = 30 \text{ Гц}$

Используя различные настройки работы порта, можно генерировать различные частоты на портах, связанные с таймерами микроконтроллера.

## 9 Сторожевой таймер

При программировании микроконтроллера никто не застрахован от появления таких условий, при которых выполнение кода будет невозможно [8], в результате чего программа закичивается и микроконтроллер не может самостоятельно выйти из «западни». С целью безотказной работы и для контроля в микроконтроллере установлен сторожевой таймер. Он позволяет осуществить программный сброс микроконтроллера через промежутки времени [9]. Работая в нормальном режиме, микроконтроллер может обнулять счетчик сторожевого таймера до сброса.

Характеристики сторожевого таймера:

- режим защиты от сбоев при отключении только программированием fuse;
- выбор режима генерации сброса от 16 миллисекунд до 8 секунд;
- прерывания и возможность сброса системы;
- отдельный генератор тактовых импульсов (128 кГц).

### Регистр WDTCSR

Регистр управления сторожевого таймера.

Bit (0x60)	7	6	5	4	3	2	1	0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

Рисунок 9.1 – Регистр сторожевого таймера WDTCSR

**Бит 7 WDIF** – флаг поступления прерывания от сторожевого таймера (1 – если поступило прерывание (сбрасывается записью 1 во флаг)).

**Бит 6 WDIE** – разрешение прерывания от сторожевого таймера при записи в него 1.

**Бит 4 WDCE** – разрешение изменения режима работы сторожевого таймера (перед изменением режима работы или предделителя необходимо записать 1).

**Бит 3 WDE** – разрешение сброса системы сторожевым таймером при записи в него 1.

**Бит 5 WDP3 и биты (2–0) WDP2-WDP0** управляют временем, через которое произойдет сброс или поступит прерывание от сторожевого таймера.

Таблица 9.1 – Настройка сброса сторожевого таймера

WDP3	WDP2	WDP1	WDP0	Время сброса
0	0	0	0	2048 циклов (16 мс)
0	0	0	1	4096 циклов (32 мс)
0	0	1	0	8192 цикла (64 мс)
0	0	1	1	16 384 цикла (0.125 с)
0	1	0	0	32 768 циклов (0.25 с)
0	1	0	1	65 536 циклов (0.5 с)
0	1	1	0	131 072 цикла (1 с)
0	1	1	1	262 144 цикла (2 с)
1	0	0	0	524 288 циклов (4 с)
1	0	0	1	1 048 576 циклов (8 с)
1010 – 1111				Резерв

В режиме работы (сброс и прерывание) сначала выполняется прерывание, а затем сброс. И если Fuse бит **WDTON** установлен в 0, вне зависимости от битов **WDE** и **WDIE** регистра **WDTCR** таймер будет сбрасывать систему по истечении времени, на которое он настроен.

Также для сброса системы сторожевым таймером необходимо в бит **WDRF (3)** регистра **MCUSR** записать 1 (сбрасывается в 0 принудительно или по включении питания микроконтроллера).

## Библиотека `avr/wdt`

Для упрощения взаимодействия с микроконтроллером воспользуемся библиотекой `avr/wdt`.

В состав библиотеки входят три функции.

## `void wdt_enable(timeout)`

Функция разрешает работу сторожевого таймера, задает время тайм-аута. Аргумент `timeout` (время тайм-аута) может принимать следующие значения:

```

WDTO_15MS // 15 мс
WDTO_30MS // 30 мс
WDTO_60MS // 60 мс
WDTO_120MS // 120 мс
WDTO_250MS // 250 мс
WDTO_500MS // 500 мс
WDTO_1S // 1 сек
WDTO_2S // 2 сек
WDTO_4S // 4 сек
WDTO_8S // 8 сек

```

## `void wdt_reset(void)`

Сброс сторожевого таймера. Для нормальной работы необходимо вызывать эту функцию не реже периода сторожевого таймера. При задержке, превышающей тайм-аут, произойдет аппаратный сброс контроллера.

## `void wdt_disable(void)`

Отключение сторожевого таймера.

Листинг программы, которая позволяет реализовать сторожевой таймер, представлен ниже:

```

1 #include <avr/wdt.h> // подключение библиотеки
   для работы со сторожевым таймером
2 void setup()
3 {
4   pinMode(13, OUTPUT); // назначить тип пина
5   digitalWrite(13, HIGH); // включить светодиод
6   delay(1000); // ждем 1 с
7   digitalWrite(13, LOW); // выключить светодиод
8   delay(1000); // ждем 1 с
9 }

```



```

10 void loop(){
11   wdt_enable(WDTO_15MS); // разрешить watchdog
12   // функция запустится через 15 мс, если
      // не сбросить таймер
13   while(1)
14   {
15     // wdt_reset(); // раскомментируйте для избежания
      // перезагрузок
16   }

```

В 1-й строке подключаем библиотеку `avr/wdt.h` для работы со сторожевым таймером. В функции `setup` настраиваем режим 13-го пина на работу в режиме «выход» (4-я строка программы). Далее в 5-й и 8-й строке осуществляет однократное мигание светодиода с периодом в 1 секунду. После этого программа заходит в основной цикл программы `loop`, где в 11-й строке разрешает работу сторожевого таймера с частотой обновления 15 мс. Далее программа попадает в бесконечный цикл, где функция сброса сторожевого таймера закомментирована. То есть попав во внутренний бесконечный цикл, программа зациклится, и если бы не активировали сторожевой таймер, то мигание светодиода мы бы не наблюдали. Однако при работе кода будем видеть, что каждые 15 мс у нас будет перезагружаться микроконтроллер и мы будем видеть постоянно мигающий светодиод, подключенный к 13-му пину порта, так как код программы будет начинаться с начала раз за разом. Если раскомментировать в 15-й строке функцию сброса сторожевого таймера в бесконечном цикле, то произойдет сброс таймера и программа перезагружаться не будет, тем самым мигание светодиода на 13-м пине мы не увидим. Таким образом, сторожевой таймер позволяет нам обеспечивать стабильную работу программы.

## 10 Аналоговые порты микроконтроллера

### Аналоговые сигналы и датчики

Микроконтроллеру необходимо работать с аналоговыми сигналами. Аналоговый сигнал – это сигнал, порождаемый физическим процессом, параметры которого можно измерить в любой момент времени. Типовым аналоговым сигналом может служить информация о переменном напряжении в сети питания (рисунок 10.1, а). Внешний вид аналогового сигнала может быть разнообразным, его общий вид представлен на рисунке 10.1, б.

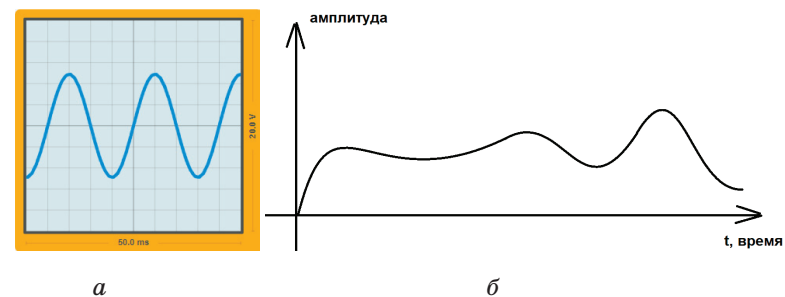


Рисунок 10.1 – Примеры аналогового сигнала

Аналоговые сигналы поступают с устройств, которые называют аналоговыми датчиками. Существует много различных типов датчиков, но основное их назначение – преобразовывать информацию об измеряемой физической величине в пропорциональный электрический сигнал, постоянный по времени. Примеры таких датчиков представлены на рисунке 10.2.

Микроконтроллер работает только с числами и оперирует только ими, поэтому для преобразования аналогового сигнала в цифровой код используется модуль – аналогово-цифровой преобразователь (АЦП). В состав микроконтроллера Atmega328 входит 10-битный АЦП. За работу АЦП микрокон-

троллера отвечает порт С, на рисунке 10.3 представлены выводы порта С с их назначением. В плате разработчика аналоговые порты обозначены (А0–А5) представлены на рисунке 10.3.

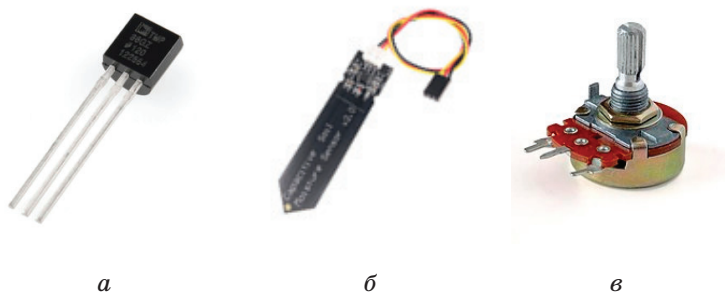


Рисунок 10.2 – Аналоговые датчики: *a* – датчик температуры; *b* – датчик влажности; *v* – потенциометр

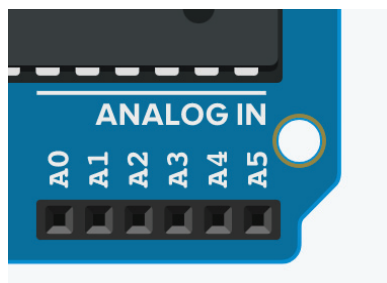


Рисунок 10.3 – Аналоговые порты на плате разработчика

## Работа с аналоговыми входами

Разрабатывая программу на языке Arduino, приходится иметь дело с аналоговыми сигналами. Для работы с аналоговыми портами существуют приведенные ниже функции.

## analogRead(pin)

Данная функция возвращает 10-битное число, соответствующее уровню напряжения на указанном порте **pin**. Значение порта можно вносить как А0–А6, так и указав номер порта 0–6.

Напряжение, поданное на аналоговый вход, преобразуется в значение от 0 до 1023, это 1024 шага с разрешением 0.0049 В. Разброс напряжения и шаг могут быть изменены функцией analogReference().

## analogReference(type)

Данная функция позволяет задать источник опорного напряжения (ИОН) для АЦП. По умолчанию опорным ИОН является напряжения питания (5 В).

type:

- DEFAULT: стандартное опорное напряжение 5 В;
- INTERNAL: встроенное опорное напряжение 1.1 В на микроконтроллерах Atmega328;
- EXTERNAL: внешний источник опорного напряжения, подключенный к выводу AREF (рисунок 10.5).

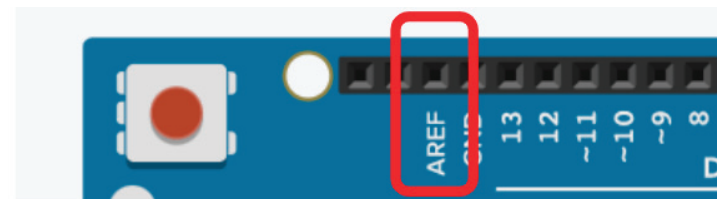


Рисунок 10.4 – Пин AREF для подключения внешнего опорного напряжения

Опорное напряжение меняется в микроконтроллере, для того чтобы повысить точность измерения канала.

Например, если с аналогового датчика поступает напряжение в диапазоне от 0 до 1 В и при настройке АЦП ИОН

оставить по умолчанию на уровне 5 В, то точность измерения составит  $5/1023 = 0,0049$  В/бит.

Если ИОН изменить на уровень напряжения 1.1 В, то точность измерения составит  $1,1/1023 = 0,0012$  В/бит.

Если же ИОН изменить на уровень напряжения 1 В, то точность измерения составит  $1/1023 = 0,001$  В/бит, но при таком способе необходим стабилизированный внешний источник питания.

Рассмотрим пример получения данных с аналогового датчика влажности (рисунок 10.5), подключенного к микроконтроллеру (рисунок 10.6). На рисунке 10.5 видны основные выводы аналогового датчика:

- VCC – положительный пин – питание, к нему подключается источник питания (+5 В);
- GND – нулевой пин, или заземление, к нему подключается минус источника питания;
- SIG – сигнальный пин, вывод аналогового сигнала.

Источником питания для датчика влажности будет плата разработчика, для этого вывод платы 5V подключим к входу датчика VCC, соединим GND платы и датчика, а выход датчика SIG соединим с аналоговым входом микроконтроллера A0. Общая схема соединения представлена на рисунке 10.6.

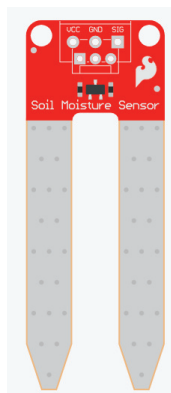


Рисунок 10.5 – Аналоговый датчик влажности

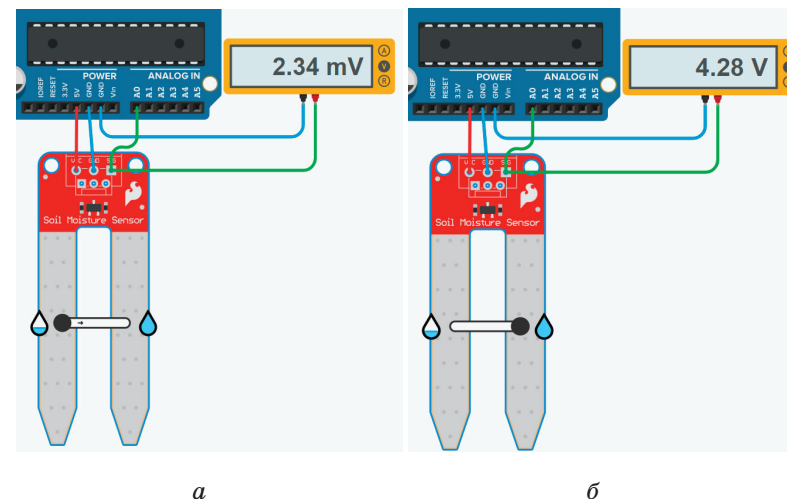


Рисунок 10.6 – Подключение аналогового датчика к микроконтроллеру

При изменении уровня влажности от минимального значения (а) до максимального (б) уровень напряжения на выводе SIG меняется от 2.34 мВ до 4.28 В. Запишем программу на микроконтроллер, которая позволит превратить аналоговый сигнал в число. Листинг программы представлен ниже:

```

1 int data;
2 void setup ()
3 {
4   Serial.begin(9600);
5 }
6 void loop ()
7 {
8   data = analogRead(A0);
9   Serial.println(data);
10 }

```

Для удобства понимания полученной информации в данном коде используем библиотеку Serial, речь о которой пойдет в других разделах пособия.

В строке 4 инициализируем протокол передачи данных по UART на скорости 9600 бит/с. В 8-й строке считываем данные с аналогового порта A0 и заносим в переменную data. Переменная **data** имеет тип данных int – integer, так как необходимо хранить число в пределах 0–1023. Далее в 9-й строке вводим переменную data в монитор порта. Активизируем просмотр монитора порта, где наблюдаем изменение переменной data.

В результате работы программы в монитор порта будут выводиться значения от 0 до 876 (рисунок 10.7).

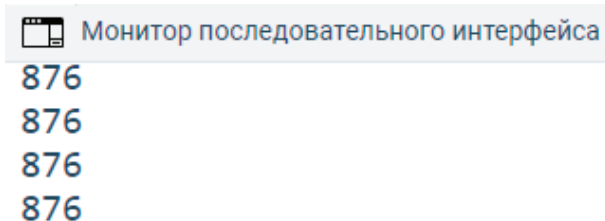


Рисунок 10.7 – Вывод результата в монитор последовательного интерфейса

Чтобы понять, почему же выводится значение 876 при максимальной влажности, воспользуемся уравнением, описанным в мануале микроконтроллера [10]:

$$ADC = (V_{in} * 1024) / V_{ref}$$

где  $V_{in}$  – входное напряжение на аналоговый порт;

$V_{ref}$  – напряжение источника опорного напряжения;

1024 – максимальное число при 10-битном регистре.

$$ABC = (4,28 * 1024) / 5 = 876,$$

что соответствует приведенному эксперименту, представленному на рисунке 10.6.

## Настройка АЦП управляющими регистрами

В состав АЦП микроконтроллера Atmega328 входят следующие регистры:

- **ADMUX** – регистр настройки мультиплексора АЦП;
- **ADCSRA** – регистр статуса и контроля А;
- **ADCSRБ** – регистр статуса и контроля В.

Настройка АЦП микроконтроллера состоит из следующих шагов:

- 1) настройка регистр ADMUX (регистр настройки мультиплексора АЦП);
- 2) настройка регистра ADCSRA (регистр статуса и контроля А);
- 3) настройка ADCSRB (регистр статуса и контроля В);
- 4) чтение результата преобразования.

### Регистр ADMUX

Регистр настройки мультиплексора АЦП представляет собой 8-битный регистр, представленный на рисунке 10.8, биты которого формируют управляющие воздействия для настройки преобразования АЦП.

Bit (0x7C)	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	ADMUX
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 10.8 – Структура регистра ADMUX

**Биты 7–6 REFS1 и REFS0.** Данная группа битов определяет, какой тип источника опорного напряжения (ИОН) будет подключен к микроконтроллеру (таблица 10.1). При инициализации начальное состояние определяется как 0. Если эти биты изменятся во время преобразования, то изменение не вступит в силу, пока это преобразование не будет завершено (установлен бит ADIF в ADCSRA). Внутреннее опорное на-

пряжение нельзя использовать, если на вывод AREF подается внешнее опорное напряжение.

Таблица 10.1 – Выбор ИОН АЦП

REFS1	REFS0	Тип ИОН, подключенного ко входу опорного напряжения
0	0	Подключен внешний опорный ИОН, внутренний ИОН выключен
0	1	Подключено напряжение питания AVcc
1	0	Состояние зарезервировано
1	1	Подключен внутренний ИОН 1.1 В, с внешним конденсатором на AREF-пине

**Бит 5 ADLAR.** Данный бит выбирает тип представления результата преобразования. При записи в этот бит логической «1» результат в регистрах ADCL и ADCH будет представлен в левостороннем представлении, иначе – правосторонний. При инициализации по умолчанию устанавливает значение 0. Результат установки бита ADLAR представлен на рисунке 10.9.

**ADLAR = 0**

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

**ADLAR = 1**

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 10.9 – Вывод данных АЦП при различной настройке бита ADLAR

**Биты 3–0 MUX3, MUX2, MUX1 и MUX0.** Данная группа битов определяет, какой вход будет активен в качестве входа для АЦП. Для определения номера пина входного канала необходимо данную группу сконфигурировать согласно таблице 10.2.

Таблица 10.2 – Выбор пина АЦП

MUX3...0	Номер пина
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8

**Регистр ADCSRA**

Регистр управления и статуса А представляет собой 8-битный регистр, представленный на рисунке 10.10, биты которого формируют управляющие воздействия для настройки работы АЦП.

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 10.10 – Структура регистра ADCSRA

**Бит 7 ADEN.** Данный бит предназначен для включения режима АЦП. Для включения АЦП необходимо в этот бит записать логическую «1», если необходимо отключить, то записываем 0. При инициализации, значение в этом бите равно 0.

**Бит 6 ADSC.** Данный бит включает режим преобразования входного аналогового сигнала в двоичный код. Для запуска преобразования необходимо установить в этот бит логическую «1». После завершения преобразования данный бит устанавливается в 0-е значение.

**Бит 5 ADATE.** Данный бит включает автоматический запуск преобразования. Автоматическое преобразование запускается с помощью внешнего сигнала. Если преобразование завершено, а внешний сигнал присутствует, то новое преобразование не начинается. Также если преобразование не завершилось, а пришел другой внешний сигнал о начале преобразования, то данный сигнал будет проигнорирован микроконтроллером. По завершении преобразования будет выставлен флаг прерывания, даже если были отключены прерывания или бит глобального прерывания был очищен. Преобразование может быть сделано таким образом, чтобы не вызывать прерываний. Для того чтобы вызвать новое преобразование, необходимо очистить бит флага прерываний.

**Бит 4 ADIF.** Когда выполнено преобразование, выставляется флаг о завершении преобразования. Сбросить флаг преобразования можно путем записи в данный бит логической «1». Также данный бит сбрасывается, если включены прерывания от АЦП и вызывается вектор прерывания.

**Бит 3 ADIE.** При записи в данный бит логической «1» и если установлен 1 бит в регистре SREG (общий регистр статуса), включаются прерывания от АЦП по выполнении преобразования.

**Биты 2–0 ADPS2, ADPS1 и ADPS0.** С помощью данной группы битов устанавливается делитель между системной частотой и входной частотой АЦП. Конфигурация данной группы битов, а также какой будет соответствовать делитель, указаны в таблице 10.3.

Таблица 10.3 – Выбор делителя АЦП

ADPS2	ADPS1	ADPS0	Делитель
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

### Регистр ADCSRB

Регистр контроля и статуса В представляет собой 8-битный регистр, представленный на рисунке 10.11, биты которого формируют управляющие воздействия для настройки работы АЦП.

Bit (0x7B)	7	6	5	4	3	2	1	0	ADCSRБ
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 10.11 – Структура регистра ADCSRB

**Биты 2–0 ADTS2, ADTS1 и ADTS0.** Предназначены для определения, от какого элемента необходимо выполнять запуск преобразования. В таблице 10.4 представлена конфигурация битов и соответствующий источник для запуска преобразования.

Таблица 10.4 – Источник запуска АЦП

ADTS2	ADTS1	ADTS0	Источник запуска
0	0	0	Свободный режим
0	0	1	Совпадение по аналоговому компаратору
0		0	Внешнее 0 прерывание

#### Окончание таблицы 10.4

ADTS2	ADTS1	ADTS0	Источник запуска
0	1	1	Совпадение с регистром сравнения А таймера/счетчика 0
1	0	0	Переполнение таймера/счетчика 0
1	0	1	Совпадение с регистром сравнения В таймера/счетчика 1
1	1	0	Переполнение таймера/счетчика 1
1	1	1	При изменении счетчика/таймера 1

Составим программу, по алгоритму действия аналогичную описанному выше, но конфигурация АЦП будет осуществляться через регистры управления АЦП микроконтроллера. Листинг программы представлен ниже:

```

1 unsigned int u;
2 void setup()
3 {
4   Serial.begin(9600);
5   ADC_Init();
6 }
7 void loop()
8 {
9   ADCSRA |= (1 << ADSC); // Начинаем преобразование
10  while ((ADCSRA & (1 << ADIF)) == 0); // пока не будет
    выставлено флага об окончании преобразования
11  u = (ADCL|ADCH << 8); // Считываем полученное значение
12  Serial.println(u);
13  //Проверяем считанное значение
14 }
15 void ADC_Init(){
16  ADCSRA |= (1 << ADEN) // Включаем АЦП
17  ADCSRA |= (1 << ADPS1)|(1 << ADPS0); // устанавливаем
    предделитель преобразователя на 8
18  ADMUX |= (0 << REFS1)|(1 << REFS0) //выставляем опорное
    напряжение, как внешний ИОН
19  |(0 << MUX0)|(0 << MUX1)|(0 << MUX2)|(0 << MUX3); //
    снимать сигнал будем с входа PC0
20 }

```

В строке 1 инициализируем переменную, которая будет хранить значение с ADCL и ADCH. В строке 4 открываем протокол передачи данных по последовательному протоколу UART. В 5-й строке вызывается пользовательская функция ADC\_Init(), которая конфигурирует АЦП в строках 17–22:

В строке 18 устанавливается лог. «1» в бит ADEN, который разрешает преобразование АЦП, в строке 19 устанавливается битами ADPS1 и ADPS0 предделитель АЦП, в строке 20 битами REFS1 и REFS0 выставляется источник опорного напряжения, напряжение питание АЦП, и в строке 21 битами MUX0, MUX1, MUX2, MUX3 устанавливаем выход порта, с которого снимаем показания.

После выполнения функции ADC\_Init() выполняются строки, находящиеся в бесконечном цикле loop ().

В строке 10 битом ADSC установкой лог. «1» начинается преобразование АЦП, преобразование будет до тех пор, пока в бите ADIF не установится 1 – это реализовано в 11-й строке циклом с предусловием while ((ADCSRA & (1 << ADIF)) == 0). После выполнения преобразования в 12-й строке данные с регистров ADCL и ADCH заносятся в переменную u. Выводим в монитор последовательного порта (строка 13) значение u.

# 11 Протокол передачи USART

## Характеристики модуля USART микроконтроллера

Микроконтроллер может осуществлять обмен данными между другими устройствами, в том числе и с компьютером. В этом ему помогает модуль UART – последовательный асинхронный интерфейс [11, 12].

Характеристики USART0:

- независимые прием и передача (полный дуплекс);
- асинхронный и синхронный режим;
- внешняя/внутренняя синхронизация;
- высокое разрешение генератора скорости передачи;
- выбор длины блока данных (5–9 бит) и длина стоп бита (1–2);
- проверка четности;
- проверка переполнения данных;
- проверка на ошибки;
- фильтрация паразитных шумов;
- прерывания;
- мультипроцессорный режим работы;
- двойная скорость в асинхронном режиме.

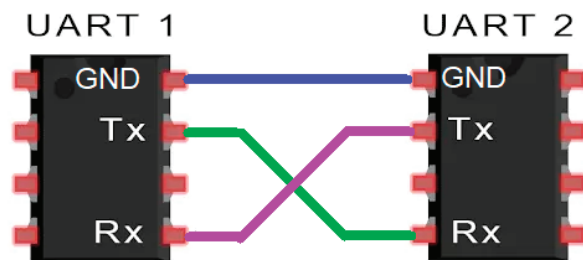


Рисунок 11.1 – Соединение двух устройств по протоколу UART

UART использует два пина контроллера – Rx и Tx, где Rx обозначает Receiver (передатчик), Tx – Transmitter (прием-

ник). Для связи двух устройств понадобятся два провода, причем соединить их следует крест-накрест Rx первого в Tx второго, и наоборот (рисунок 11.1).

Микроконтроллер Atmega328 обладает одним модулем UART, используя при этом пины 0 и 1 платы разработчика (2-й и 3-й выводы микроконтроллера) (рисунок 11.2).

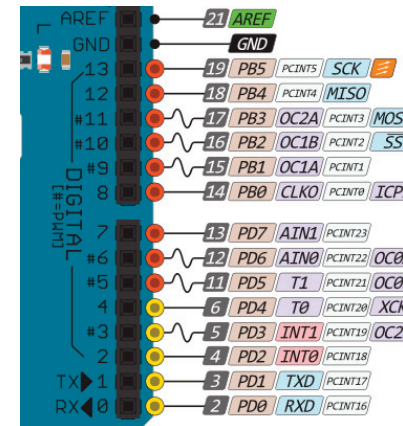


Рисунок 11.2 – Назначение портов платы разработчика

Передача данных выглядит как цепочка битов, разделенных на байты, плюс (опционально) сигналов служебных. Пакет данных представлен на рисунке 11.3.

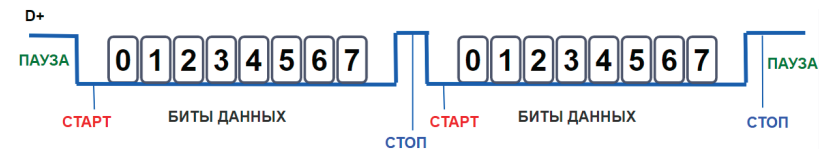


Рисунок 11.3 – Пакет данных по протоколу UART

Пока информация не передается, в линии сохраняется высокий уровень сигнала. Спад сигнала – это команда принимающей стороне, что сейчас начнется передача данных. Через



определенное количество времени, зависящее от заранее заданной скорости, начинается передача байта в виде ряда нулей и/или единиц согласно заранее оговоренной скорости передачи данных. После восьмого бита следует стоп-сигнал в виде высокого уровня сигнала, и ситуация повторяется до тех пор, пока не будут переданы все нужные байты.

Служебные сигналы используются в качестве дополнительной меры борьбы с ошибками, которые вызываются помехами в проводах и контактах. Если добавить в протокол бит четности, он будет передаваться в цепочке данных, сразу за последним битом информационного байта и перед стоповым битом. Наличие или отсутствие такого бита тоже оговаривается в протоколе заранее, наряду со скоростью и прочими параметрами. Основные параметры: скорость, количество битов, паритет (четность), длина стоп-сигнала. Отсюда вытекает важное условие для работы UART: оба устройства должны быть настроены одинаково, иначе они друг друга не поймут.

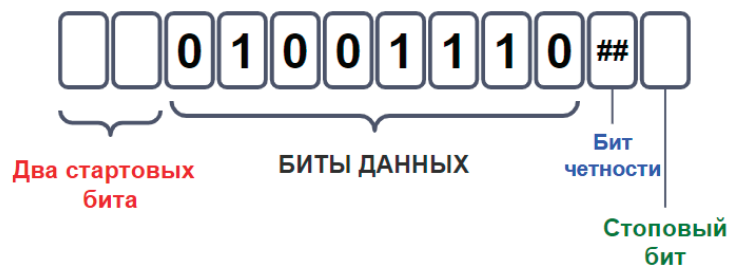


Рисунок 11.4 – Служебные биты протокола UART

Скорость передачи измеряется в бодах (битах в секунду) округленно до сотни. Существует стандартный ряд скорости передачи данных: 4800, 9600, 19 200, 38 400, 57 600, 115 200 бод.

Чем выше скорость, тем быстрее передаются данные, но тем больше вероятность появления ошибок, поэтому рекомендуется соблюдать принцип разумной достаточности. Для большинства случаев рекомендуется скорость 9600 бод – примерно байт за 1 мс.

## Настройка USART управляющими регистрами

В состав USART микроконтроллера atmega328 входят следующие регистры:

**UDR0** – входной/выходной регистр данных;

**UCSR0A, UCSR0B, UCSR0C** – регистры управления;

**UBRR0H, UBRR0L** – регистры управления скоростью передачи.

### Регистр UDR0

Регистр буфера передачи данных USART и регистр буфера приема данных USART совместно используют один и тот же адрес ввода/вывода, называемый регистр данных USART или UDRn.

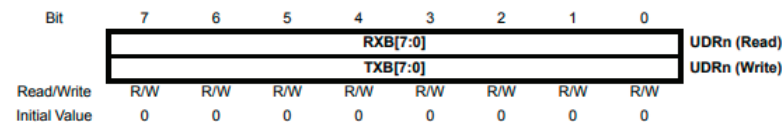


Рисунок 11.5 – Входной/выходной регистр данных UDR0

### Регистр UCSR0A

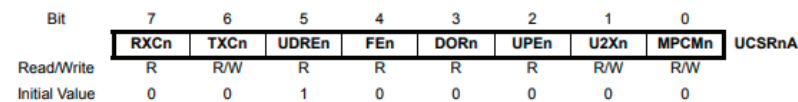


Рисунок 11.6 – Регистры управления UCSR0A

**Бит 7 RXC0** – флаг завершения приема (1 – если в регистре UDR0 есть непрочитанные данные / 0 – после того, как регистр UDR0 опустошен).

**Бит 6 TXC0** – флаг завершения передачи (1 – после завершения передачи из сдвигового регистра и если в UDR0 не было загружено нового значения. Флаг сбрасывается записью в него 1).

**Бит 5 UDRE0** – флаг опустошения регистра данных UDR0 (устанавливается в 1 после пересылки данных из UDR0 в сдвиговый регистр передатчика и означает, что в регистр данных можно загружать новое значение. сбрасывается при записи в UDR0 новых данных).

**Бит 4 FE0** – флаг ошибки кадрирования (при обнаружении ошибки кадрирования (первый стоп-бит равен 0) устанавливается в 1, сбрасывается при приеме стоп-бита, равного 1).

**Бит 3 DOR0** – флаг переполнения (устанавливается в 1, если в момент обнаружения нового старт-бита в сдвиговом регистре находится последнее принятое слово, а буфер приемника полон (2 значения)).

**Бит 2 UPEN0** – флаг ошибки контроля четности (устанавливается в 1 при ошибке четности).

**Бит 1 U2X0** – удвоение скорости обмена, если установить в 1 (только в асинхронном режиме, в синхронном следует установить этот бит в 0).

**Бит 0 MPCM0** – режим микропроцессорного обмена (если установлен в 1, режим включен).

### Регистр UCSR0B

Bit	7	6	5	4	3	2	1	0	UCSRnB
	<b>RXCIEn</b>	<b>TXCIEn</b>	<b>UDRIEn</b>	<b>RXENn</b>	<b>TXENn</b>	<b>UCSZn2</b>	<b>RXB8n</b>	<b>TXB8n</b>	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 11.7 – Регистры управления UCSR0B

**Бит 7 RXCIE0** – разрешение прерывания при завершении приема, если установлен в 1.

**Бит 6 TXCIE0** – разрешение прерывания при завершении передачи, если установлен в 1.

**Бит 5 UDRIE0** – разрешение прерывания при очистке регистра данных, если установлен в 1.

**Бит 4 RXEN0** – разрешение приема, если установлен в 1.

**Бит 3 TXEN0** – разрешение передачи, если установлен в 1.

**Бит 2 UCSZ02** – формат посылки данных (используется совместно с битами UCSZ01 и UCSZ00 регистра UCSR0C).

**Бит 1 RXB80** – 8-й разряд принимаемых данных при использовании 9-разрядных слов.

**Бит 0 TXB80** – 8-й разряд передаваемых данных при использовании 9-разрядных слов.

### Регистр UCSR0C

Bit	7	6	5	4	3	2	1	0	UCSRnC
	<b>UMSELn1</b>	<b>UMSELn0</b>	<b>UPMn1</b>	<b>UPMn0</b>	<b>USBSn</b>	<b>UCSZn1</b>	<b>UCSZn0</b>	<b>UCPOLn</b>	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

Рисунок 11.8 – Регистры управления UCSR0C

**Биты 7 и 6 UMSEL01 и UMSEL00** отвечают за режим работы USART0.

Таблица 11.1 – Установка режима работы протокола

UMSEL01	UMSEL00	Режим
0	0	Асинхронный режим
0	1	Синхронный режим
1	0	Резерв
1	1	Ведущий SPI

**Биты 5 и 4 UPM01 и UPM00** отвечают за режим работы системы контроля и формирования четности USART0.

Таблица 11.2 – Установка режима работы системы контроля и четности

UPM01	UPM00	Режим
0	0	Выключен
0	1	Резерв
1	0	Проверка четности
1	1	Проверка нечетности

**Бит 3 USBS0** устанавливает количество стоп-битов (1 стоп-бит, если сброшен в 0 / 2 стоп-бита, если установлен в 1).

**Бит 2 UCSZ02 (2) регистра UCSR0B и биты 2 и 1 UCSZ01 и UCSZ00 (2, 1) регистра UCSR0C** устанавливают длину передаваемых посылок.

Таблица 11.3 – Установка длины передаваемых посылок

UCSZ02	UCSR01	UCSZ00	Длина передаваемых посылок
0	0	0	5 бит
0	0	1	6 бит
0	1	0	7 бит
0	1	1	8 бит
1	0	0	Резерв
1	0	1	Резерв
1	1	0	Резерв
1	1	1	9 бит

**Бит 0 UCPOL0** устанавливает полярность тактовых сигналов:

- 0 – передача по спаду, прием по нарастанию;
- 1 – передача по нарастанию, прием по спаду.

### Регистры UBRR0H, UBRR0L

В регистр UBRR0 записывается определенное значение в зависимости от тактовой частоты и скорости передачи. Значение вычисляется по формуле  $UBRR0 = CLK / (16 * \text{Скорость}) - 1$ .

Bit	15	14	13	12	11	10	9	8	
	-	-	-	-	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 11.9 – Регистры управления скоростью передачи

Рассмотрим листинг программы для использования протокола передачи USART для микроконтроллера Atmega328, настройку протокола передачи данных будем осуществлять с помощью регистров управления. Схема для проведения эксперимента представлена на рисунке 11.10.

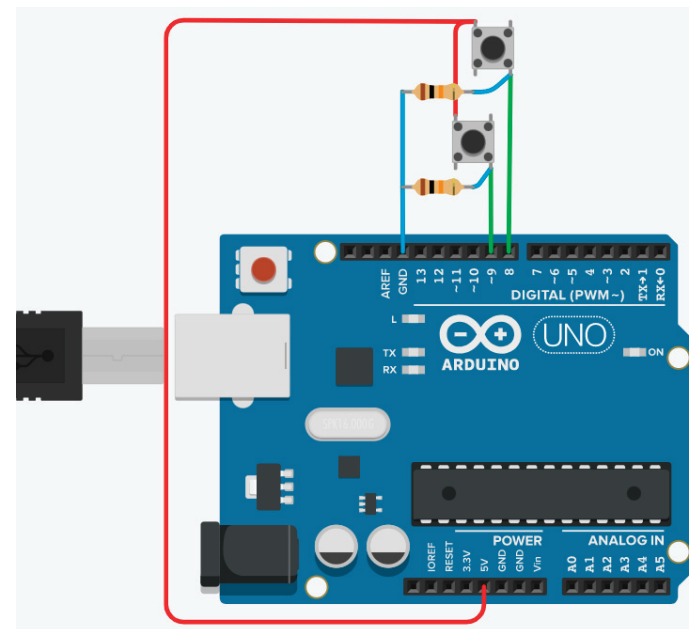


Рисунок 11.10 – Схема тестирования протокола USART

При запуске программы по последовательному протоколу передачи данных отправим слово «OK!». При нажатии на кнопку, подключенную к 8-му пину платы разработчика, выводим слово «PB0», при нажатии на кнопку, подключенную к пину 9, – слово «PB1».

```

1 #include <avr/io.h>
2 #include <util/delay.h>
3 #define F_CPU 16000000 // Рабочая частота контроллера

```

```

4 #define BAUD 9600L // Скорость обмена данными
5 #define UBRR0L_value (F_CPU/(BAUD*16))-1 //Согласно
заданной скорости подсчитываем значение для регистра
UBRR0L
6 void init_USART() {
7     UBRR0L = UBRR0L_value; //Младшие 8 бит UBRR0L_value
8     UBRR0H = UBRR0L_value >> 8; //Старшие 8 бит UBRR0L_
value
9     UCSR0B |= (1 << TXEN0); //Бит разрешения передачи
10    UCSR0B |= (1 << UCSZ02); //Устанавливаем формат 8 бит
данных
11    UCSR0C |= (1 << UCSZ00) | (1 << UCSZ01);
12 }
13 void send_UART(char value) {
14     while (!(UCSR0A & (1 << UDRE0))); // Ожидаем, когда
очистится буфер передачи
15     UDR0 = value; // Помещаем данные в буфер, начинаем
передачу
16 }
17 int main(void)
18 {
19     init_pin();
20     init_USART(); //инициализация USART в режиме
9600/8-N-1
21     send_UART(0x4F); //посылаем ASCII код знака 'O'
22     send_UART(0x4B); //посылаем ASCII код знака 'K'
23     send_UART(0x21); //посылаем ASCII код знака '!'
24     send_UART(0x0D); //переход в начало строки
25     send_UART(0x0A); //переход на новую строку
26     while(1)
27     {
28         if (PINB & (1 << 0)) // если кнопка нажата
29         {
30             send_UART(0x50); // символ ASCII буквы 'P'
31             send_UART(0x42); //символ ASCII буквы 'B'
32             send_UART(0x30); //символ ASCII цифры '0'
33             send_UART(0x0D);
34             send_UART(0x0A);
35             _delay_ms(500);

```

```

36     }
37     if (PINB & (1 << 1)) // если кнопка нажата
38     {
39         send_UART('P');
40         send_UART('B');
41         send_UART('!');
42         send_UART(0x0D);
43         send_UART(0x0A);
44         _delay_ms(500);
45     }
46 }
47 }
48 void init_pin(void)
49 {
50     DDRB = 0b00000000; //PB0, PB1 input

```

В 1-й строке подключаем к исполняемому коду файл `avr/io.h`. В этом файле находятся определения констант, имен регистров и всего необходимого, что может понадобиться для выполнения кода программы. Во 2-й строке `util/delay.h` подключаем файл, позволяющий использовать задержки в коде программы. В 3-й строке устанавливаем рабочую частоту контроллера. В 4-й строке задаем скорость обмена данными. В 5-й строке рассчитываем по формуле  $UBRR0 = CLK / (16 * \text{Скорость}) - 1$ , согласно заданной скорости подсчитываем значение для регистра `UBRR0`.

С 6-й по 12-ю строки формируем подпрограмму `init_USART()` производим настройку протокола USART.

В 7-ю и 8-ю строки заносим рассчитанную частоту передачи данных в регистры `UBRR0L` и `UBRR0H`.

В 9-й строке устанавливаем бит разрешения передачи `TXEN0` в регистре `UCSR0B`.

В 10-й и 11-й строках битами `UCSZ02`, `UCSZ01` и `UCSZ00` в соответствующих регистрах `UCSR0B` и `UCSR0C` устанавливаем 8-битный формат данных.

В строках 13–16 формируем подпрограмму `send_UART`, позволяющую отправить символы по протоколу данных USART:

- в 14-й строке с помощью цикла `while` ожидаем, когда очистится буфер передачи и при этом произойдет сброс бита UDRE0 регистра UCSRA;
- в 15-й строке в регистр UDR0 передаем отправляемый символ `value`.

В строках 48–52 формируем подпрограмму `init_pin` настройки пинов порта В, к которому подключены тактовые кнопки, – это пины 0 и 1 регистра В, что соответствует 8-му и 9-му выводам платы разработчика. Данные пины настраиваются на ввод данных.

С 17-й по 47-ю строку выполняется основной код:

- в 19-й строке вызывается подпрограмма настройки пинов `init_pin`;
- в 20-й строке вызывается программа инициализации протокола UART;
- в 21-й строке отправляется символ «О» с вызовом подпрограммы `send_UART`;
- в 22-й строке отправляется символ «К» с вызовом подпрограммы `send_UART`;
- в 23-й строке отправляется символ «!» с вызовом подпрограммы `send_UART`;
- в 24-й строке отправляется номер – символ в шестнадцатеричном формате `0x0D`, переход в начало строки с вызовом подпрограммы `send_UART`;
- в 25-й строке отправляется номер – символ в шестнадцатеричном формате `0x0A`, переход на новую строку с вызовом подпрограммы `send_UART`;
- в строках с 26-й по 46-ю попадаем в бесконечный цикл `while`, где производим опрос тактовых кнопок, подключенных к 0 и 1 пина порта В;
- в 28-й строке проверяем, если кнопка подключенная к пину 0 порта В нажата `PINB&(1<<0)`, то с вызовом подпрограммы `send_UART` отправляем посимвольно слово `PB0`;
- в 35-й строке делаем задержку на 500 мс;

- в строках с 37-й по 45-ю производим опрос 1-го пина порта В и с вызовом подпрограммы `send_UART` отправляем посимвольно слово `PB1`.

Аналогичную задачу можно осуществить, используя библиотеку `Serial`.

Рассмотрим листинг программы с использованием библиотеки:

```
1 void setup ()
2 {
3   Serial.begin(9600);
4   Serial.println(«OK!»);
5 }
6 void loop ()
7 {
8   if(PINB&(1<<0))// если кнопка нажата
9   {
10    Serial.println(«PB0»);
11    delay(500);
12  }
13  if(PINB&(1<<1))// если кнопка нажата
14  {
15    Serial.println(«PB1»);
16    delay(500);
17  }
18 }
```

Использование библиотеки `Serial` позволяет сократить код. В 3-й строке программы производится с помощью команды `Serial.begin(9600)` инициализация протокола данных на скорости 9600 бит в секунду, в 4-й строке вводим с помощью функции `Serial.println(«OK!»)` строку «OK!». В бесконечном цикле `loop` осуществляем опрос кнопок и при нажатии на соответствующую кнопку выводим строки «PB0» или «PB1».

## 12 Прерывание

Прерывания прекращают работу основной программы, для того чтобы выполнить более приоритетную, определяемую внутренними или внешними событиями, влияющими на работу микроконтроллера. При этом в стек записывается содержимое счетчика команд, а в сам счетчик записывается вектор прерывания, по которому находится команда безусловного перехода к подпрограмме обработки прерывания. После выполнения подпрограммы обработки прерывания в счетчик команд загружается сохраненное значение из стека, и выполнение основной программы продолжается с того места, где оно остановилось.

Таблица векторов прерывания располагается в памяти программ, начиная с адреса 0x0002, либо в области загрузчика. Приоритет прерывания зависит также от его расположения в таблице векторов прерывания: чем меньше адрес, тем выше приоритет прерывания.

### Регистр MCUCR

Регистр управления микроконтроллером.

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	-	BODS	BODSE	PUD	-	-	IVSEL	IVCE	MCUCR
Read/Write	R	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 12.1 – Регистр управления микроконтроллером MCUCR

**Бит 1 IVSEL** указывает, где будет располагаться таблица векторов прерывания:

- 0 – по адресу 0x0002;
- 1 – по адресу начала загрузчика + 0x0002.

**Бит 0 IVCE** разрешает изменение бита IVSEL в течение 4 машинных циклов после установки его в 1.

Разрешение обработки прерываний осуществляется установкой 1 в бит I (7) регистра SREG:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 12.2 – Регистр статуса SREG

Таблица векторов прерывания микроконтроллера Atmega328 выглядит следующим образом:

0x0000 RESET – сброс;

0x0002 INTO – внешнее прерывание 0;

0x0004 INT1 – внешнее прерывание 0;

0x0006 PCINT0 – прерывание по изменению состояния нулевой группы выводов;

0x0008 PCINT1 – прерывание по изменению состояния первой группы выводов;

0x000A PCINT2 – прерывание по изменению состояния второй группы выводов;

0x000C WDT – прерывание от сторожевого таймера;

0x000E TIMER2 COMPA – прерывание от таймера/счетчика T2 при совпадении с A;

0x0010 TIMER2 COMPB – прерывание от таймера/счетчика T2 при совпадении с B;

0x0012 TIMER2 OVF – прерывание по переполнению таймера/счетчика T2;

0x0014 TIMER1 CAPT – прерывание от таймера/счетчика T1 по записи;

0x0016 TIMER1 COMPA – прерывание от таймера/счетчика T1 при совпадении с A;

0x0018 TIMER1 COMPB – прерывание от таймера/счетчика T2 при совпадении с B;

0x001A TIMER1 OVF – прерывание по переполнению таймера/счетчика T1;

0x001C TIMER0 COMPA – прерывание от таймера/счетчика T0 при совпадении с A;

0x001E TIMER0 COMPB – прерывание от таймера/счетчика T0 при совпадении с B;

0x0020 TIMER0 OVF – прерывание по переполнению таймера/счетчика T0;

0x0022 SPI, STC – прерывание по окончании передачи модуля SPI;

0x0024 USART, RX – прерывание по окончании приема модуля USART;

0x0026 USART, UDRE – прерывание по опустошении регистра данных модуля USART;

0x0028 USART, TX – прерывание по окончании приема модуля USART;

0x002A ADC – прерывание по завершении преобразования АЦП;

0x002C EE READY – прерывание по готовности памяти EEPROM;

0x002E ANALOG COMP – прерывание от аналогового компаратора;

0x0030 TWI – прерывание от модуля I2C (TWI);

0x0032 SPM READY – прерывание по готовности FLASH-памяти.

## Внешние прерывания

Внешние прерывания – прерывания от внешних устройств, подключенных к микроконтроллеру, а также событий, происходящих на входах микроконтроллера. Для управления внешними прерываниями предназначен регистр EICRA.

Bit	7	6	5	4	3	2	1	0	
(0x69)	-	-	-	-	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 12.3 – Регистр EICRA

**Бит 3 ISC11 и Бит 2 ISC10** управляют событиями, в результате которых будет сгенерировано прерывание INT1.

Таблица 12.1 – Выбор события прерывания INT1

ISC11	ISC10	Событие
0	0	Низкий уровень сигнала на входе INT1
0	1	Любое изменение уровня сигнала на входе INT1
1	0	По спадающему сигналу на входе INT1
1	1	По возрастающему сигналу на входе INT1

**Бит 1 ISC01 и Бит 0 ISC00** управляют событиями, в результате которых будет сгенерировано прерывание INT0.

Таблица 12.2 – Выбор события прерывания

ISC01	ISC00	Событие
0	0	Низкий уровень сигнала на входе INT0
0	1	Любое изменение уровня сигнала на входе INT0
1	0	По спадающему сигналу на входе INT0
1	1	По возрастающему сигналу на входе INT0

Разрешением внешних прерываний управляет регистр EIMSK (рисунок 12.4).

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	-	-	-	-	-	-	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 12.4 – Структура регистра EIMSK

**Бит 1 INT1** разрешает внешние прерывания INT1 при записи в него 1.

**Бит 0 INT0** разрешает внешние прерывания INT0 при записи в него 1.

Чтобы определить, откуда поступило внешнее прерывание, существует регистр флагов прерываний EIFR (рисунок 12.5).

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	-	-	-	-	-	-	INTF1	INTF0	EIFR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 12.5 – Структура регистра EIFR

**Бит 1 INTF1** устанавливается в 1, если прерывание поступило от INT1.

**Бит 0 INTF0** устанавливается в 1, если прерывание поступило от INT0.

Помимо внешних прерываний на выводах INT1 (5-я ножка) и INT0 (4-я ножка) можно использовать различные группы выводов микроконтроллера. В микроконтроллере Atmega328 существует три группы выводов, управление которыми осуществляется регистром PCICR (регистр управления прерыванием смены контакта) (рисунок 12.6).

Bit	7	6	5	4	3	2	1	0	
(0x68)	-	-	-	-	-	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 12.6 – Структура регистра PCICR

**Бит 2 PCIE2** разрешает прерывания по изменению состояния второй группы выводов при установке в него 1.

**Бит 1 PCIE1** разрешает прерывания по изменению состояния первой группы выводов при установке в него 1.

**Бит 0 PCIE0** разрешает прерывания по изменению состояния нулевой группы выводов при установке в него 1.

Контроль за тем, от какой группы выводов поступило прерывание, осуществляется регистром PCIFR (рисунок 12.7).

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	-	-	-	-	-	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 12.7 – Структура регистра PCIFR

**Бит 2 PCIF2** устанавливается в 1, если прерывание поступило от второй группы выводов.

**Бит 1 PCIF1** устанавливается в 1, если прерывание поступило от первой группы выводов.

**Бит 0 PCIF0** устанавливается в 1, если прерывание поступило от нулевой группы выводов.

Влиять на генерацию прерывания может любое изменение на любом выводе группы. Для того чтобы включить необходимые выводы групп, существуют регистры-маски PCMSK2, PCMSK1, PCMSK0.

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
(0x6C)	-	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 12.8 – Структура регистра PCMSK2-PCMSK0

Для включения необходимого входа соответствующий бит в регистре-маске должен быть установлен в 1.

Рассмотрим электрическую схему, представленную на рисунке 12.10. Задача состоит в том, чтобы с помощью внешнего прерывания определить количество нажатий на тактовую кнопку. В микроконтроллере Atmega328 пины 2 и 3 платы разработчика (5-й и 6-й пины микроконтроллера) обладают дополнительными функциями внешнего прерывания INT0 и INT1 соответственно. Задача заключается в следующем: инкрементировать переменную в прерывании при каждом нажатии от тактовой кнопки. Количество нажатий выводить в последовательный интерфейс UART.



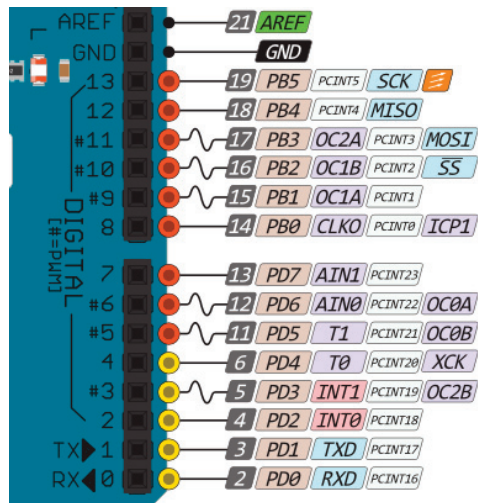


Рисунок 12.9 – Назначение портов платы разработчика

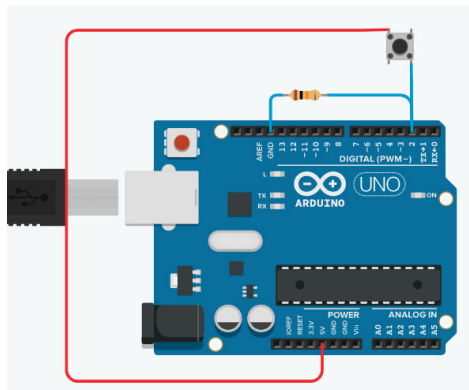


Рисунок 12.10 – Электрическая схема эксперимента

Алгоритм реализован в листинге программы, представленном ниже:

```

1 int a=0; // количество импульсов
2 void setup()
3 {
4   Serial.begin(9600);
5   int_ini();
6   sei();
7 }
8 void loop()
9 {
10  //выводим количество импульсов
11  Serial.print(« a =»);
12  Serial.println(a);
13
14 }
15 void int_ini(void)
16 {
17  //включим прерывания INT0 по нисходящему фронту
18  EICRA |= (1<<ISC00);
19  EICRA |= (1<<ISC01);
20  //разрешаем внешние прерывания INT0
21  EIMSK |= (1<<INT0);
22 }
23 // обработчик прерывания
24 ISR(INT0_vect)
25 {
26  a++;
27 }

```

В первой строке инициализируем переменную, отвечающую за количество нажатий. В 4-й строке инициализируем последовательный протокол передачи данных UART. В 5-й строке вызывается пользовательская функция int\_ini() для настройки внешнего прерывания. 15-я и 22-я строки содержат код функции int\_ini(). В 18-й и 19-й строках настраивается прерывание по возрастающему сигналу на входе INT0, для этого в биты регистра EICRA записывается лог. «1» в соответствующие биты ISC01 и ISC00. Битом INT0 регистра EIMSK в 21-й строке разрешаем прерывание. При возникновении события внешнего прерывания на 2-м порту платы

разработчика вызывается обработчик прерывания ISR(INT0\_vect), в котором инкрементируется переменная а. 6-я строка разрешает все прерывания. В основном цикле программы происходит вывод переменной а в последовательный интерфейс UART.

Приведем пример программы, реализующий описанный алгоритм выше, но с использованием языка Arduino:

```
1 int a=0; // количество импульсов
2 void setup()
3 {
4   Serial.begin(9600);
5   attachInterrupt(0, INT0_vect, FALLING);
6 }
7 void loop()
8 {
9   //выводим количество импульсов
10  Serial.print(« a =»);
11  Serial.println(a);
12 }
13 void INT0_vect()
14 {
15   a++;
16 }
```

В первой строке инициализируем переменную, отвечающую за количество нажатий. В 4-й строке инициализируем последовательный протокол передачи данных UART. В 5-й строке вызывается функция attachInterrupt для настройки внешнего прерывания:

- **attachInterrupt(interrupt, function, mode)**
- **interrupt:** номер прерывания (int)
- **function:** функция, вызываемая прерыванием, функция должна быть без параметров и не возвращать значений.
- **mode** задает режим обработки прерывания.

Допустимо использование следующих констант:

- **LOW** вызывает прерывание, когда на порту LOW;

- **CHANGE** – прерывание вызывается при смене значения на порту с LOW на HIGH и наоборот;
- **RISING** – прерывание вызывается только при смене значения на порту с LOW на HIGH;
- **FALLING** – прерывание вызывается только при смене значения на порту с HIGH на LOW.

При возникновении события внешнего прерывания на 2-м порту платы разработчика вызывается обработчик прерывания INT0\_vect(), в котором инкрементируется переменная а. В основном цикле программы происходит вывод переменной а в последовательный интерфейс UART.

## Определение скорости двигателя с помощью энкодера и прерывания

Рассмотрим электрическую схему, представленную на рисунке 12.11. Задача состоит в том, чтобы с помощью инкрементного энкодера определить скорость вращения мотора.

Данную задачу будем решать с использованием внешних прерываний на микроконтроллере. Пины 2 и 3 платы разработчика (5-й и 6-й пины микроконтроллера) обладают дополнительными функциями внешнего прерывания INT0 и INT1 соответственно. Задача заключается в следующем: инкрементировать при каждом импульсе от датчика энкодера при вращении двигателя и при достижении определенного значения времени производить расчет скорости. Рассчитанную скорость вывести в последовательный интерфейс UART.

Двигатель подключается через регулируемый источник напряжения, изменяя напряжение, будем менять скорость двигателя. Двигатель содержит импульсный энкодер, при полном обороте которого на выводах А и В появляются 660 импульсов. Энкодер питается также от независимого источника напряжения. Для отслеживания импульсов от энкодера в цепи измерения подключен осциллограф. Импульсы от энкодера подаются на 2-й порт платы разработчика, функцией которого является внешнее прерывание INT0.

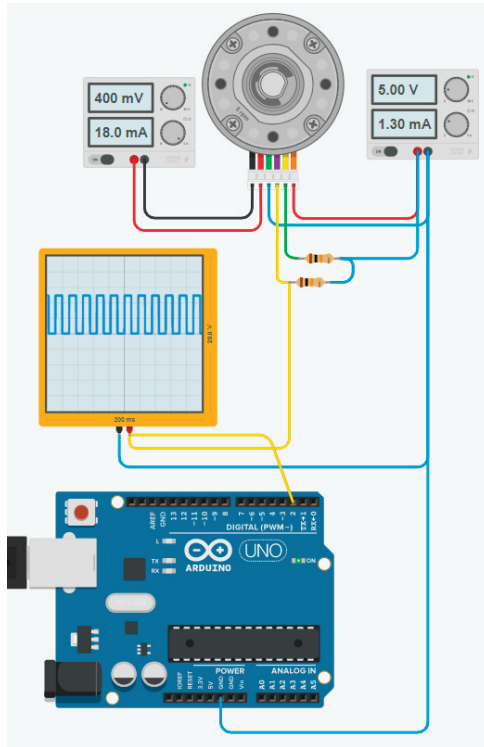


Рисунок 12.11 – Электрическая схема эксперимента

Листинг программы, реализующий данный алгоритм, представлен ниже:

```

1 unsigned long t=0;
2 volatile int a=0; // количество импульсов
3 float speed=0;
4 void setup()
5 {
6   Serial.begin(9600);
7   int_ini(); // инициализация внешнего прерывания INT0
8   sei(); // разрешение глобального прерывания

```

```

9   }
10 void loop()
11 {
12   //выводим количество импульсов
13   if (millis() - t>250)
14   {
15     speed = (a*4*60/660); // расчет скорости 660 имп./об
16     a=0; // сбрасываем счетчик при достижении времени 250
17     т=millis();
18     Serial.print(« speed =»);
19     Serial.println(speed);
20   }
21 }
22 void int_ini(void)
23 {
24   //включаем прерывания INT0 по нисходящему фронту
25   EICRA |= (1<<ISC00);
26   EICRA |= (1<<ISC01);
27   //разрешаем внешние прерывания INT0
28   EIMSK |= (1<<INT0);
29 }
30 // обработчик прерывания
31 ISR(INT0_vect)
32 {
33   a++; // счетчик импульсов от датчика
34 }

```

В строках с 22-й по 29-ю осуществляем настройку внешнего прерывания, которое вызывается по нисходящему фронту сигнала, поданного на 2-й порт платы разработчика. Для этого в биты ISC00 и ISC01 регистра EICRA устанавливается лог. «1». Установкой лог. «1» в бит INT0 регистра EIMSK разрешаем внешнее прерывание. При возникновении внешнего прерывания вызывается программа обработки прерывания ISR(INT0\_vect), в которой инкрементируется счетчик импульсов. В основной программе формируем таймер, который обновляется каждые 250 мс. За каждые 250 мс, если двигатель совершает вращение, происходит инкрементирование пере-

менной  $a$ . Поэтому в строке 15 используем следующую формулу для расчета скорости:

$$\text{speed} = a * 4 * 60 / 660,$$

где  $a$  – количество импульсов за 250 мс;

4 – коэффициент, чтобы получить количество импульсов за секунду;

60 – коэффициент, чтобы получить количество импульсов за минуту;

660 – количество импульсов за оборот;

speed – рассчитанная скорость.

Передаем полученную скорость в последовательный протокол порта.

## 13 EEPROM – постоянная память данных

Бывает так, что микроконтроллер отключают от питания, и для того чтобы ему запомнить данные о внешнем мире или записать важные системообразующие параметры, необходимо надежное хранилище, где такие данные будут храниться много лет без питания микроконтроллера. Для этой цели в микроконтроллере существует энергонезависимая память EEPROM.

EEPROM (англ. Electrically Erasable Programmable Read-Only Memory) – электрически стираемое перепрограммируемое ПЗУ – энергонезависимая память данных, в которой данные будут храниться даже при отключении питания микроконтроллера. В данной памяти можно хранить настройки выполнения программы, собранные данные для статистики работы устройства и другую полезную информацию. К примеру, собрав маленькую метеостанцию на микроконтроллере, в EEPROM на каждый день можно сохранять данные о температуре воздуха, давлении, силе ветра, а потом в любой момент считать эти собранные данные и провести статистические исследования.

EEPROM микроконтроллер Atmega328P обладает емкостью 1024 байта. Количество перезаписей для данного типа памяти составляет порядка 100 000, что в 10 раз больше, чем ресурс FLASH-памяти.

### Регистры EEPROM

В состав EEPROM микроконтроллера Atmega328 входят следующие регистры:

- **EEAR** (16 бит) – регистр адреса;
- **EEDR** – регистр данных;
- **EECR** – регистр управления.

## EEARH и EEARL – регистры адреса

В данный регистр загружается адрес ячейки, к которой будет производиться обращение. Регистр доступен как для записи, так и для чтения.

Bit	15	14	13	12	11	10	9	8	
0x22 (0x42)	-	-	-	-	-	-	-	EEAR8	EEARH
0x21 (0x41)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	X	

Рисунок 13.1 – Регистры адреса EEARH и EEARL

## EEDR – регистр данных

При записи в этот регистр загружаются данные, которые должны быть помещены в EEPROM, а при чтении в этот регистр помещаются данные, считанные с EEPROM.

Bit	7	6	5	4	3	2	1	0	
0x20 (0x40)	MSB							LSB	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 13.2 – Регистр данных EEDR

## EECR – регистр управления

Регистр используется для управления доступом к EEPROM-памяти.

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	-	-	EEMPM1	EEMPM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

Рисунок 13.3 – Регистр управления EECR

Бит 5 EEMPM1 и бит 4 EEMPM0 управляют режимами обновления EEPROM.

Таблица 13.1 – Управляющие биты режимами обновления EEPROM

EEMPM1	EEMPM0	Действия
0	0	Стирание старого значения и запись нового (3.4 мс)
0	1	Только стирание старого значения (1.8 мс)
1	0	Только запись нового значения (1.8 мс)
1	1	Резерв

Бит 3 EERIE разрешает прерывания по завершении записи в EEPROM при записи в него 1.

Бит 2 EEMPE разрешает запись в EEPROM, если в него записать 1 (сбрасывается в 0 через 4 машинных цикла).

Бит 1 EEPE управляет записью в EEPROM (если записать в него 1, то будет произведена запись данных из регистра EEDR в EEPROM по адресу EEAR).

Бит 0 EERE управляет чтением в EEPROM (если записать в него 1, то будет произведено чтение данных из EEPROM по адресу EEAR в регистр EEDR).

## Процедура записи в EEPROM

Процедура записи одного байта в EEPROM состоит из следующих этапов:

- 1) дождаться готовности EEPROM к записи данных (ждать, пока не сбросится флаг EEPE регистра EECR);
- 2) дождаться завершения записи во FLASH-память программ (ждать, пока не сбросится флаг SPEN регистра SPMCR);
- 3) загрузить байт данных в регистр EEDR, а требуемый адрес – в регистр EEAR (при необходимости);
- 4) установить лог. «1» во флаг EEMPE регистра EECR для разрешения записи в EEPROM;
- 5) записать в разряд EEPE регистра EECR лог. «1» в течение 4 машинных циклов. После установки этого разряда процессор пропускает 3 машинных цикла перед выполнением следующей инструкции.

Второй пункт введен из-за того, что запись в EEPROM не может выполняться одновременно с записью в FLASH-память. Поэтому перед выполнением записи в EEPROM следует убедиться, что программирование FLASH-памяти завершено. Если в программе отсутствует загрузчик, т.е. содержимое памяти программ не будет меняться, то второй шаг можно пропустить.

```
void EEPROM_write(unsigned int uiAddress, unsigned char
ucData){
    while(EECR & (1<<EEMPE)); // проверка готовности EEPROM
    EEARH = ((uiAddress & 0xF0) << 2); // регистр адреса H
    EEARL = uiAddress & 0x0F; // регистр адреса L
    EEDR = ucData; // регистр данных
    EECR |= (1<<EEMPE); // Разрешение записи в EEPROM
    EECR |= (1<<EEPE); // Запись в EEPROM
}
```

## Процедура чтения из EEPROM

Для чтения одного байта из EEPROM необходимо:

- 1) проконтролировать состояние флага EEPE. Дело в том, что, пока выполняется операция записи в EEPROM-память (флаг EEPE установлен), нельзя выполнять ни чтения EEPROM-памяти, ни изменения регистра адреса;
- 2) загрузить требуемый адрес в регистр EEARH и EEARL;
- 3) установить лог. «1» в разряд EERE регистра EECR.

Когда запрошенные данные будут помещены в регистр данных EEDR, произойдет аппаратный сброс этого разряда. Однако следить за состоянием разряда EERE для определения момента завершения операции чтения не требуется, так как операция чтения из EEPROM всегда выполняется за один машинный цикл. Кроме того, после установки лог. «1» в разряд EERE процессор пропускает 4 машинных цикла перед началом выполнения следующей инструкции.

```
unsigned char EEPROM_read(unsigned int uiAddress){
    while(EECR & (1<<EEPE)); // проверка готовности EEPROM
    EEARH = ((uiAddress & 0xF0) << 2); // регистр адреса H
    EEARL = uiAddress & 0x0F; // регистр адреса L
    EECR |= (1<<EERE); // чтение EEPROM
    return EEDR; // вывод значения
}
```

Листинг программы, позволяющей записать данные в EEPROM, а затем считать их и передать в монитор последовательного порта, представлен ниже:

```
1 #include <EEPROM.h>
2 void setup() {
3     Serial.begin(9600);
4     // пишем 123 по адресу 555
5     EEPROM_write(555, 123);
6     Serial.println(EEPROM_read(555)); // выведет 123
7 }
8 void loop() {
9     unsigned char EEPROM_read(unsigned int uiAddress){
10    while(EECR & (1<<EEPE)); // проверка готовности
    EEPROM
11    EEARH = ((uiAddress & 0xF0) << 2); // регистр адреса H
12    EEARL = uiAddress & 0x0F; // регистр адреса L
13    EECR |= (1<<EERE); // чтение EEPROM
14    return EEDR; // вывод значения
15 }
16 void EEPROM_write(unsigned int uiAddress, unsigned char
    ucData){
17    while(EECR & (1<<EEPE)); // проверка готовности
    EEPROM
18    EEARH = ((uiAddress & 0xF0) << 2); // регистр адреса H
19    EEARL = uiAddress & 0x0F; // регистр адреса L
20    EEDR = ucData; // регистр данных
21    EECR |= (1<<EEMPE); // Разрешение записи в EEPROM
22    EECR |= (1<<EEPE); // Запись в EEPROM
23 }
```

Для упрощения работы с EEPROM применяется библиотека EEPROM.h. В данной библиотеке используются следующие функции:

- **EEPROM.write(адрес, данные)** – пишет данные (только byte!) по адресу;
- **EEPROM.update(адрес, данные)** – обновляет байт данных, находящийся по адресу;
- **EEPROM.read(адрес)** – читает и возвращает байт данных, находящийся по адресу;
- **EEPROM.put(адрес, данные)** – записывает данные любого типа (типа переданной переменной) по адресу;
- **EEPROM.get(адрес, данные)** – читает данные по адресу и сам записывает их в данные – указанную переменную;
- **EEPROM[]** – библиотека позволяет работать с EEPROM-памятью как с обычным массивом типа byte (uint8\_t).

```
1 #include <EEPROM.h>
2 void setup() {
3   Serial.begin(9600);
4   // пишем 555 по адресу 123
5   EEPROM.update(555, 123);
6   Serial.println(EEPROM.read(555)); // выведет 123
7   Serial.println(EEPROM[555]);    // выведет 123
8 }
9 void loop() {}
```

## 14 Дисплей LCD1602

Отображение информации необходимо практически в каждом проекте при программировании микроконтроллера. Если компьютер доступен, то мы всегда можем передать информацию в последовательный протокол передачи данных, а затем или вывести в монитор последовательного порта, или считать сторонними программами интерфейса. Но что делать, если доступа к компьютеру нет, а выводить информацию необходимо для пользователей, которые эксплуатируют ваш продукт? Можно использовать дисплей LCD1602, представленный на рисунке 14.1.

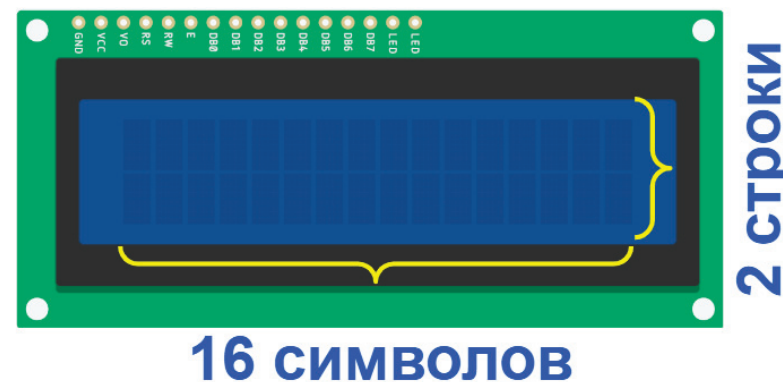


Рисунок 14.1 – Дисплей LCD1602

Представленный дисплей содержит 2 строки по 16 символов в каждой. Данный дисплей обладает следующими возможностями:

- символьный тип отображения, есть возможность загрузки символов;
- светодиодная подсветка;
- контроллер HD44780;
- напряжение питания 5 В;
- формат 16×2 символов;

Назначение портов для подключения дисплея:

- DB0–DB7 – отвечают за входящие/исходящие данные;
- RS – высокий уровень сигнала, поданный на данный вход, означает, что сигналы на выводах DB0–DB7 являются данными, низкий – командой;
- R/W – определяет направление данных (чтение/запись). Если только выводим данные, устанавливаем лог. «0»;
- E – импульс длительностью не менее 500 мс на этом выводе определяет сигнал для чтения/записи данных с выводов DB0–DB7, RS и W/R;
- V0 – используется для задания контраста изображения;
- LEDA, LEDK – питание подсветки (анод и катод);
- VSS – земля;
- VDD – питание ЖК-индикатора.

Подключим дисплей согласно схеме, представленной на рисунке 14.2.

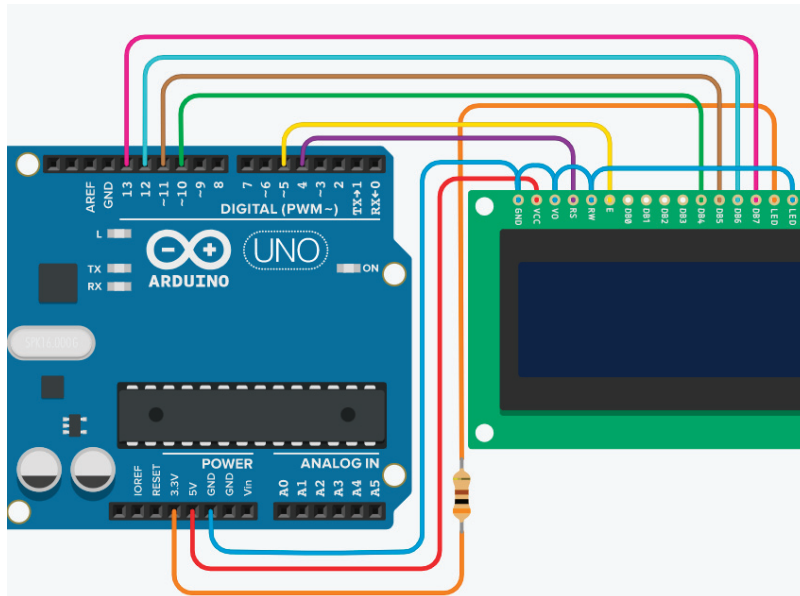


Рисунок 14.2 – Схема подключения дисплея

Для удобства работы с дисплеем LCD1602 разработчики предлагают библиотеку **LiquidCrystal.h**. Подробно познакомиться с библиотекой можно на официальном сайте: <https://www.arduino.cc/reference/en/libraries/liquidcrystal/>.

Для знакомства с дисплеем воспользуемся командами данной библиотеки:

**LiquidCrystal lcd(4, 5, 10, 11, 12, 13)** – создает объект lcd библиотеки LiquidCrystal, где 4, 5, 10, 11, 12, 13 – номера контактов платы разработчика, подключенные соответственно к LCD дисплею: RS, E, D4, D5, D6, D7;

**lcd.begin()** – команда инициализации дисплея;

**lcd.setCursor()** – команда устанавливает курсор в заданную позицию;

**lcd.print()** – команда выводит информацию на экран дисплея.

Кроме стандартных символов на дисплее можем формировать собственные пользовательские символы. Для этого воспользуемся командой **createChar()**:

`lcd.createChar(num, data),`

где lcd – переменная типа LiquidCrystal;

num – номер создаваемого символа (0 до 7);

data – данные символьных пикселей.

Создает пользовательский символ (глиф) для использования на жидкокристаллическом дисплее. Поддерживаются до восьми символов 5×8 пикселей (нумерация с 0 до 7). Создание каждого пользовательского символа определяется массивом из восьми байтов – один байт для каждой строки. Пять младших значащих битов каждого байта определяют пиксели в этой строке. Чтобы вывести пользовательский символ на экран, используйте функцию write() с номером символа в качестве параметра.

Пример глифа и массив, его хранящий, представлены на рисунке 14.3.



Для создания глифа можно воспользоваться информационным ресурсом по ссылке [https://radioaktiv.ru/custom-character\\_generator\\_for\\_hd44780.html](https://radioaktiv.ru/custom-character_generator_for_hd44780.html).

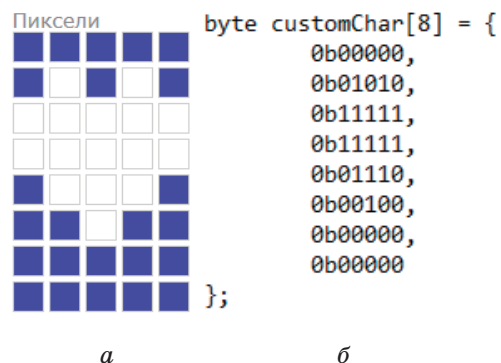


Рисунок 14.3 – Глиф размером 5×8 пикселей (а) и массив пикселей (б)

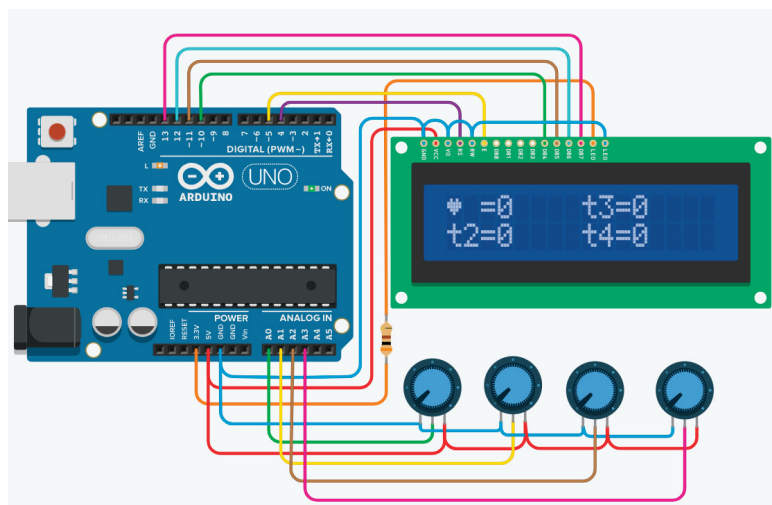


Рисунок 14.4 – Схема экспериментальной установки

Выведем на экран показания с четырех аналоговых портов (A0, A1, A2, A3), к соответствующим пинам подключены потенциометры, которые изменяют уровень напряжения. Дисплей подключен согласно рекомендуемой схеме. Экспериментальная установка представлена на рисунке 14.4.

Рассмотрим листинг программы, отображающий информацию на LCD-дисплее:

```
1 #include <LiquidCrystal.h>
2 unsigned long time1 =0;
3 #define T_period1 250
4 LiquidCrystal lcd(4, 5, 10, 11, 12, 13); // инициализация
  объекта lcd библиотеки
5 byte customChar1[8] = {
6 0b00000,
7 0b01010,
8 0b11111,
9 0b11111,
10 0b01110,
11 0b00100,
12 0b00000,
13 0b00000
14 };
15 void setup()
16 {
17 lcd.begin(16, 2);
18 lcd.createChar(0, customChar1);
19 }
20 void loop()
21 {
22 if (millis()-time1>=T_period1)
23 {
24 time1 = millis();
25 lcd.clear();
26 }
27 else
28 {
29 lcd.setCursor(0,0);
```

```

30 lcd.write(byte(0));
31 lcd.setCursor(2,0);
32 lcd.print(«=»);
33 lcd.setCursor(3,0);
34 lcd.print(analogRead(A0));
35 lcd.setCursor(8,0);
36 lcd.print(«t3»);
37 lcd.setCursor(10,0);
38 lcd.print(«=»);
39 lcd.setCursor(11,0);
40 lcd.print(analogRead(A2));
41 lcd.setCursor(0,1);
42 lcd.print(«t2»);
43 lcd.setCursor(2,1);
44 lcd.print(«=»);
45 lcd.setCursor(3,1);
46 lcd.print(analogRead(A1));
47 lcd.setCursor(8,1);
48 lcd.print(«t4»);
49 lcd.setCursor(10,1);
50 lcd.print(«=»);
51 lcd.setCursor(11,1);
52 lcd.print(analogRead(A3));
53 }
54 }

```

В 1-й строке программы подключаем библиотеку LiquidCrystal. Во 2-й строке задаем переменную time, необходимую для настройки таймера для очистки дисплея. В 3-й строке, используя функцию define, вводим параметр T\_period1, который отвечает за период обновления дисплея. В 4-й строке производим инициализацию объекта lcd библиотеки LiquidCrystal, указав, что пины платы разработчика 4, 5, 10, 11, 12, 13 будут подключены к выводам дисплея RS, E, D4, D5, D6, D7 соответственно. С 5-й по 14-ю строчку формируем одномерный массив customChar1[8] из 8 символов, заполненный параметрами, представленными в битовой форме записи.

В функции setup производим активацию дисплея. Строка 17 lcd.begin(16, 2) инициализирует дисплей, в котором действуем 16 символов и 2 строки. С помощью функции lcd.createChar в 18-й строке создаем 0-й пользовательский символ (глиф), описанный массивом customChar1. Далее переходим в основной цикл loop, где в 22-й строке с использованием функции if ...else... сформирован таймер, настроенный на обновление с периодом T\_period1, также после обновления таймера происходит очистка экрана (25 программы). Данное очищение экрана сделано для корректного отображения информации. Убедиться в этом можно, если закомментировать 25-ю строку и, запустив программы, изменить положение потенциометра от максимального значения до минимального. Пока таймер не сбросится, выполняем строки программы с 29-й по 52-ю. Перед выводом информации в 29-й строке программы устанавливаем курсор в позицию 0-й символ, 0-я строка, lcd.setCursor(0,0). Далее в 30-й строке вводим на экран 0-й пользовательский символ lcd.write(byte(0)). Далее в строке 31 lcd.setCursor(2,0) передвигаем курсор в позицию 2-й символ, 0-я строка и в 32-й строке программы выводим символ «=», и после этого в строке 33 lcd.setCursor(3,0) передвигаем курсор в позицию 3-й символ, 0-я строка и в 34-й строке выводим данные, считанные с аналогового порта A0 lcd.print(analogRead(A0)). После реализации данных строк на экране появляется картинка, представленная на рисунке 14.5.



Рисунок 14.5 – Результат работы кода

Аналогичные перемещения курсора по дисплею и вывод информации на экран осуществляются в строках 35–52. Результат работы программы представлен на рисунке 14.6.

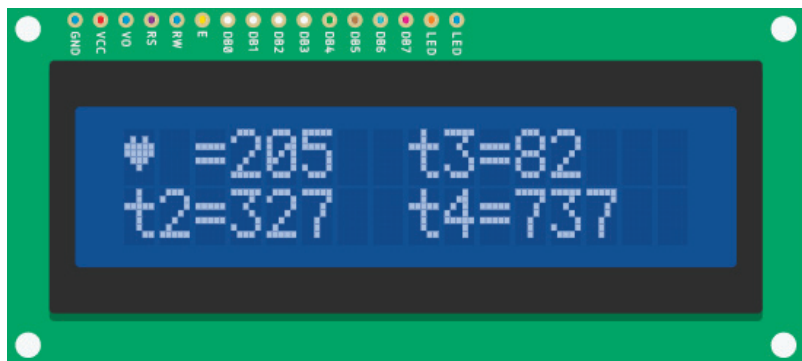


Рисунок 14.6 – Результат работы программы

## 15 Создание игры с использованием дисплея LCD1602

### Разработка стратегии игры

Разрабатываем игру с использованием микроконтроллера, дисплея LCD1602 и тактовой кнопки. Электрическая схема компонентов представлена на рисунке 15.1. Герою необходимо перепрыгивать через преграды, если герой перепрыгнул через препятствие или миновал его, ведем счет. Увеличиваем счет каждый раз, когда герой миновал препятствие. Если герой миновал 5 преград, увеличиваем скорость его перемещения. Если герой столкнулся с препятствием, то необходимо вывести слово GAME OVER, обнулить счет и вернуть скорость на начальную. Управлять роботом можно с помощью одной кнопки: если кнопка нажата, герой прыгает, если кнопка не нажата, герой перемещается без прыжка. Преграды в игре меняются после того, как они миновали героя и скрылись с экрана.

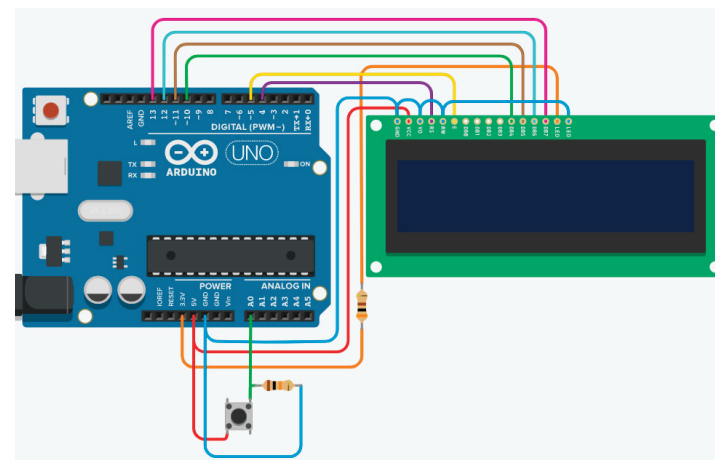


Рисунок 15.1 – Электрическая схема аппаратной части игры

Листинг программы, реализующие описанный алгоритм выше, представлен ниже:

```
1 #include <LiquidCrystal.h>
2 unsigned long time =0;
3 #define T_period 50
4 unsigned long time1 =0; // переменная для настройки Таймера 1
5 unsigned long time2 =0; // переменная для настройки Таймера 2
6 #define T_period1 300
7 int T_period2 = 300; // период, отвечающий за скорость в игре
8 LiquidCrystal LCD(4, 5, 10, 11, 12, 13);
9 int data =0;
10 byte p=0; // баллы
11 byte d =0; // ордината героя
12 byte x =15; // абсцисса препятствия
13 byte y =0; // ордината препятствия
14 byte i =0; // выбор препятствия
15 byte robot[8] = {
16 0b00100,
17 0b01110,
18 0b00100,
19 0b11111,
20 0b10101,
21 0b00100,
22 0b01010,
23 0b01010
24 };
25 byte grib[8] = {
26 0b00000,
27 0b00000,
28 0b00000,
29 0b00000,
30 0b01110,
31 0b11111,
32 0b00100,
33 0b00100
```

```
34 };
35 byte tree[8] = {
36 0b00100,
37 0b01110,
38 0b11111,
39 0b00100,
40 0b01110,
41 0b11111,
42 0b00100,
43 0b00100
44 };
45 byte snej[8] = {
46 0b00000,
47 0b00000,
48 0b10101,
49 0b01110,
50 0b11011,
51 0b01110,
52 0b10101,
53 0b00000
54 };
55 void setup()
56 {
57 LCD.begin(16, 2);
58 LCD.createChar(0,robot);
59 LCD.createChar(1,grib);
60 LCD.createChar(2,tree);
61 LCD.createChar(3,snej);
62 }
63 void loop()
64 {
65 i = random(1,4);
66 if(i==3) y=0;
67 else y=1;
68 while (x>0)
69 {
70 //Рисуем препятствие
71 LCD.setCursor(x,y);
72 LCD.write(byte(i));
```

```

73 // управление героем
74 if(digitalRead(A0))
75 {
76   d = 0;
77 }
78 if(!digitalRead(A0))
79 {
80   d = 1;
81 }
82 // Рисуем героя
83 LCD.setCursor(4,d);
84 LCD.write(byte(0));
85 //вывод счета
86 LCD.setCursor(0,0);
87 LCD.print(p);
88 // проверяем столкновение
89 if ((x==4) && (d==y))
90 {
91   LCD.setCursor(4,1);
92   LCD.print(«GAME OVER»);
93   p=0;
94   T_period2 = 300;
95   delay(1000);
96   LCD.clear();
97   x=15;
98 }
99 //подсчет баллов
100 if ((x==4) && !(d==y)&& (millis()-time2 >= T_
    period1+10 ))
101 {
102   time2 = millis();
103   p++;
104   if (p==2) T_period2=200;
105   if (p==4) T_period2=150;
106 }
107 if (millis()-time1 >= T_period2 )
108 {
109   time1= millis();
110   x--;

```

```

111 }
112 if (millis()-time >= T_period )
113 {
114   time= millis();
115   LCD.clear();
116 }
117 }
118 x=15;
119 }

```

Рассмотрим основные моменты игры.

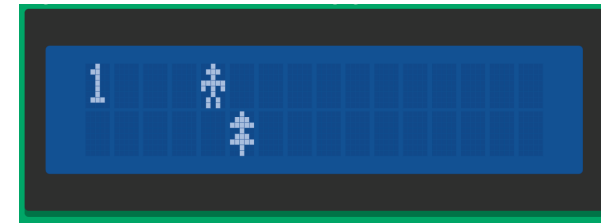


Рисунок 15.2 – Экран игры

Герой – в нашем примере это робот, реализованный в виде глифа. Препрады (рисунок 15.3): дерево, гриб и снежинка, также реализованы в виде глифов, при этом дерево и гриб перемещаются внизу экрана, а снежинка – вверх экрана [15].

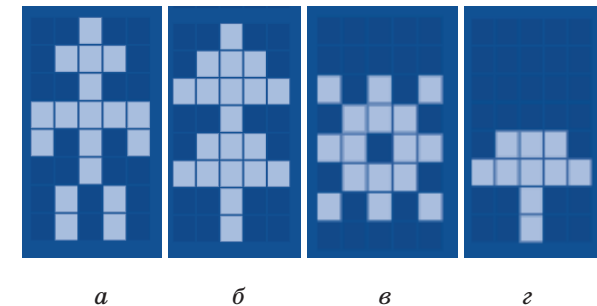


Рисунок 15.3 – Действующие лица игры: *а* – робот; *б* – дерево; *в* – снежинка; *г* – гриб

## Реализация движения в игре

Осуществляется двумя способами: прыжок героя и движение препятствия.

### Прыжок героя

Переход героя из 1-й строки дисплея в 0-ю строку дисплея при нажатии на кнопку, если кнопка не нажата, герой располагается в 0-й строке дисплея.

Данный алгоритм реализуется в 74-й и 81-й строках программы. Считываем данные с порта A0 и проверяем, если параметр равен лог. «1», то в переменную d, отвечающую за ординату героя, записываем ноль – номер 0-й строки, если на порт подан лог. «0» – то в d записываем единицу – номер 1-й строки.

```
1  if(digitalRead(A0))
2  {
3      d = 0;
4  }
5  if(!digitalRead(A0))
6  {
7      d = 1;
8  }
```

### Движение препятствий

Герой неподвижен по отношению к преградам и располагается всегда на 5-м символе дисплея 1-й или 0-й строки. Перемещаются преграды из 15-го символа в 0-й символ, таким образом создается эффект движения героя.

Для управления героем и препятствиями в программе вводятся переменные:

- byte d =0; // ордината героя;
- byte x =15; // абсцисса препятствия;
- byte y =0; // ордината препятствия.

Изменение x начинается с 15-го символа, и пока значения больше 0, происходит уменьшение на единицу с периодом, указанным в переменной T\_period2.

```
1  while (x>0)
2  {
3  ...
4  if (millis()-time1 >= T_period2 )
5  {
6      time1= millis();
7      x--;
8  }
9  ...
10 }
11 x=15;
```

Создание объектов реализуется в виде глифа с помощью матрицы элементов, описанных в строках с 35-й по 54-ю программы. Сами пользовательские символы создаются в 58-й и 61-й строках:

```
1  LCD.createChar(0,robot);
2  LCD.createChar(1,grib);
3  LCD.createChar(2,tree);
4  LCD.createChar(3,snej);
```

Причем герою присваивается 0-й порядковый номер. Это сделано для того, чтобы при произвольном выборе препятствия не был выбран 0-й элемент.

```
1  i = random(1,4);
2  if(i==3) y=0;
3  else y=1;
```

Произвольный выбор препятствия осуществляется в 65-й строке, где переменная i отвечает за номер выбранного препятствия (1 – гриб, 2 – дерево и 3 – снежинка). 66-я и 67-я строки программы отвечают за положение препятствия. Если номер выбранного препятствия равен 3, т.е. это снежинка, то

параметру у, отвечающему за номер строки дисплея размещения препятствия, присваивается 1, а если это гриб или дерево – присваивается 0.

## Проверка столкновения

Проверка столкновения осуществляется в 89 строке программы по условию

```
if ((x==4) && (d==y)),
```

где x – абсцисса препятствия;

d – ордината героя;

y – ордината препятствия.

Таким образом, фиксируем столкновение, если ординаты и абсциссы героя и препятствия совпадают, тогда выводим надпись GAME OVER в 91-й и 92-й строках программы. Кроме того, необходимо сбросить счет, обнуляем переменную p в 93-й строке программы, возвращаем значение 15 ординате препятствия в строке 97 и возвращаем значение периода T\_period2 к 300 (данный параметр отвечает за скорость перемещения объекта). Делаем паузу в 1 секунду (95-я строка) и очищаем дисплей (96-я строка).

```
if ((x==4) && (d==y))
```

```
1 {
2   LCD.setCursor(4,1);
3   LCD.print(«GAME OVER»);
4   p=0;
5   T_period2 = 300;
6   delay(1000);
7   LCD.clear();
8   x=15;
9 }
```

## Проверка удачного выполнения миссии и увеличение скорости

Проверка удачного выполнения миссии осуществляется в 100-й строке программы по условию

```
if ((x==4) && !(d==y)&& (millis()-time2 >= T_period1+10)),
```

где x – абсцисса препятствия;

d – ордината героя;

y – ордината препятствия.

Таким образом, фиксируем успешное выполнение миссии, если абсциссы героя и препятствия совпадают, а ординаты не совпадают, тогда переменную p, отвечающую за счет, увеличиваем на единицу. Кроме того, в условии проверки миссии добавлено условие проверки таймера, и период обработки таймера настраивается из условия, что T\_period1+10 > T\_period2. Это сделано для того, чтобы в момент анализа успешного освоения миссии переменная p не успевала многократно инкрементировать.

Сделайте эксперимент самостоятельно: уберите в 100-й строке проверку (millis()-time2 >= T\_period1+10, и вы увидите, что в момент прыжка переменная p успевает увеличиться более чем в 1 раз!

В 104-й и 105-й строках при значениях, равных 2 и 4 соответственно, происходит уменьшение периода T\_period2, отвечающего за скорость движения препятствия.

## Вывод героя, препятствия и счета на дисплей

Прорисовка героя и других элементов осуществляется параметрически.

За вывод героя на экран отвечают 137-я и 138-я строки программы:

```
1 LCD.setCursor(4,d);
2 LCD.write(byte(0)),
```

где 4 – абсцисса героя, она всегда равна 4, так герой у нас находится всегда на 4-м символе дисплея;

d – ордината героя, изменяется программно при анализе прыжка героя;

0 – герой в программе – это 0-й пользовательский символ.

За вывод препятствия на экран отвечают 125-я и 126-я строки программы:

```
1 //Рисуем препятствие
2 LCD.setCursor(x,y);
3 LCD.write(byte(i)),
```

где x – абсцисса препятствия;

y – ордината препятствия;

i – номер пользовательского символа (1 – гриб, 2 – дерево, 3 – снежинка).

## Обновление экрана

После того как проверили координаты, нарисовали героев и препятствия, в основном цикле движения необходимо обязательно обновить экран, т.е. наша игра перемещается по кадрам. Таким образом, смену экрана необходимо осуществлять с определенной периодичностью, для этого в строках 112 и 116 запускается таймер, при срабатывании которого осуществляется очистка экрана в 115-й строке программы:

```
1 if (millis()-time >= T_period )
2 {
3   time= millis();
4   LCD.clear();
5 }
```

## Библиографический список

1 Анализ рынка микроконтроллеров. Прошлое и настоящее : [Электронный ресурс]. – URL: <https://commarketru.com/mikrokontrollery-proshloe-i-nastoyashhee/> (дата обращения: 12.12.2022).

2 Правильное подключение микроконтроллера : [Электронный ресурс]. – URL: <https://radio-magic.ru/microcontrollers/400-podklyuchenie-mikrokontrollera> (дата обращения: 12.12.2022).

3 Подключение матричной клавиатуры к микроконтроллерам AVR : [Электронный ресурс]. – URL: <https://radioparty.ru/programming/avr/c/455-lesson-matrix-keyboard> (дата обращения: 12.12.2022).

4 Что такое дребезг контактов и как его устранить? : [Электронный ресурс]. – URL: <https://www.asutpp.ru/drebezg-kontaktov.html> (дата обращения: 12.12.2022).

5 Timer/Counter1 Atmega328 : [Электронный ресурс]. – URL: <http://integrator.adior.ru/index.php/robototekhnika/184-timer-counter1-atmega328> (дата обращения: 12.12.2022).

6 Что такое таймер? : [Электронный ресурс]. – URL: <https://habr.com/ru/post/453276/> (дата обращения: 12.12.2022).

7 Реализация ШИМ : [Электронный ресурс]. – URL: <https://radioparty.ru/prog-avr/program-c/240-lesson8?showall=1&limitstart=> (дата обращения: 12.12.2022).

8 сторожевой таймер // Оборудование Технологии Разработка : [Электронный ресурс]. – URL: <http://mypractic.ru/urok-16-povyshenie-nadezhnosti-programm-dlya-arduino-storozhevoj-tajmer.html> (дата обращения: 12.12.2022).

9 Зачем нужен watchdog (сторожевой таймер)? : [Электронный ресурс]. – URL: [https://geekmatic.in.ua/watchdog\\_arduino\\_uno\\_sleep](https://geekmatic.in.ua/watchdog_arduino_uno_sleep) (дата обращения: 12.12.2022).

10 Atmega328P : [Электронный ресурс]. – URL: [https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-Atmega328P\\_Datasheet.pdf](https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-Atmega328P_Datasheet.pdf) (дата обращения: 12.12.2022).



11 Интерфейс передачи данных – UART //3DiYshop : [Электронный ресурс]. – URL: <https://3d-diy.ru/wiki/arduino-moduli/interfeysy-peredachi-dannykh-uart/> (дата обращения: 12.12.2022).

12 UART – Последовательный интерфейс передачи данных // Вольтик : [Электронный ресурс]. – URL: <https://volti.ru/wiki/uart-interface/> (дата обращения: 12.12.2022).

13 EXINT или внешние прерывания // Программирование микроконтроллеров : [Электронный ресурс]. – URL: <https://narodstream.ru/avr-urok-42-exint-ili-vneshnie-preryvaniya/> (дата обращения: 12.12.2022).

14 AttachInterrupt // Аппаратная платформа Arduino : [Электронный ресурс]. – URL: <https://arduino.ru/Reference/AttachInterrupt> (дата обращения: 12.12.2022).

15 Генератор символов для LCD HD44780 : [Электронный ресурс]. – URL: [https://radioaktiv.ru/custom\\_character\\_generator\\_for\\_hd44780.html](https://radioaktiv.ru/custom_character_generator_for_hd44780.html) (дата обращения: 12.12.2022).

16 Библиотека LiquidCrystal : [Электронный ресурс]. – URL: <http://developer.alexanderklimov.ru/arduino/liquidcrystal.php#leftToRight> (дата обращения: 12.12.2022).

17 Евтисеев А.В. Микроконтроллеры AVR семейства Tiny и Mega фирмы ATMEGA (+CD). – М.: Додэка-XXI, 2004.

18 Гостева Е.А. Солнечный парус : метод. указания к разработке проекта в виртуальной среде / Е.А. Гостева, М.Н. Давыдкин. – М.: Изд. Дом НИТУ «МИСиС», 2020. – 57 с.

19 Давыдкин М.Н. Мехатроника и робототехника Arduino. Мобильный робот : метод. указания / М.Н. Давыдкин. – М.: Изд. Дом НИТУ «МИСиС», 2019. – 22 с.

20 Давыдкин М.Н. Мехатроника и робототехника Arduino. Дистанционное управление : метод. указания / М.Н. Давыдкин. – М.: Изд. Дом НИТУ «МИСиС», 2019. – 28 с.

21 Давыдкин М.Н. Мехатроника и робототехника LEGO. От идеи до проекта : метод. указания / М.Н. Давыдкин. – М.: Изд. Дом НИТУ «МИСиС», 2019. – 24 с.

22 Давыдкин М.Н. Система хранения на основе интернет вещей и RFID-технологии / М.Н. Давыдкин // Наука и производство Урала. – 2018. – № 14. – С. 59–60.

*Учебное издание*

**Давыдкин Максим Николаевич**

## **ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ**

### **Методические указания**

Корректор *В.В. Демидова*  
Технический редактор *Т.В. Суханова*  
Верстальщик *Ю.Б. Пашкова*

---

Подписано в печать 30.12.22

Уч.-изд. л. 11

Формат 60 × 90 1/16

---

Университет науки и технологий МИСиС,  
119049, Москва, Ленинский пр-кт, д. 4, стр. 1

Издательский Дом НИТУ «МИСиС»,  
119049, Москва, Ленинский пр-кт, д. 2А  
Тел. 8 (495) 638-44-06

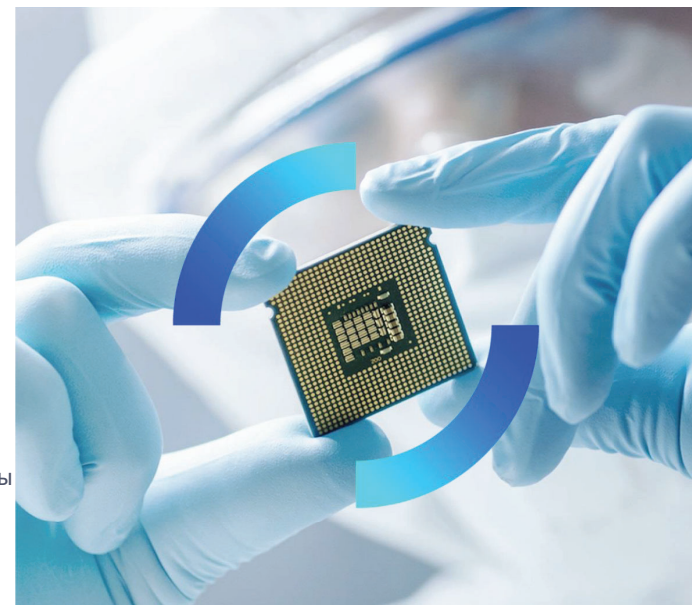
Отпечатано в типографии  
Издательского Дома НИТУ «МИСиС»,  
119049, Москва, Ленинский пр-кт, д. 4А  
Тел. 8 (495) 638-44-16, 8 (495) 638-44-43

Презентации, содержащие иллюстративный материал и инфографику для использования учителем во время занятия



## Программирование микроконтроллеров

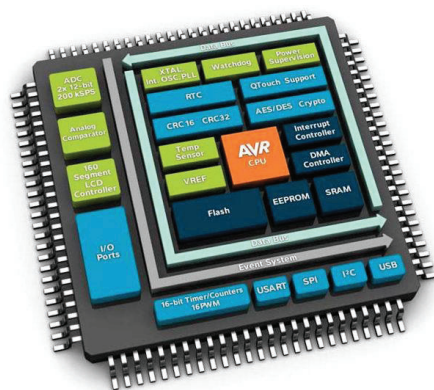
Введение в микроконтроллеры



## AVR микроконтроллер семейства ATMEGA

Микроконтроллер AVR семейства Mega являются 8 разрядными микроконтроллерами с RISC-архитектурой

**RISC (Reduced Instruction Set Computer)** – архитектура с сокращённой системой команд.



2

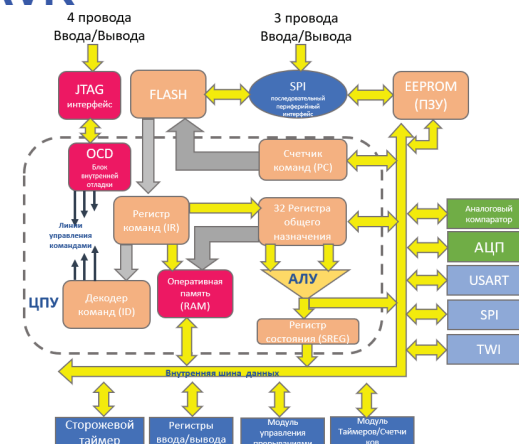
## Структурная схема AVR микроконтроллера

ЦПУ - это мозг микроконтроллера, который содержит в себе АЛУ, регистры и оперативную память.

**OCD (On-Chip Debugger)** - блок внутренней отладки;

К АЛУ подключен блок из 32-х регистров общего назначения (**32 General Purpose Registers** - регистровая память), каждый из которых представляет собою 1 байт памяти (8 бит).

**RAM (Random Access Memory)** - оперативная память процессора. В нее можно записывать данные из регистров, считывать данные в регистры, все операции с данными и расчеты производятся в регистрах.



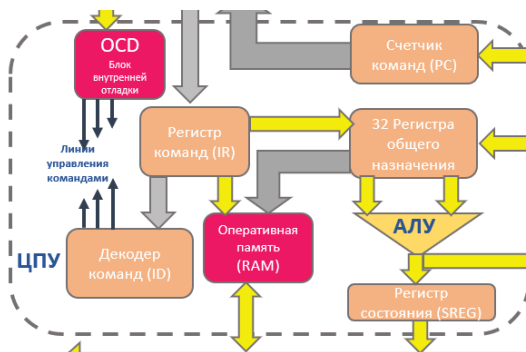
## Структурная схема AVR микроконтроллера

**ЦПУ** - это мозг микроконтроллера, который содержит в себе АЛУ, регистры и оперативную память.

**OCD** (On-Chip Debugger) - блок внутренней отладки;

К АЛУ подключен блок из 32-х регистров общего назначения (**32 General Purpose Registers** - регистровая память), каждый из которых представляет собою 1 байт памяти (8 бит).

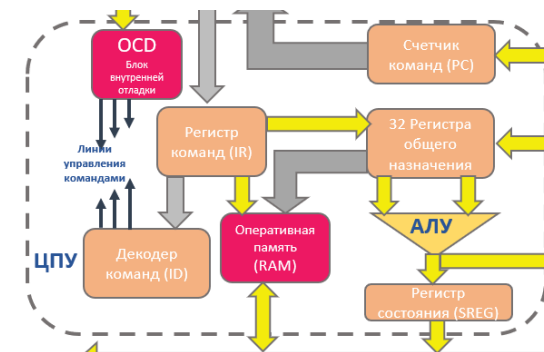
**RAM** (Random Access Memory) - оперативная память процессора. В нее можно записывать данные из регистров, считывать данные в регистры, все операции с данными и расчеты производятся в регистрах.



4

## Структурная схема AVR микроконтроллера

**АЛУ** - Арифметико-логическое устройство, которое синхронно с тактовым сигналом и опираясь на состояние счетчика команд (**Program Counter**) выбирает из памяти программ (**FLASH**) очередную команду и производит ее выполнение.



5

## Структурная схема AVR микроконтроллера

**FLASH** - перепрограммируемая память для сохранения программы;

**FLASH** - память программ, энергонезависимое ПЗУ(постоянное запоминающее устройство) что выполнено по технологии FLASH. Здесь хранится программа, которая будет исполняться блоком ALU микроконтроллера. Флеш-память чипа можно многократно перезаписывать, тем самым меняя или дополняя программный код для выполнения. Данный тип памяти может сохранять записанные в нее данные в течение 40 лет, а количество возможных циклов стирания/записи может достигать 10000.

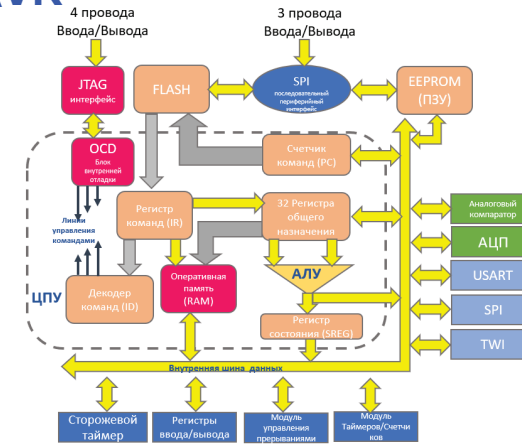


## Структурная схема AVR микроконтроллера

**EEPROM** (Electrically Erasable Programmable Read-Only Memory) - перепрограммируемое ПЗУ, энергонезависимая память;

**EEPROM** - энергонезависимая память данных в которой данные будут храниться даже при отключении питания микроконтроллера. В данной памяти можно хранить настройки выполнения программы, собранные данные для статистики работы устройства и другую полезную информацию.

Количество перезаписей для данного типа памяти составляет порядка 100000 что в 10 раз больше чем ресурс FLASH памяти.

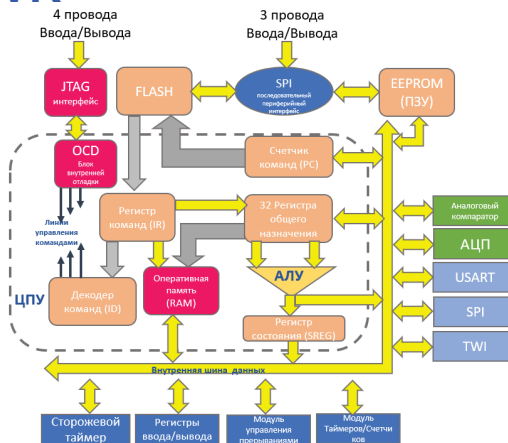


## Структурная схема AVR микроконтроллера

**Serial Peripheral Interface, SPI** - последовательный периферийный интерфейс (3 провода);

**Serial Peripheral Interface, SPI** - последовательный периферийный интерфейс (SPI) который зачастую применяется для обмена данными между несколькими микроконтроллерами со скоростью до нескольких МГц (нескольких миллионов тактов в секунду).

SPI интерфейс используется для внутрисхемного SPI программирования, по этому интерфейсу к микроконтроллеру подключается программатор.



## Структурная схема AVR микроконтроллера

**Analog Comparator** - данный блок сравнивает между собою два уровня сигнала и запоминает результат сравнения в определенном регистре, после чего сданный результат можно проанализировать и выполнить необходимые действия.

**A/D Converter** - данный блок преобразовывает аналоговое значение напряжения в цифровое значение, с которым можно работать в программе и на основе которого можно выполнять определенные действия.



## Структурная схема AVR микроконтроллера

**USART** - последовательный асинхронный интерфейс для обмена данными с другими устройствами. Есть поддержка протокола RS-232, благодаря чему микроконтроллер можно соединить для обмена данными с компьютером.

**TWI** - интерфейс для обмена данными по двухпроводной шине. К такой шине данных можно подключить до 128 различных устройств, используя две линии данных: тактовый сигнал (SCL) и сигнал данных (SDA). Интерфейс TWI является аналогом базовой версии интерфейса I2C.



## Структурная схема AVR микроконтроллера

**Watchdog Timer** - сторожевой или контрольный таймер, представляет собою систему контроля зависания устройства с последующим его перезапуском.

поддержка протокола RS-232, благодаря чему микроконтроллер можно соединить для обмена данными с компьютером.

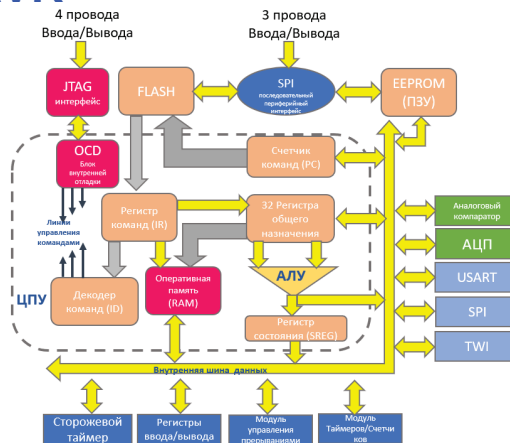
**I/O Ports, GPIO** - это набор блоков портов ввода/вывода к пинам которых можно подключить разнообразные датчики, исполняющие устройства и цепи. Количество пинов вход/выход, что идут от портов в микроконтроллере, может быть от 3 до 86.



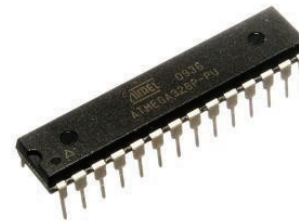
## Структурная схема AVR микроконтроллера

**Interrupts** - блок управления и реакции на прерывания, это блок который отвечает за реакцию и запуск на выполнение определенных функций при поступлении сигнала на определенные входы микроконтроллера или же по какому-то внутреннему событию (например тиканью таймера). Под каждое прерывание разрабатывается и записывается в память отдельная подпрограмма.

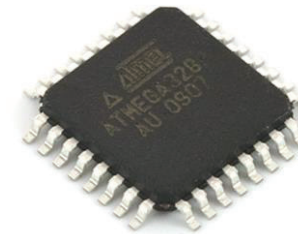
**Timers/Counters** - набор таймеров и счетчиков. Микроконтроллер, как правило, содержит в себе от одного до четырех таймеров и счетчиков.



## Корпус микроконтроллера Atmega328



ATmega328 – DIP (28pin)



ATmega328 – TQFP (32 pin)



ATmega328 – VQFN-(32 pin)



## Пример маркировки AVR микроконтроллера



## Пример маркировки AVR микроконтроллера



## Пример маркировки AVR микроконтроллера



Семейство  
MEGA

## Пример маркировки AVR микроконтроллера



32KB  
FLASH

## Пример маркировки AVR микроконтроллера



Модификация 8

## Пример маркировки AVR микроконтроллера



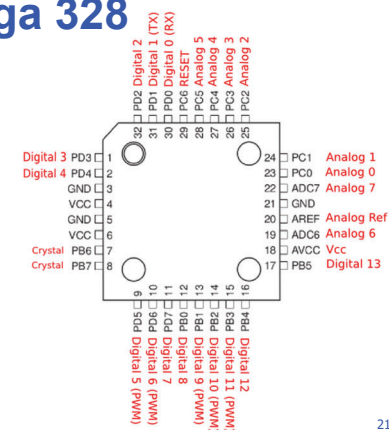
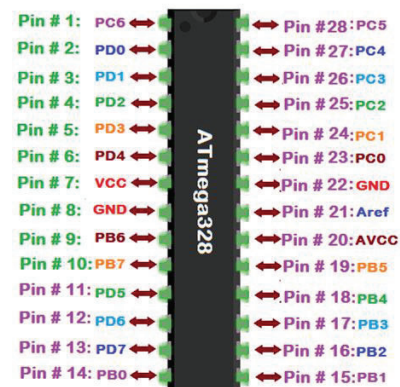
Корпус DIP

## Пример маркировки AVR микроконтроллера



Диапазон температур от -40 до +85

## Назначение портов микроконтроллера Atmega 328



## Семейство микроконтроллеров Мега

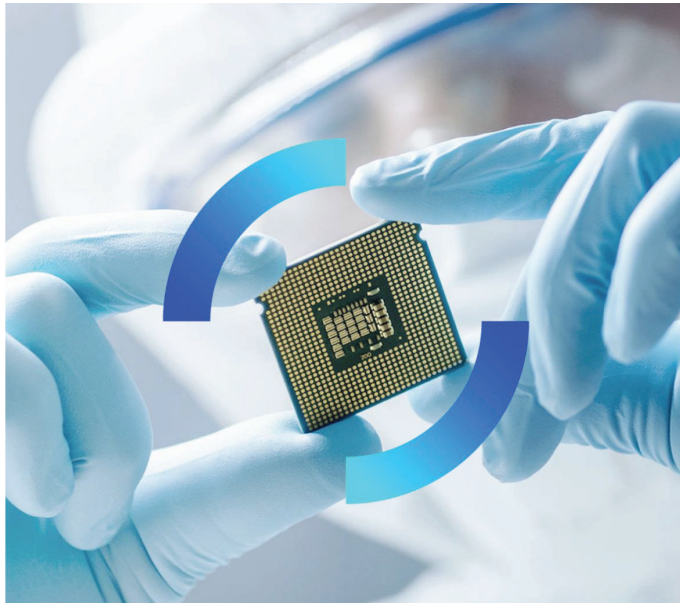
Название	Flash-память, кбайт	EEPROM, байт	ОЗУ, байт	Порты В / В	Дополнительно	Интерфейсы	8/ 16-битные таймеры	ШИМ, каналов	Аналоговый компаратор	10-битный АЦП, каналов	RTC	Сторожевой таймер	Апп. Умножитель	Напряжение питания, В	Макс. тактовая частота, МГц	Тип корпуса
ATmega48P	4	256	512	23	Общ. назн. (GP)	UART, 2 SPI, I <sup>2</sup> C	2/1	6	+	8	+	+	+	1,8–5,5	20	TQFP32, MLF32
ATmega88P	8	512	1024	23	GP	UART, 2 SPI, I <sup>2</sup> C	2/1	6	+	8	+	+	+	1,8–5,5	20	TQFP32, MLF32
ATmega168P	16	512	1024	23	GP	UART, 2 SPI, I <sup>2</sup> C	2/1	6	+	8	+	+	+	1,8–5,5	20	TQFP32, MLF32
ATmega328P	32	1024	2048	23	GP	UART, 2 SPI, I <sup>2</sup> C	2/1	6	+	8	+	+	+	1,8–5,5	20	TQFP32, MLF32
ATmega164D**	16	512	1024	32	GP	2 USART, SPI, I <sup>2</sup> C	2/1	6	+	8	+	+	+	1,8–5,5	20	DIP40, TQFP44, MLF44

**Спасибо  
за внимание!**

Ленинский проспект, д. 4  
 Москва, 119049  
 тел. +7 (495) 955-00-32  
 e-mail: kancela@misis.ru  
 misis.ru



**Программируемые  
микроконтроллеры**  
Платформа разработчика



## Минимальный набор для работы с микроконтроллером

**ATMEGA328P**  
DIP PACKAGE

RESET	1	28	A5	SCL
RX	D0	2	A4	SDA
TX	D1	3	A3	
INT 0	D2	4	A2	
INT 1	D3	5	A1	
D4	6	23	A0	
VCC	7	22	GND	
GND	8	21	AREF	
XTAL	9	20	VCC	
XTAL	10	19	D13	SCK
-D5	11	18	D12	MISO
-D6	12	17	D11	MOSI
D7	13	16	D10	
D8	14	15	D9	

## Микроконтроллер

[https://aliexpress.ru/item/32831839565.html?spm=a2g2w.productlist.list.0.2d9f1855mE2PxK&sku\\_id=65010990571](https://aliexpress.ru/item/32831839565.html?spm=a2g2w.productlist.list.0.2d9f1855mE2PxK&sku_id=65010990571)

## Минимальный набор для работы с микроконтроллером



### Источник питания

[https://aliexpress.ru/item/1005001936119656.html?spm=a2g2w.productlist.list.1.229e3c99qkYh0l&sku\\_id=12000018156495725](https://aliexpress.ru/item/1005001936119656.html?spm=a2g2w.productlist.list.1.229e3c99qkYh0l&sku_id=12000018156495725)

## Минимальный набор для работы с микроконтроллером

### Среда разработки

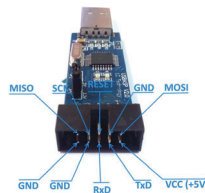


Atmel Studio

[https://cxem.net/software/soft\\_mcu.php](https://cxem.net/software/soft_mcu.php)

## Минимальный набор для работы с микроконтроллером

### Программатор



[https://aliexpress.ru/item/32670511994.html?spm=a2g2w.prouductlist.list.0.7bf55c0dy6EoJy&sku\\_id=66714631038](https://aliexpress.ru/item/32670511994.html?spm=a2g2w.prouductlist.list.0.7bf55c0dy6EoJy&sku_id=66714631038)

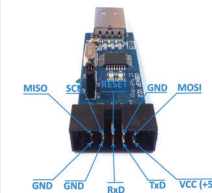
5

## Минимальный набор для работы с микроконтроллером



**ATMEGA328P**  
DIP PACKAGE

RESET	1	28	A5	SCL
RX	DO	2	A4	SDA
TX	D1	3	A3	
INT 0	D2	4	A2	
INT 1	-D3	5	A1	
D4	6	23	A0	
VCC	7	22	GND	
GND	8	21	AREF	
XTAL	9	20	VCC	
XTAL	10	19	D13	SCK
-D5	11	18	D12	MISO
-D6	12	17	D11	MOSI
D7	13	16	D10	
D8	14	15	D9	



Условно бесплатно



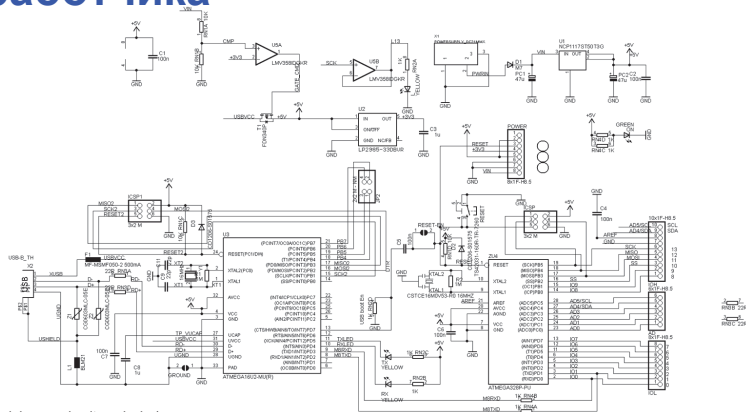
Atmel Studio

540 p

6



## Принципиальная схема платы разработчика



<https://chip.by/product/arduino-uno-kupit-v-minske/>

## Платформа Arduino UNO



Микроконтроллер Atmega328

Встроенный программатор

Источник питания на 3.3 и 5 вольт

Контактные площадки для подключения к пинам

микроконтроллера

Индикация питания

Встроенный DC-DC преобразователь

Бесплатное программное обеспечение

**300 р**

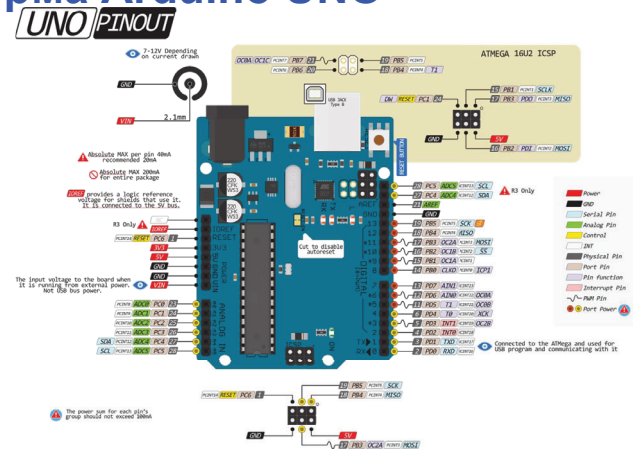
[https://aliexpress.ru/item/32831857729.html?item\\_id=32831857729&sku\\_id=65554550823&spm=a2g2w.productlist.li](https://aliexpress.ru/item/32831857729.html?item_id=32831857729&sku_id=65554550823&spm=a2g2w.productlist.list.0.24684fd9SmXJmS)

## Платформа Arduino UNO



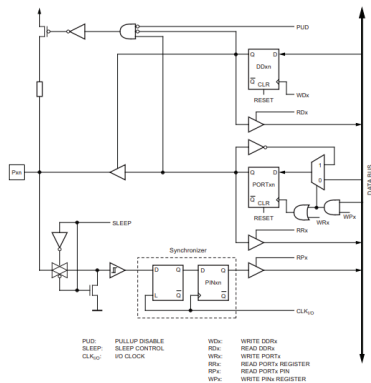
Микроконтроллер: АТмега328  
 Диапазон допустимого напряжения питания: 5-20 В  
 Рекомендуемое напряжение питания: 7-12 В  
 Количество цифровых вводов/выводов: 14  
 ШИМ: 6 цифровых пинов могут быть использованы как выводы ШИМ  
 Количество аналоговых выводов: 8  
 Максимальная сила тока: 40 mA с одного вывода и 500 mA со всех выводов.  
 Flash память: 32 кб  
 SRAM: 2 кб  
 EEPROM: 1 кб  
 Тактовая частота: 16 МГц

## Платформа Arduino UNO



# Atmega 328

## Регистры портов ввода/вывода GPIO



### PORTx – Data Register

Регистр данных. Режим управления состоянием вывода.

### DDRx – Data Direction Register

Регистр направления порта  
 DDRx<sub>y</sub>=0 — вывод работает как ВХОД.  
 DDRx<sub>y</sub>=1 вывод работает на ВЫХОД

### PINx – Input Pins Address

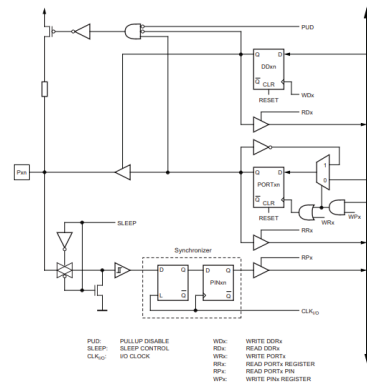
Регистр чтения. Из него можно только читать.  
 В регистре **PINx** содержится информация о **реальном** текущем логическом уровне на выводах порта.

<http://easyelectronics.ru/avr-uchebnyj-kurs-ustrojstvo-i-rabota-portov-vvoda-vyvoda.html>

# Atmega 328

## Регистры портов ввода/вывода GPIO

DDR<sub>x</sub><sub>y</sub>=0 — вывод работает как ВХОД.  
 DDR<sub>x</sub><sub>y</sub>=1 вывод работает на ВЫХОД



### PORTx – Data Register

Регистр данных. Режим управления состоянием вывода.

Когда ножка настроена на **выход**, то значение соответствующего бита в регистре PORTx определяет состояние вывода. Если **PORTx<sub>y</sub>=1** то на выводе лог1, если **PORTx<sub>y</sub>=0** то на выводе лог0.

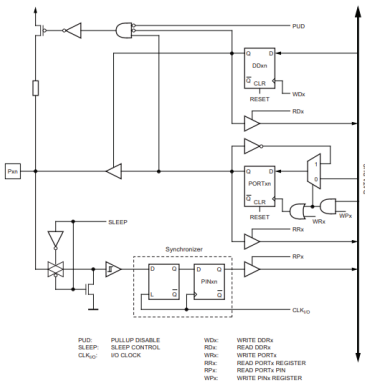
Когда ножка настроена на **вход**, то если **PORTx<sub>y</sub>=0**, то вывод в режиме **Hi-Z**. Если **PORTx<sub>y</sub>=1** то вывод в режиме **PullUp** с подтяжкой резистором в 100к до питания.

**Вход Hi-Z** — режим высокоимпендансного входа.

**Вход PullUp** — вход с подтяжкой.

<http://easyelectronics.ru/avr-uchebnyj-kurs-ustrojstvo-i-rabota-portov-vvoda-vyvoda.html>

# Atmega 328 Регистры портов ввода/вывода GPIO



**PORTB – The Port B Data Register**

Bit	7	6	5	4	3	2	1	0	PORTB
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

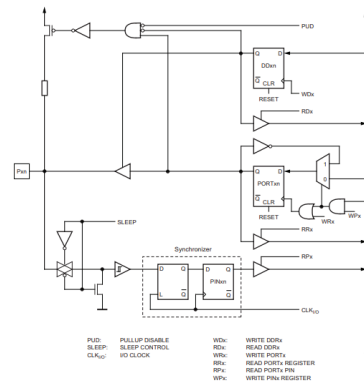
**DDRB – The Port B Data Direction Register**

Bit	7	6	5	4	3	2	1	0	DDRB
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**PINB – The Port B Input Pins Address**

Bit	7	6	5	4	3	2	1	0	PINB
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

# Atmega 328 Регистры портов ввода/вывода GPIO



**PORTC – The Port C Data Register**

Bit	7	6	5	4	3	2	1	0	PORTC
0x08 (0x28)	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

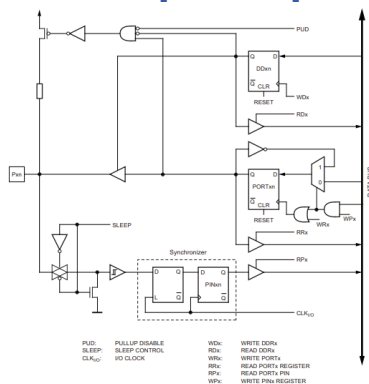
**DDRC – The Port C Data Direction Register**

Bit	7	6	5	4	3	2	1	0	DDRC
0x07 (0x27)	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**PINC – The Port C Input Pins Address**

Bit	7	6	5	4	3	2	1	0	PINC
0x06 (0x26)	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

# Atmega 328 Регистры портов ввода/вывода GPIO



## PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	PORTD
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

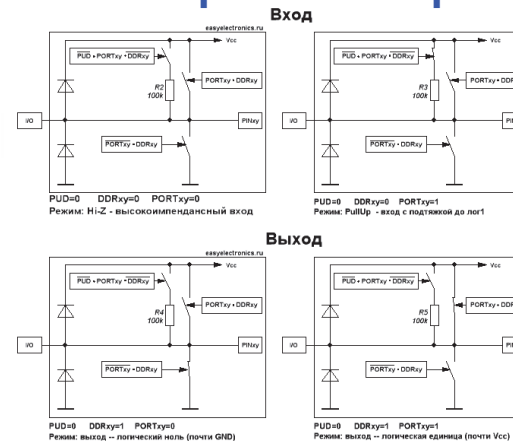
## DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	DDRD
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

## PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	PIND
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

# Режимы работы порта GPIO



DDR<sub>x</sub>=0 — вывод работает как ВХОД.

DDR<sub>x</sub>=1 вывод работает на ВЫХОД

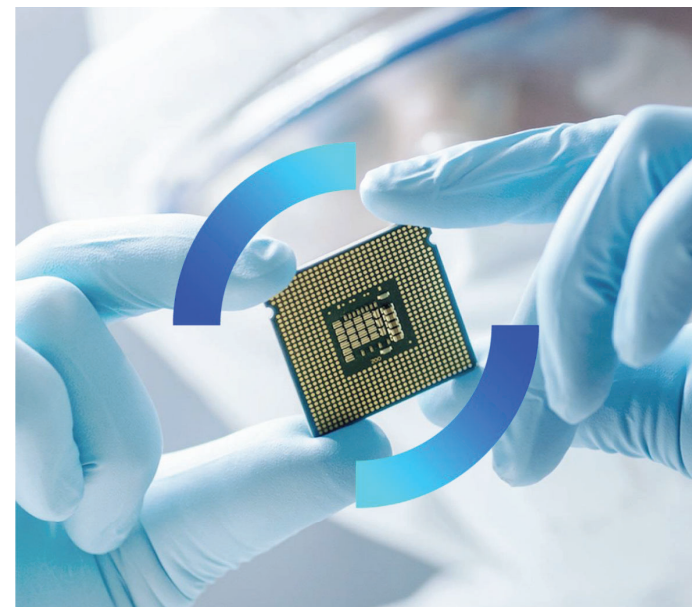


**Спасибо  
за внимание!**

Ленинский проспект, д. 4  
Москва, 119049  
тел. +7 (495) 955-00-32  
e-mail: [kancela@misys.ru](mailto:kancela@misys.ru)  
[misys.ru](http://misys.ru)



**Программирование  
микроконтроллеров**  
Битовые операции



## 8 битовый регистр микроконтроллера Atmega328

Умение работать с битами в микроконтроллере позволит:

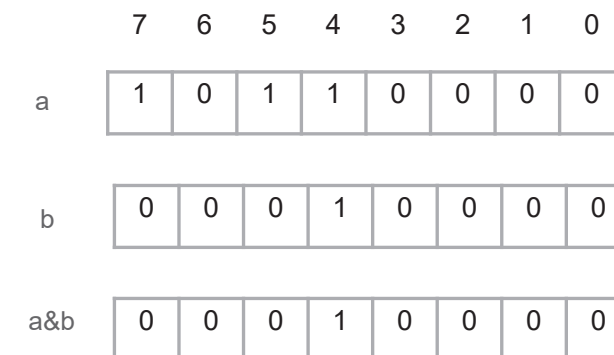
- 1) Работать напрямую с регистрами микроконтроллера
- 2) Работать с внешними микросхемами и осуществлять быстрый парсинг сигнала из двоичной системы
- 3) Эффективно организовывать хранение данных: упаковывать данные с цифровых данных в одну переменную и распаковывать обратно
- 4) Создавать символы и другую информацию для матричных дисплеев
- 5) Позволит понимать чужой код
- 6) Осуществлять быстрые вычисления



## Логическое умножение (битовое И)

Оператор & или and

x1	x0	x1&x0
0	0	0
0	1	0
1	0	0
1	1	1



## Логическое сложение (битовое ИЛИ)

Оператор | или or

x1	x0	x1&x0
0	0	0
0	1	1
1	0	1
1	1	1

	7	6	5	4	3	2	1	0
a	1	0	1	1	0	0	0	0
b	0	0	0	1	0	0	0	0
a   b	1	0	1	1	0	0	0	0

## Отрицание (инверсия) (битовое НЕ)

Оператор ~

x0	~ x0
0	1
1	0

	7	6	5	4	3	2	1	0
a	1	0	1	1	0	0	0	0
~ a	0	1	0	0	1	1	1	1



## Исключающее или (битовое ИЛИ)

Оператор ^ или xor

x1	x0	x1^X0
0	0	0
0	1	1
1	0	1
1	1	0

	7	6	5	4	3	2	1	0
a	1	0	1	1	0	0	0	0
b	0	0	0	1	0	0	0	0
a ^ b	1	0	1	0	0	0	0	0

## Умножение и деление 2<sup>n</sup> (битовый сдвиг)

Оператор << или >>

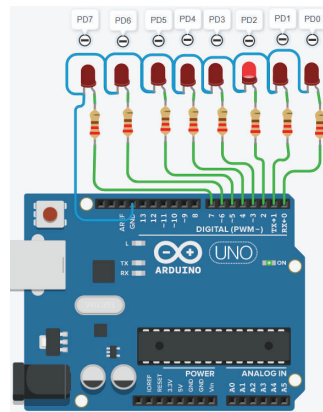
Двоичная	Десятичная
0b000011	3
0b000011 << 2 == 1100	3*4 = 12

	7	6	5	4	3	2	1	0
a	0	0	0	0	0	0	1	1
a<<2	0	0	0	0	1	1	0	0

## Установка лог. 1 в требуемый пин порта

`PORTD |= (1 << 2);`

```
1. void setup() {
2.   DDRD = 0b11111111; // установка порта D в режим вывода
3.   PORTD = 0b00000000; // устанавливаем "0" в пины порта D
4.   PORTD |= (1 << 2); // устанавливаем "1" на 2 пине порта D
5. }
6. void loop()
7. {
8. }
```

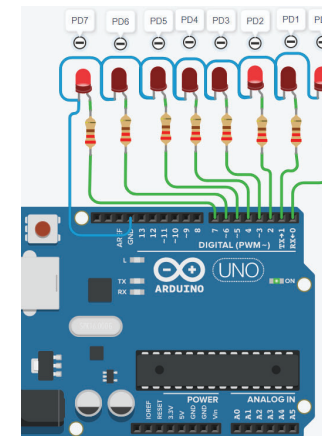


8

## Установка лог. 1 в несколько пинов порта

`PORTD |= (1 << 0) | (1 << 2) | (1 << 7);`

```
1. void setup() {
2.   DDRD = 0b11111111; // установка порта D в режим вывода
3.   PORTD = 0b00000000; // устанавливаем "0" в пины порта D
4.   PORTD |= (1 << 0) | (1 << 2) | (1 << 7); // устанавливаем "1" на
   0, 2 и 7 пине порта D
5. }
6. void loop()
7. {
8. }
```

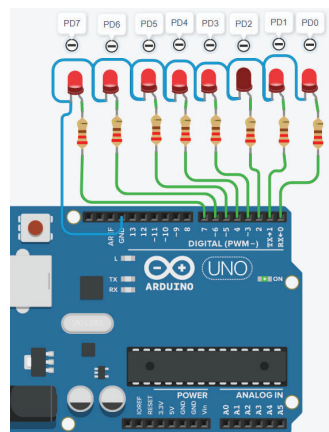


9

## Установка лог. 0 в требуемый пин порта

1. PORTD = ~(1 << 2);

```
1. void setup() {
2.   DDRD = 0b11111111; // установка порта D в режим вывода
3.   PORTD = 0b11111111; // устанавливаем "1" в пины порта D
4.   PORTD = ~(1 << 2); // устанавливаем "0" на 2 пине порта D
5. }
6. void loop()
7. {
8. }
```

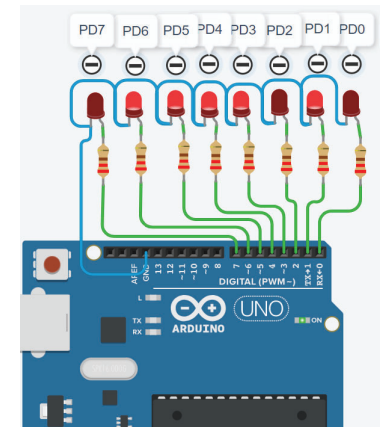


10

## Установка лог. 0 в несколько пинов порта

PORTD &= ~( (1 << 0) | (1 << 5) );

```
1. void setup() {
2.   DDRD = 0b11111111; // установка порта D в режим вывода
3.   PORTD = 0b11111111; // устанавливаем "1" в пины порта D
4.   PORTD &= ~( (1 << 0) | (1 << 5) ); // устанавливаем "0" на 2 пине
   порта D
5. }
6. void loop()
7. {
8. }
```



11

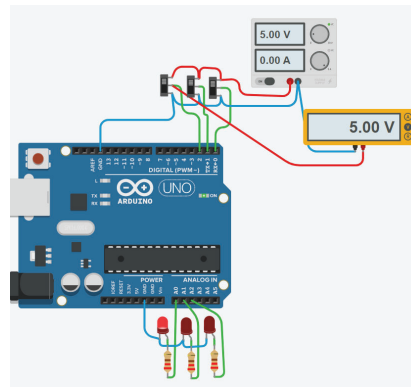
## Проверка лог. 1 в регистре

`PORTD & (1 << 2);`

```

1. void setup() {
2.   DDRD = 0b00000000; // установка 0-7 пин порта D в режим
   ввода
3.   DDRC = 0b111111; // установка 0-5 порта C в режим вывода
4. }
5. void loop() {
6.   if (PIND & (1 << 2)) {
7.     PORTC |= (1 << 0);
8.   } else {
9.     PORTC &= !(1 << 0);
10.  }
11. }

```



12

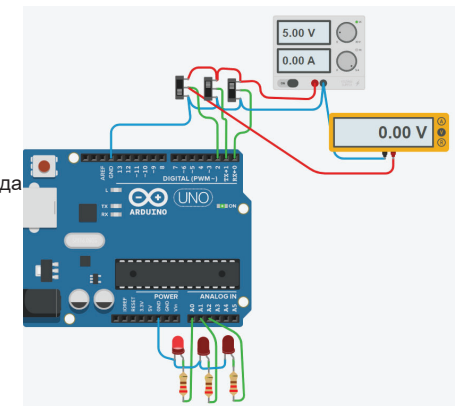
## Проверка лог. 0 в регистре

`PORTD & (1 << 2);`

```

1. void setup() {
2.   DDRD = 0b00000000; // установка 0-7 пин порта D в режим
   ввода
3.   DDRC = 0b111111; // установка 0-5 порта C в режим вывода
4. }
5. void loop() {
6.   if (!(PIND & (1 << 2))) {
7.     PORTC |= (1 << 0);
8.   } else {
9.     PORTC &= !(1 << 0);
10.  }
11. }

```



13

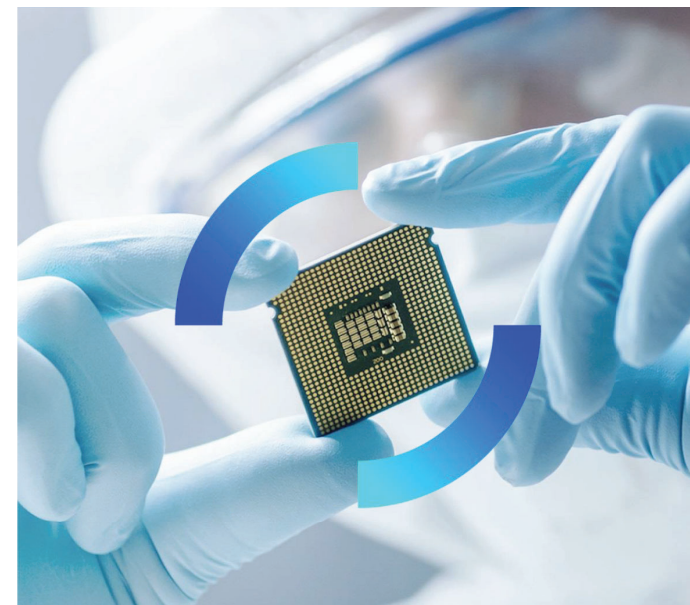


**Спасибо  
за внимание!**

Ленинский проспект, д. 4  
Москва, 119049  
тел. +7 (495) 955-00-32  
e-mail: [kancela@misys.ru](mailto:kancela@misys.ru)  
[misys.ru](http://misys.ru)



**Программирование  
микроконтроллеров**  
Таймер-работа со временем



## Таймер на millis()

millis() - программа возвращающая количество миллисекунд от начала работы программы

10:00



Запомнил время!

10:02 — 10:00



В дороге 2 минуты

10:05 — 10:00



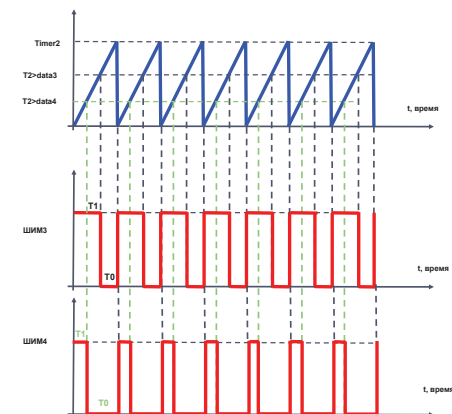
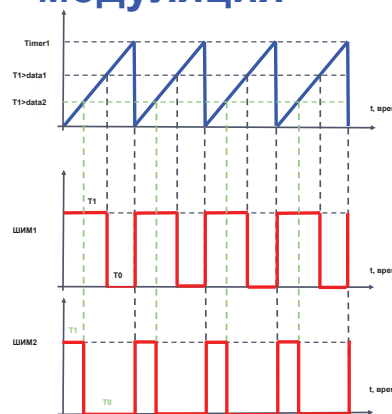
В дороге 5 минут

А

Б

2

## Широтно-импульсная модуляция



3

## Широтно-импульсная модуляция

`analogWrite`(порт, величина сигнала)

порт - номер порта с ШИМ управлением

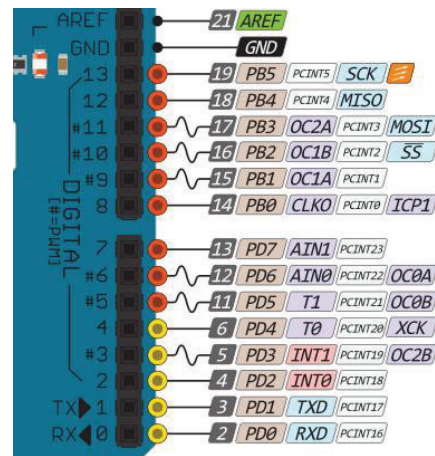
Atmega 328 имеет 6 выводов с возможностью ШИМ управлением, это 3, 5, 6, 9, 10 и 11 выводы Arduino или

5, 11, 12, 15, 16, 17 выводы микроконтроллера Atmega328

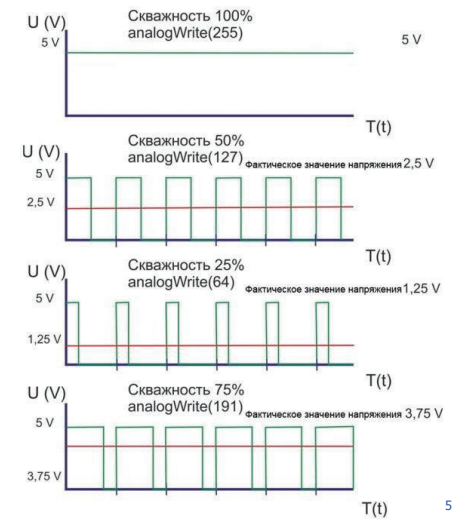
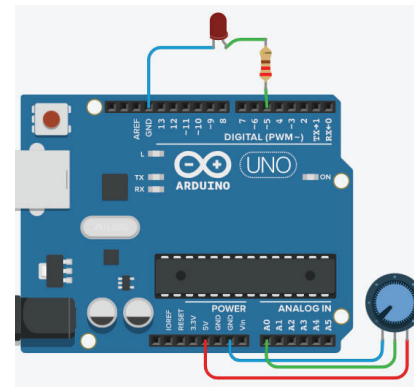
величина сигнала

ШИМ 8 битный поэтому величина меняется от 0 до 255

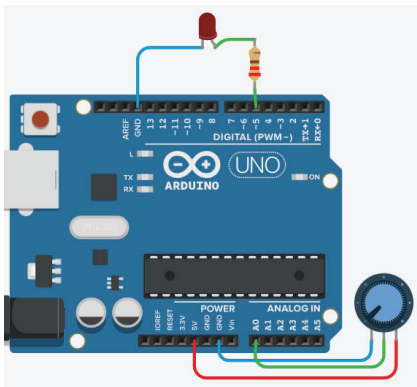
скважность — отношение периода импульсного сигнала к длительности импульса



## Широтно-импульсная модуляция



## Широтно-импульсная модуляция

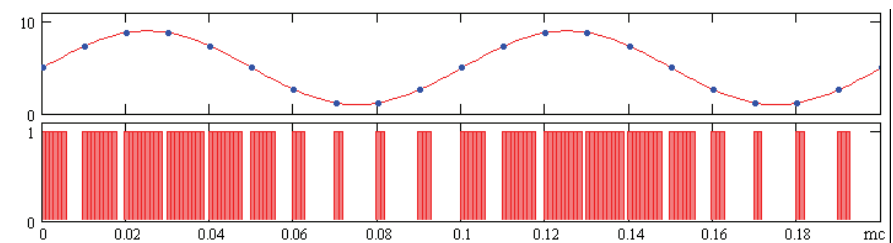
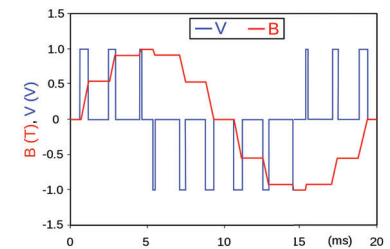


```

1. int data =0;
2. void setup()
3. {
4.   pinMode(5, OUTPUT);
5. }
6. void loop()
7. {
8.   data = analogRead(A0);
9.   data = map(data, 0, 1023, 0, 255);
10.  analogWrite(5, data);
11.}

```

## Широтно-импульсная модуляция

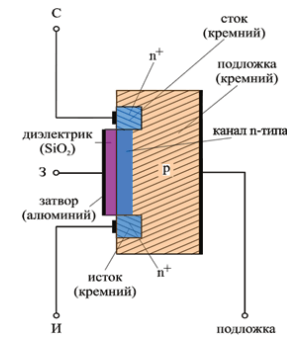
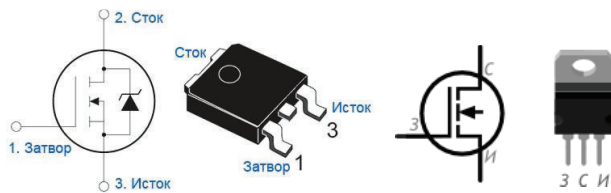




## Управление мощной нагрузкой

MOSFET (*metal-oxide-semiconductor field-effect transistor*) – полевой транзистор с изолированным затвором

особенностью полевого транзистора заключается в возможности управления протекающим через него током с помощью электрического поля (напряжения).

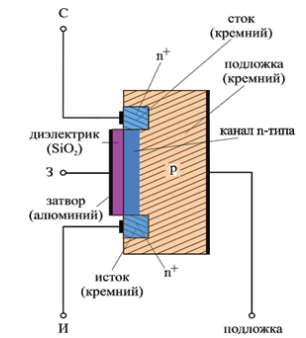
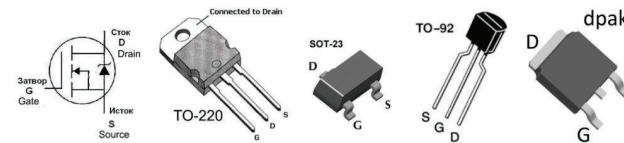


транзистор с изолированным затвором p - типа

## Управление мощной нагрузкой

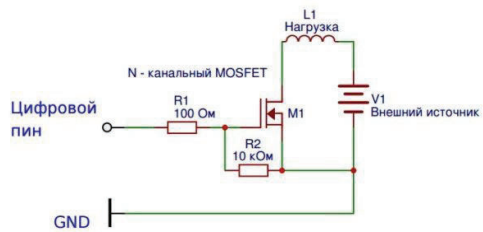
MOSFET (*metal-oxide-semiconductor field-effect transistor*) – полевой транзистор с изолированным затвором

особенностью полевого транзистора заключается в возможности управления протекающим через него током с помощью электрического поля (напряжения).



транзистор с изолированным затвором p - типа

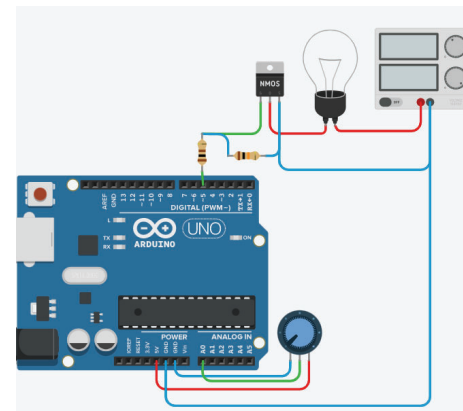
## Управление мощной нагрузкой



```

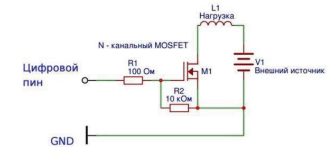
1. int data =0;
2. void setup()
3. {
4.   pinMode(5, OUTPUT);
5. }
6. void loop()
7. {
8.   data = analogRead(A0);
9.   data = map(data, 0, 1023, 0, 255);
10.  analogWrite(5, data);
11.}
    
```

## Управление мощной нагрузкой

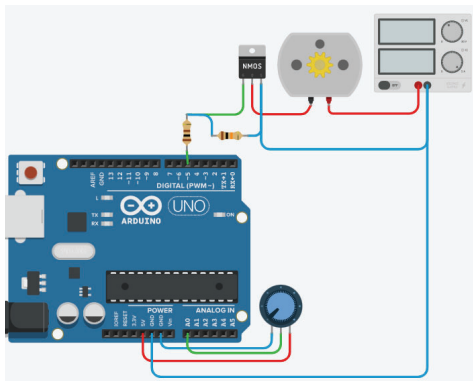


```

1. int data =0;
2. void setup()
3. {
4.   pinMode(5, OUTPUT);
5. }
6. void loop()
7. {
8.   data = analogRead(A0);
9.   data = map(data, 0, 1023, 0, 255);
10.  analogWrite(5, data);
11.}
    
```

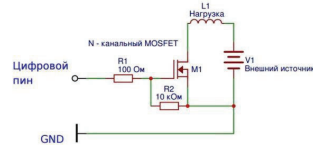


## Управление мощной нагрузкой

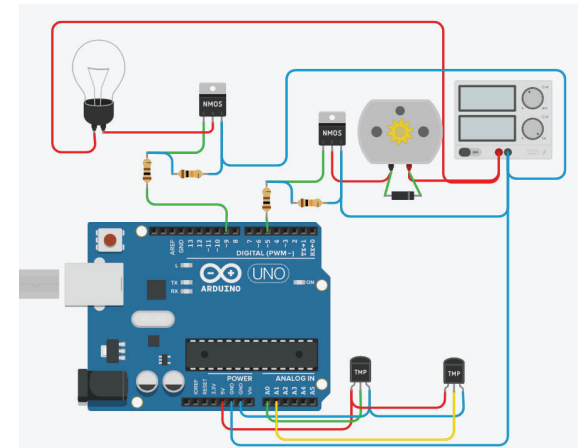
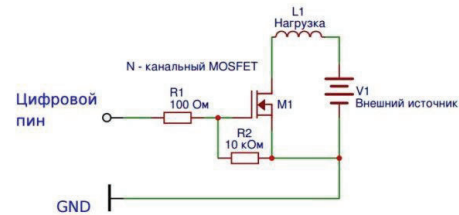


```

1. int data =0;
2. void setup()
3. {
4.   pinMode(5, OUTPUT);
5. }
6. void loop()
7. {
8.   data = analogRead(A0);
9.   data = map(data, 0, 1023, 0, 255);
10.  analogWrite(5, data);
11.}
    
```



## Проект инкубатор, теплица, климат



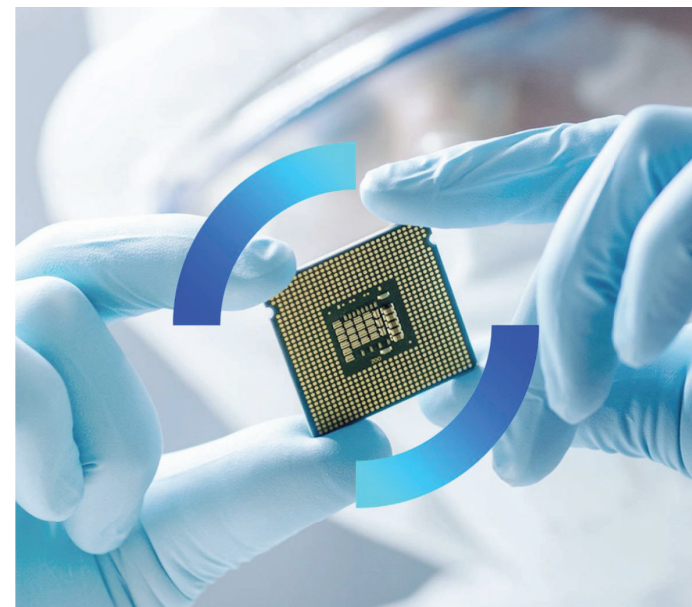


**Спасибо  
за внимание!**

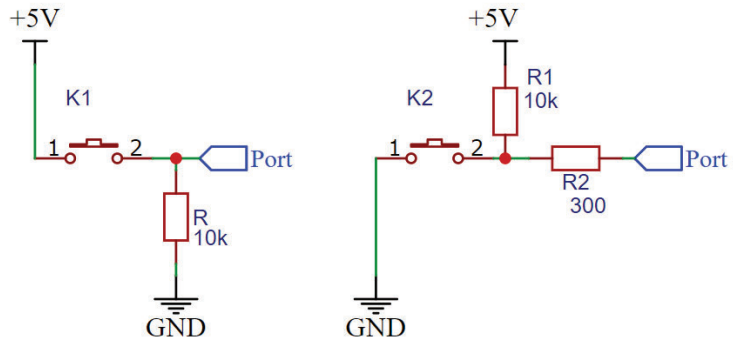
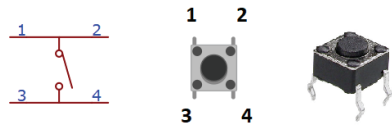
Ленинский проспект, д. 4  
Москва, 119049  
тел. +7 (495) 955-00-32  
e-mail: [kancela@misys.ru](mailto:kancela@misys.ru)  
[misys.ru](http://misys.ru)



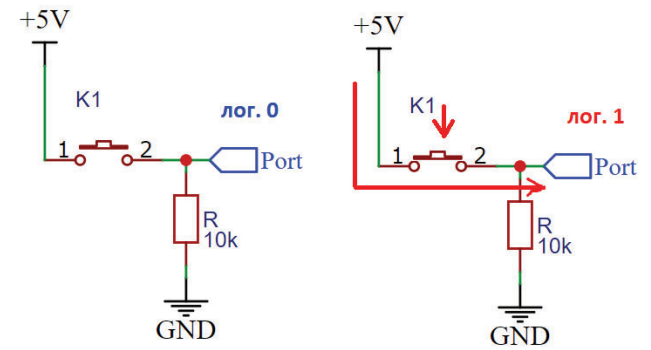
**Программирование  
микроконтроллеров**  
Ввод информации



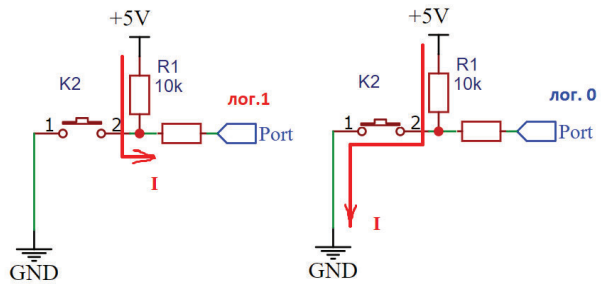
## Подключение кнопки



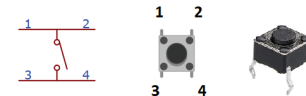
## Подключение кнопки



## Подключение кнопки



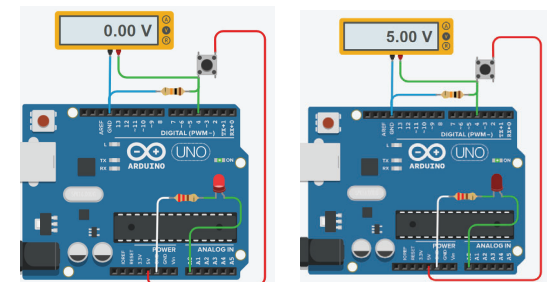
## Подключение кнопки



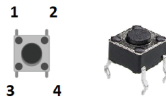
```

1. void setup() {
2.   DDRD = 0b00000000; // установка 0-7 пин порта D в режим ввода
3.   DDRC = 0b1111111; // установка 0-5 порта C в режим вывода
4. }
5. void loop() {
6.   if (!(PIND & (1 << 4))) {
7.     PORTC |= (1 << 0);
8.   } else {
9.     PORTC &= ~(1 << 0);
10.  }
11. }

```



## Подключение кнопки

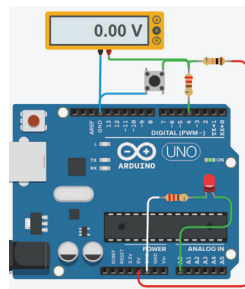
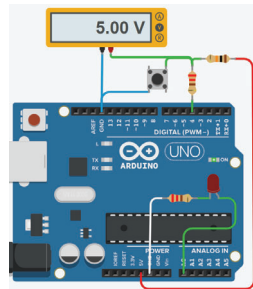


misis.ru

```

1. void setup(){
2.   DDRD = 0b00000000; // установка 0-7 пин порта D в режим ввода
3.   DDRC = 0b1111111; // установка 0-5 порта C в режим вывода
4. }
5. void loop() {
6.   if (!(PIND & (1 << 4))){
7.     PORTC |= (1 << 0);
8.   } else {
9.     PORTC &= ~(1 << 0);
10.  }
11.}

```



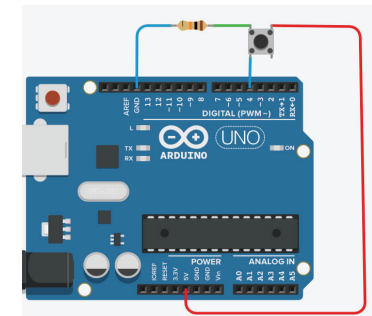
6

## Обработка нажатия

```

1. #define button 4
2. void setup()
3. {
4.   Serial.begin(9600);
5.   pinMode(button, INPUT);
6. }
7. void loop()
8. {
9.   if (digitalRead(button))
10.  {
11.    Serial.println("Push button");
12.  }
13.}

```



misis.ru

Монитор последовательного интерфейса

```

Push button
Push button
Push button
Push button
Push button
Push button
Push button
Push button
Push button
Push button

```

**один клик на кнопку  
приводит к многократному  
считыванию сигнала!**

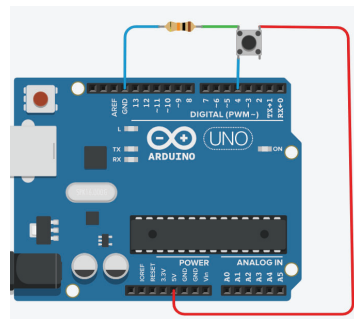
7

## Обработка однократного нажатия

```

1. void setup() {
2.   Serial.begin(9600);
3.   pinMode(4, INPUT);
4. }
5. bool flag = false;
6. void loop() {
7.   bool btnState = digitalRead(4);
8.   if (btnState && !flag) { // обработчик нажатия
9.     flag = true;
10.    Serial.println("PUSH DOWN");
11.  }
12.  if (!btnState && flag) { // обработчик отпускания
13.    flag = false;
14.    //Serial.println("PUSH UP");
15.  }
16.}

```

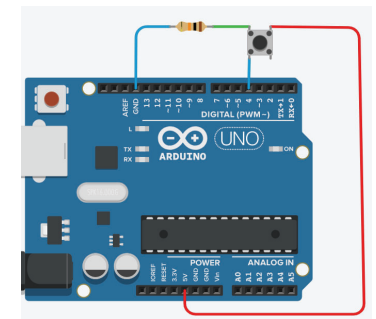


## Обработка однократного нажатия

```

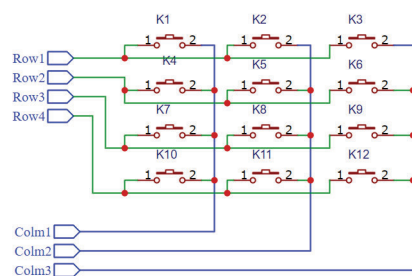
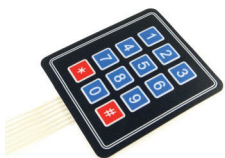
1. void setup() {
2.   Serial.begin(9600);
3.   PORTD &= !(1 << 4); // устанавливаем 4 пин D на выход
4. }
5. byte flag = 0; //обнуляем флаги
6. byte state = 0; //обнуляем флаги текущего состояния кнопок
7. void loop() {
8.   state = ((PIND & (1 << 4)) >> 4); // записали состояние в 0 бит state
9.   if (((state & (1 << 0)) & !(flag & (1 << 0)))) // обработчик нажатия
10.  {
11.    flag |= (1 << 0); // устанавливаем flag 1
12.    Serial.println("PUSH DOWN");
13.  }
14.  if (!(((state & (1 << 0)) & (flag & (1 << 0))))) { // обработчик отпускания
15.    flag &= !(1 << 0); // устанавливаем 0 бит flag лог 0
16.    // Serial.println("PUSH UP");
17.  }
18.}

```





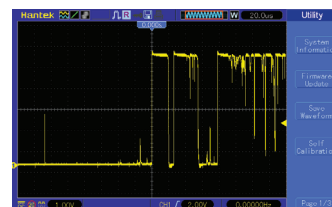
## Подключение клавиатуры



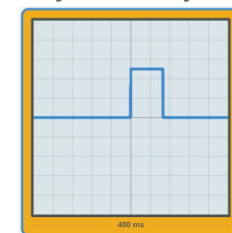
## Дребезг контактов

Это явление в электромеханических коммутационных устройствах наблюдается в момент замыкания контактов, происходят многократные неконтролируемые замыкания и размыкания контактов за счёт упругости материалов и деталей контактной системы — некоторое время контакты отскакивают друг от друга при соударениях, размыкая и замыкая электрическую цепь.

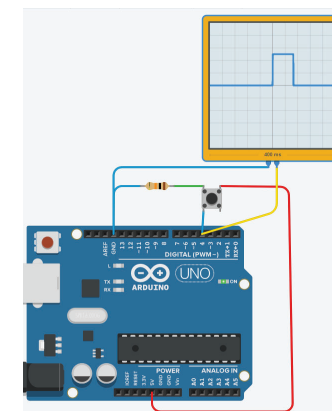
### реальная коммутация



### коммутация в эмуляторе

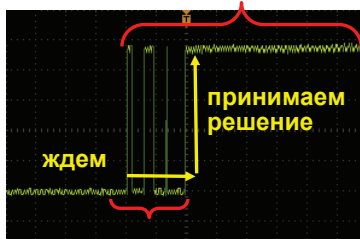


идеальная



## Дребезг контактов

### коммутация



### дребезг

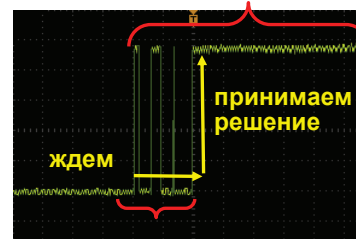
```

1. void setup() {
2.   Serial.begin(9600);
3.   pinMode(3, INPUT_PULLUP);
4. }
5. bool flag = false;
6. uint32_t butt_Timer = 0;
7. void loop() {
8.   // читаем инвертированное значение для удобства
9.   bool butt_State = !digitalRead(3);
10.  if (butt_State && !flag && millis() - butt_Timer > 100) {
11.    flag = true;
12.    butt_Timer = millis();
13.    Serial.println("PUSH DOWN");
14.  }
15.  if (!butt_State && flag && millis() - butt_Timer > 100) {
16.    flag = false;
17.    butt_Timer = millis();
18.    //Serial.println("PUSH UP");
19.  }
20.}

```

## Дребезг контактов

### коммутация



### дребезг

```

1. uint32_t butt_Timer = 0;
2. void setup() {
3.   Serial.begin(9600);
4.   PORTD &= !(1 << 4); // устанавливаем 4 пин D на выход
5. }
6. byte flag = 0; //обнуляем флаги
7. byte state = 0; //обнуляем состояния кнопок
8. void loop() {
9.   // читаем значение пина
10.  state = ((PIND & (1 << 4)) >> 4); // записали состояние в 0 бит state
11.  // обработчик нажатия
12.  if (((state & (1 << 0)) & !(flag & (1 << 0))) && (millis() - butt_Timer > 100)) {
13.    flag |= (1 << 0); // устанавливаем flag 1
14.    butt_Timer = millis();
15.    Serial.println("PUSH DOWN");
16.  }
17.  // обработчик отпущения
18.  if (!(state & (1 << 0)) & (flag & (1 << 0))) && (millis() - butt_Timer > 100) {
19.    flag &= !(1 << 0); // устанавливаем 0 бит flag лог 0
20.    butt_Timer = millis();
21.    // Serial.println("PUSH UP");
22.  }
23.}

```

## Дребезг контактов

дребезг



коммутация

```
1. uint32_t butt_Timer = 0;
2. void setup() {
3.   Serial.begin(9600);
4.   PORTD &= !(1 << 4); // устанавливаем 4 пин D на выход
5. }
6. byte flag = 0; //обнуляем флаги
7. byte state = 0; //обнуляем состояния кнопок
8. void loop() {
9.   // читаем значение пина
10.  state = ((PIND & (1 << 4)) >> 4); // записали состояние в 0 бит state
11. // обработчик нажатия
12.  if (((state & (1 << 0)) & !(flag & (1 << 0)))&& (millis() - butt_Timer > 100)) {
13.    flag |= (1 << 0); // устанавливаем flag 1
14.    butt_Timer = millis();
15.    Serial.println("PUSH DOWN");
16.  }
17. // обработчик отпущения
18.  if (!(state & (1 << 0)) & (flag & (1 << 0)))&& (millis() - butt_Timer > 100)) {
19.    flag &= !(1 << 0); // устанавливаем 0 бит flag лог 0
20.    butt_Timer = millis();
21.    // Serial.println("PUSH UP");
22.  }
23. }
```

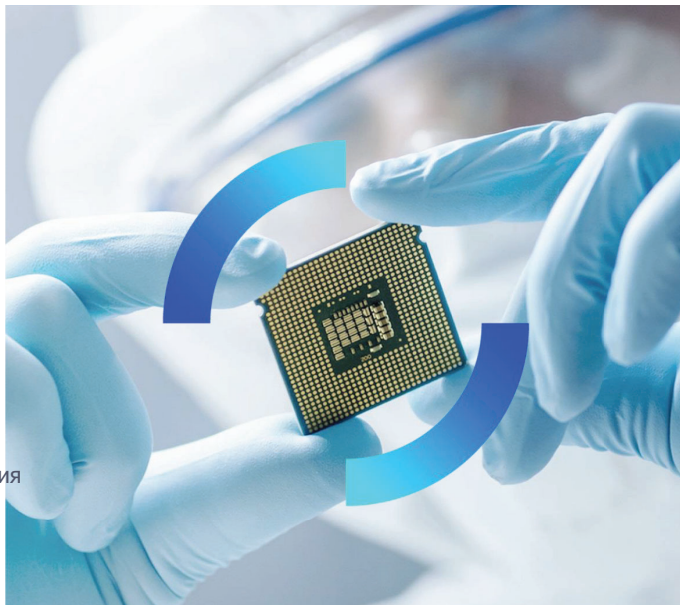
misis.ru

14

## Спасибо за внимание!

Ленинский проспект, д. 4  
Москва, 119049  
тел. +7 (495) 955-00-32  
e-mail: kancela@misis.ru  
misis.ru





## Широтно-импульсная модуляция

**analogWrite(порт, величина сигнала)**

**порт - номер порта с ШИМ управлением**

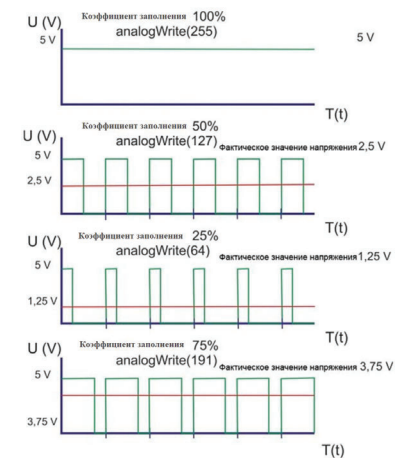
Atmega 328 имеет 6 вывод с возможностью ШИМ управлением, это 3, 5, 6, 9, 10 и 11 выводы Arduino или 5, 11, 12, 15, 16, 17 выводы микроконтроллера Atmega328

**величина сигнала**

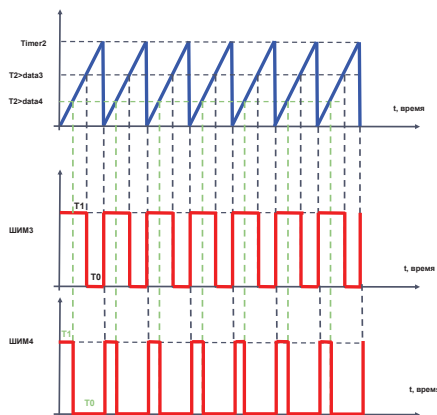
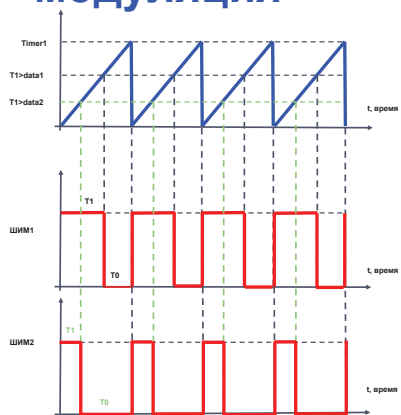
**ШИМ 8 битный** поэтому величина меняется от 0 до 255

**коэффициент заполнения (D)**– отношение длительности импульса к периоду импульсного сигнала

$$D = T1/T$$

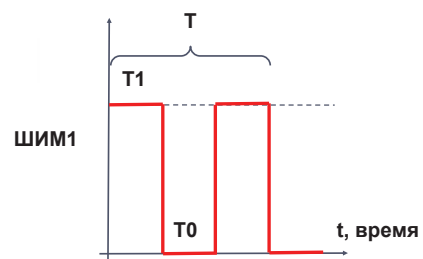


## Широтно-импульсная модуляция



3

## Широтно-импульсная модуляция



скважность – отношение периода импульсного сигнала к длительности импульса  $T_1/(T)$

$$f = 62.5 \text{ кГц}$$

$$T = (1/62.5 \cdot 1000) = 0,016 \cdot 10^{-3} = 16 \text{ мкс}$$

$$T_1 = 8 \text{ мкс}$$

$$T_0 = 8 \text{ мкс}$$

$$\text{скважность}[\%] = (8/16) \cdot 100\% = 50\%$$

## Широтно-импульсная модуляция

`analogWrite`(порт, величина сигнала)

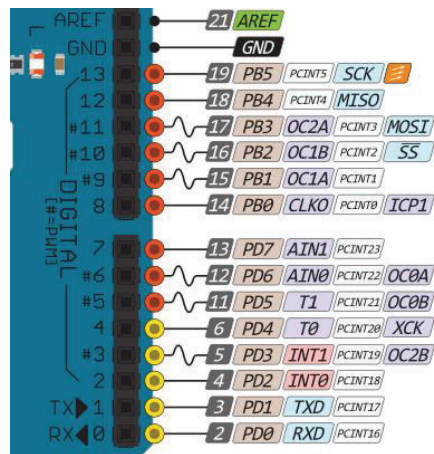
порт - номер порта с ШИМ управлением

Atmega 328 имеет 6 выводов с возможностью ШИМ управлением, это 3, 5, 6, 9, 10 и 11 выводы Arduino или 5, 11, 12, 15, 16, 17 выводы микроконтроллера Atmega328

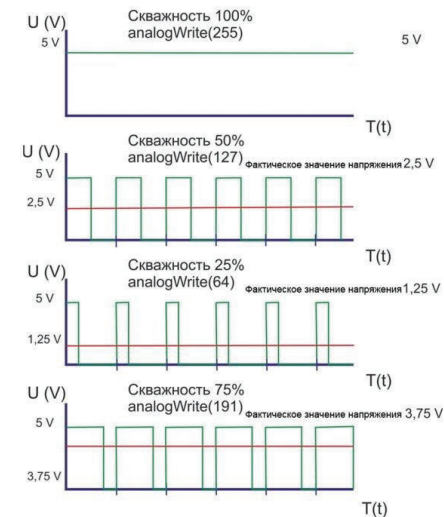
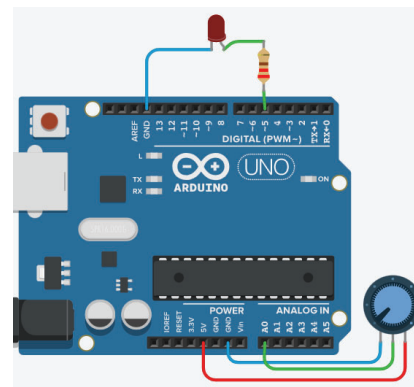
величина сигнала

ШИМ 8 битный поэтому величина меняется от 0 до 255

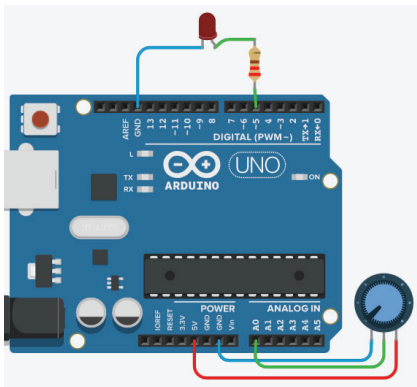
скважность — отношение периода импульсного сигнала к длительности импульса



## Широтно-импульсная модуляция



## Широтно-импульсная модуляция

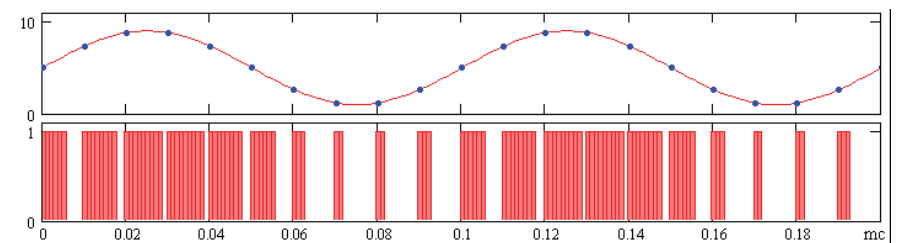
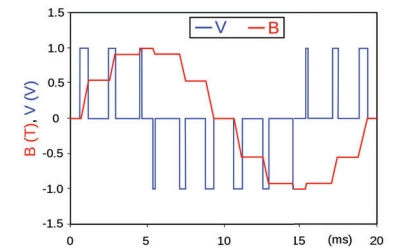


```

1. int data =0;
2. void setup()
3. {
4.   pinMode(5, OUTPUT);
5. }
6. void loop()
7. {
8.   data = analogRead(A0);
9.   data = map(data, 0, 1023, 0, 255);
10.  analogWrite(5, data);
11. }

```

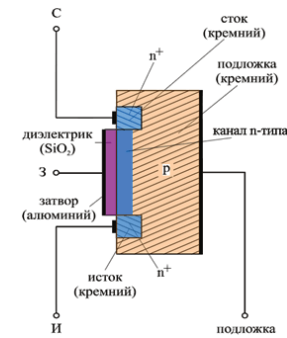
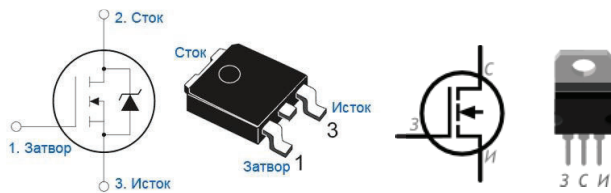
## Широтно-импульсная модуляция



## Управление мощной нагрузкой

MOSFET (*metal-oxide-semiconductor field-effect transistor*) – полевой транзистор с изолированным затвором

особенностью полевого транзистора заключается в возможности управления протекающим через него током с помощью электрического поля (напряжения).

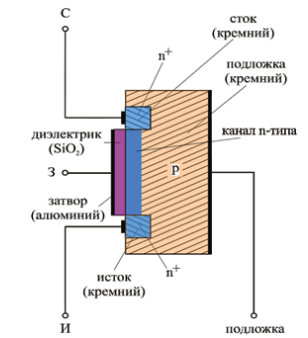
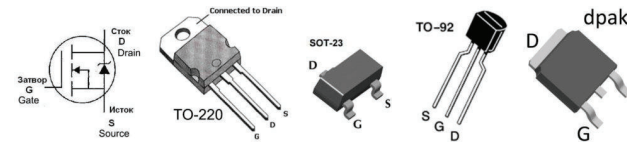


транзистор с изолированным затвором p - типа

## Управление мощной нагрузкой

MOSFET (*metal-oxide-semiconductor field-effect transistor*) – полевой транзистор с изолированным затвором

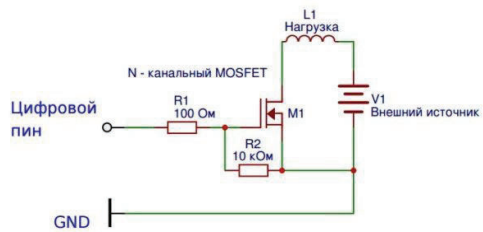
особенностью полевого транзистора заключается в возможности управления протекающим через него током с помощью электрического поля (напряжения).



транзистор с изолированным затвором p - типа



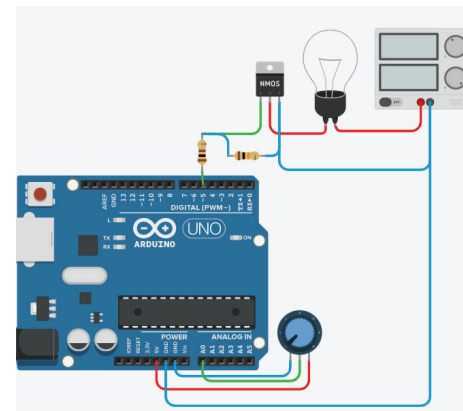
## Управление мощной нагрузкой



```

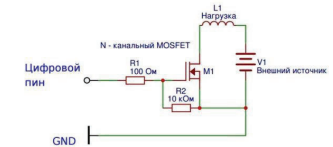
1. int data =0;
2. void setup()
3. {
4.   pinMode(5, OUTPUT);
5. }
6. void loop()
7. {
8.   data = analogRead(A0);
9.   data = map(data, 0, 1023, 0, 255);
10.  analogWrite(5, data);
11.}
    
```

## Управление мощной нагрузкой

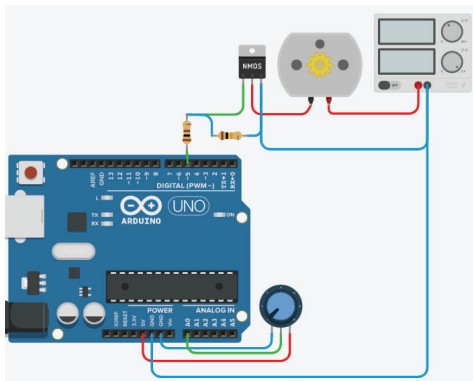


```

1. int data =0;
2. void setup()
3. {
4.   pinMode(5, OUTPUT);
5. }
6. void loop()
7. {
8.   data = analogRead(A0);
9.   data = map(data, 0, 1023, 0, 255);
10.  analogWrite(5, data);
11.}
    
```

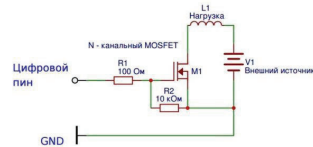


## Управление мощной нагрузкой

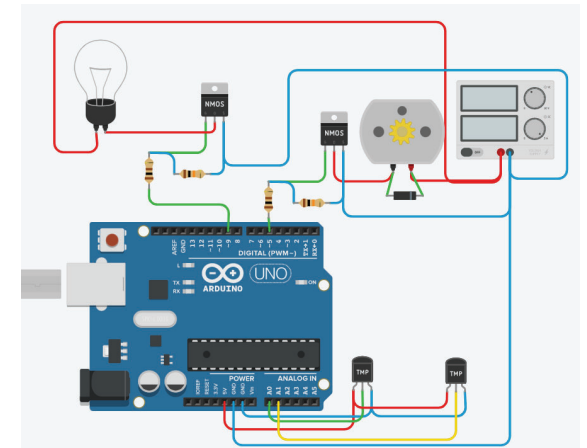
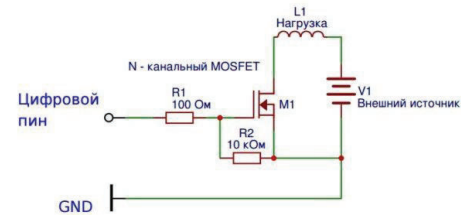


```

1. int data =0;
2. void setup()
3. {
4.   pinMode(5, OUTPUT);
5. }
6. void loop()
7. {
8.   data = analogRead(A0);
9.   data = map(data, 0, 1023, 0, 255);
10.  analogWrite(5, data);
11.}
    
```



## Проект инкубатор, теплица, климат



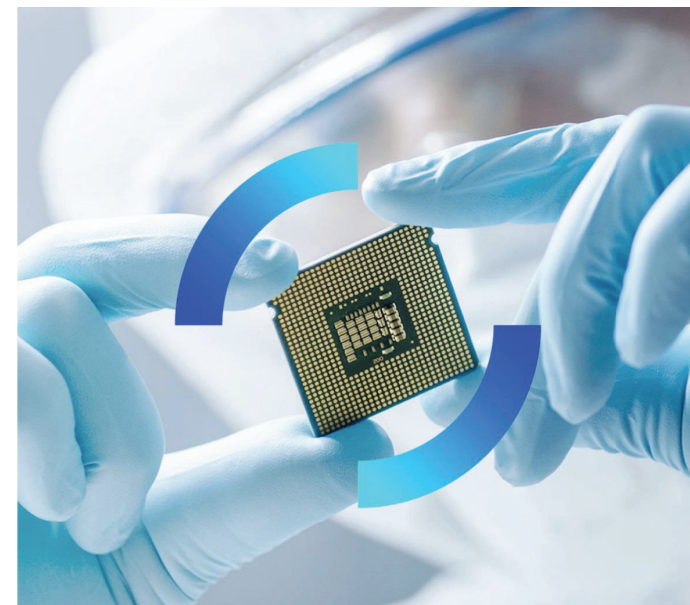


**Спасибо  
за внимание!**

Ленинский проспект, д. 4  
Москва, 119049  
тел. +7 (495) 955-00-32  
e-mail: [kancela@misys.ru](mailto:kancela@misys.ru)  
[misys.ru](http://misys.ru)

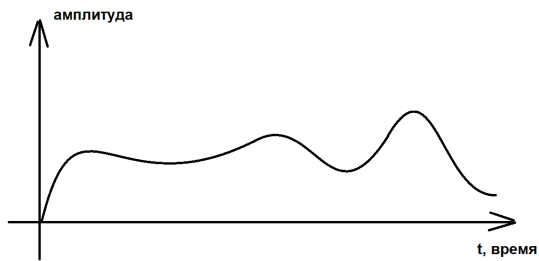


**Программирование  
микроконтроллеров**  
Аналоговые порты



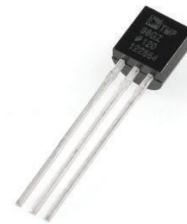
## Аналоговый сигнал

**Аналоговый сигнал** - это сигнал, порождаемый физическим процессом, параметры которого можно измерить в любой момент времени



## Аналоговый датчики

**Аналоговый датчик** - это электротехническое устройство преобразующее измеряемую физическую величину в пропорциональный электрический сигнал постоянный по времени



**температурный датчик**  
TMP35, TMP37, LM35, LM335

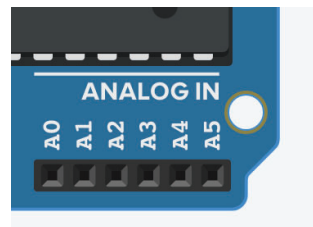
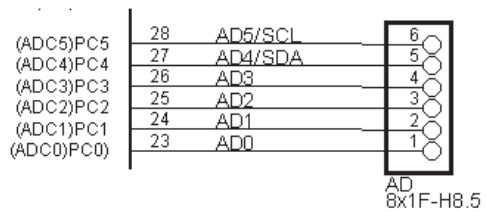


**датчик влажности**



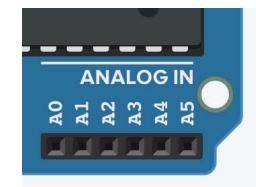
**потенциометр**

## Аналоговые порты (АЦП)



## Функция analogRead()

```
analogRead(pin)
```



6 каналов с 10-битным аналого-цифровым преобразователем (АЦП)

Напряжение поданное на аналоговый вход преобразуется в значение от 0 до 1023, это 1024 шага с разрешением 0.0049 Вольт. Разброс напряжение и шаг может быть изменен функцией analogReference().

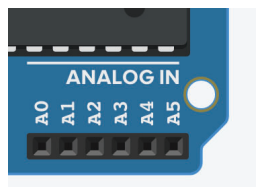
## Функция analogReference()

analogReference(type)

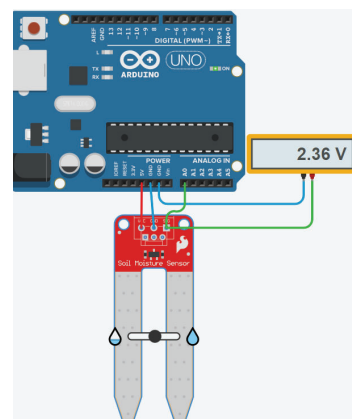
DEFAULT: стандартное опорное напряжение 5 В

INTERNAL: встроенное опорное напряжение 1.1 В на микроконтроллерах ATmega328

EXTERNAL: внешний источник опорного напряжения, подключенный к выводу AREF



## Подключение датчиков



```

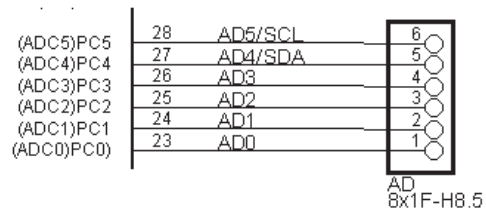
1. int u;
2. void setup ()
3. {
4.   Serial.begin(9600);
5. }
6. void loop ()
7. {
8.   u = analogRead(A0);
9.   Serial.println(u);
10.}

```

## Аналогово-цифровой преобразователь

АЦП, англ. Analog-to-digital converter, ADC)

Atmega328 имеет 10 разрядный АЦП, который работает на порту С



4 вида опорных напряжений

В регистры ADCH и ADCL записываются показания и рассчитываются по формуле:

$$ADC = \frac{V_{in} \cdot 1024}{V_{ref}}$$

## Настройка АЦП

1. Настройка регистра **ADMUX** (Регистр настройки мультиплексора АЦП).
2. Настройка регистра **ADCSRA** (Регистр статуса и контроля А)
3. Настройка **ADCSRB** (Регистр статуса и контроля В)
3. Чтение результата преобразования

## Регистр *ADMUX*

Регистр настройки мультимплектора АЦП

Bit	7	6	5	4	3	2	1	0	
(0x7C)	<b>REFS1</b>	<b>REFS0</b>	<b>ADLAR</b>	<b>-</b>	<b>MUX3</b>	<b>MUX2</b>	<b>MUX1</b>	<b>MUX0</b>	<b>ADMUX</b>
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**Биты 7-6 REFS1 и REFS0.** Данная группа битов определяет какой тип источник опорного напряжения (ИОН) будет подключен к микроконтроллеру. При инициализации начальное состояние определяется как 0.

REFS1	REFS0	Тип ИОН подключенного ко входу опорного напряжения
0	0	Подключен внешний опорный ион, внутренний ион выключен
0	1	Подключено напряжение питания AVcc
1	0	Состояние зарезервировано
1	1	Подключен внутренний ИОН 1.1В, с внешним конденсатором на AREF пине

## Регистр *ADMUX*

Регистр настройки мультимплектора АЦП

Bit	7	6	5	4	3	2	1	0	
(0x7C)	<b>REFS1</b>	<b>REFS0</b>	<b>ADLAR</b>	<b>-</b>	<b>MUX3</b>	<b>MUX2</b>	<b>MUX1</b>	<b>MUX0</b>	<b>ADMUX</b>
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**Бит 5 ADLAR.** Данный бит выбирает тип представления результата преобразования. При записи в этот бит логической 1 результат в регистрах ADCL и ADCH будет представлен в левостороннем представлении, иначе правосторонний. При инициализации по умолчанию устанавливает 0 значение.



## Регистр *ADCL ADCH*

**ADLAR = 0**

Bit	15	14	13	12	11	10	9	8	
(0x79)	-	-	-	-	-	-	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

**ADLAR = 1**

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	-	-	-	-	-	-	ADCL
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

## Регистр *ADMUX*

Регистр настройки мультимплексора АЦП

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**Биты 3-0 MUX3, MUX2, MUX1 и MUX0.** Данная группа битов определяет какой вход будет активен в качестве входа для АЦП. Для определения номера пина входного канала, необходимо данную группу сконфигурировать согласно таблице представленной ниже.

MUX3...0	Номер пина
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8

## Регистр ADCSRA

Регистр управления и статуса A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**Бит 7 ADEN.** Данный бит предназначен для включения режима АЦП. Для включения АЦП необходимо в этот бит записать логическую 1, если необходимо отключить, то записываем 0. При инициализации, значение в этом бите равно 0.

**Бит 6 ADSC.** Данный бит включает режим преобразования входного аналогового сигнал в двоичный код. Для запуска преобразования, необходимо установить в этот бит логическую 1. После завершения преобразования, данный бит устанавливается в 0 значение.

## Регистр ADCSRA

Регистр управления и статуса A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**Бит 5 ADATE.** Данный бит включает автоматический запуск преобразования. Автоматическое преобразование запускается с помощью внешнего сигнала. Если преобразование завершено, а внешний сигнал присутствует, то новое преобразование не начинается. Так же, если преобразование не завершилось, а пришёл другой внешний сигнал о начале преобразования, то данный сигнал будет проигнорирован микроконтроллером. По завершению преобразования, будет выставлен флаг прерывания, даже если были отключены прерывания или бит глобального прерывания был очищен. Преобразование может быть сделано таким образом, чтобы не вызывать прерываний. Для того, чтобы вызвать новое преобразование, необходимо очистить бит флага прерываний.

## Регистр ADCSRA

Регистр управления и статуса A

Bit (0x7A)	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**Бит 4 ADIF.** Когда выполнено преобразование, выставляется флаг о завершении преобразования. Сбросить флаг преобразования можно путем записи в данный бит логической единицы. Так же данный бит сбрасывается, если включены прерывания от АЦП и вызывается вектор прерывания.

**Бит 3 ADIE.** При записи в данный бит логической 1 и если установлен 1-бит в регистре SREG (общий регистр статуса), включаются прерывания от АЦП по выполнению преобразования.

## Регистр ADCSRA

Регистр управления и статуса A

Bit 0x30 (0x50)	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

**Биты 2-0 ADPS2, ADPS1 и ADPS0.** С помощью данной группы битов устанавливается предделитель между системной частотой и входной частотой АЦП. Конфигурация данной группы битов и какой будет соответствовать предделитель указаны в таблице ниже

## Регистр ADCSRA

Регистр управления и статуса А

Bit	7	6	5	4	3	2	1	0	
0x30 (0x50)	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACISO	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	NA	0	0	0	0	0	

ADPS2	ADPS1	ADPS0	Пределитель
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

## Регистр ADCSRB

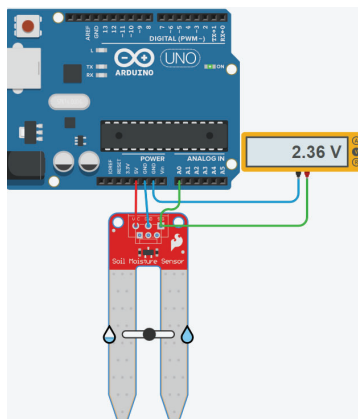
Регистр контроля и статуса В

Bit	7	6	5	4	3	2	1	0	
(0x7B)	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**Биты 2-0 ADTS2, ADTS1 и ADTS0.** Предназначены для определения от какого элемента необходимо выполнять запуск преобразования. Конфигурируя данную группу битов согласно таблице ниже, можно выбрать источник запуска преобразования.

ADTS2	ADTS1	ADTS0	Источник запуска
0	0	0	Свободный режим
0	0	1	Совпадение по аналоговому компаратору
0	1	0	Внешнее 0 прерывание
0	1	1	Совпадение с регистром сравнения А таймера/счётчика0
1	0	0	Переполнение таймера/счётчика0
1	0	1	Совпадение с регистром сравнения В таймера/счётчика1
1	1	0	Переполнение таймера/счётчика1
1	1	1	При изменении счётчика/таймера1

## Подключение датчиков



```

1. unsigned int u;
2. void setup()
3. {
4.   Serial.begin (9600);
5.   ADC_Init();
6. }

7. void loop()
8. {
9.   ADCSRA |= (1 << ADSC); // Начинаем преобразование
10.  while ((ADCSRA & (1 << ADIF)) == 0); // пока не будет выставлено
    флага об окончании преобразования
11.  u = (ADCL|ADCH << 8); // Считываем полученное значение
12.  Serial.println(u);
13.  //Проверяем считанное значение
14.}

15.void ADC_Init(){
16. ADCSRA |= (1 << ADEN) // Включаем АЦП
17. |(1 << ADPS1)|(1 << ADPS0); // устанавливаем делитель
    преобразователя на 8
18. ADMUX |= (0 << REFS1)|(1 << REFS0) //выставляем опорное
    напряжение питание AVcc
19. |(0 << MUX0)|(0 << MUX1)|(0 << MUX2)|(0 << MUX3); // снимать сигнал
    будем с входа PC0
20.}

```

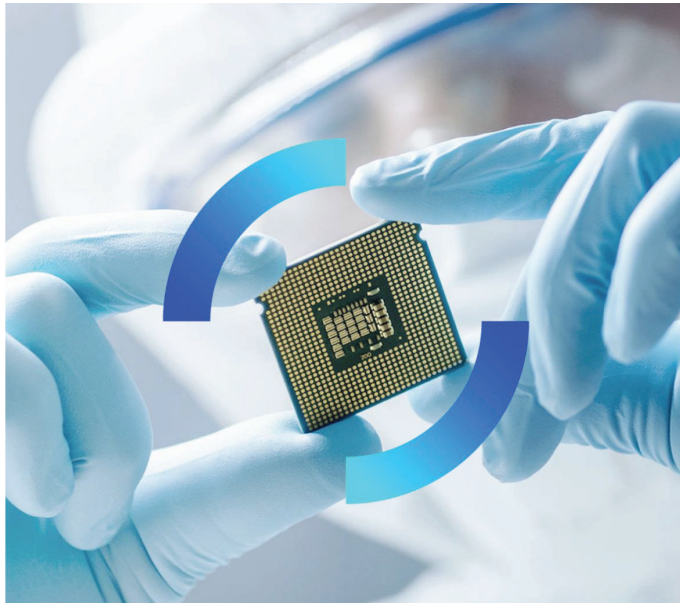
misis.ru

20

**Спасибо  
за внимание!**

Ленинский проспект, д. 4  
Москва, 119049  
тел. +7 (495) 955-00-32  
e-mail: kancela@misis.ru  
misis.ru

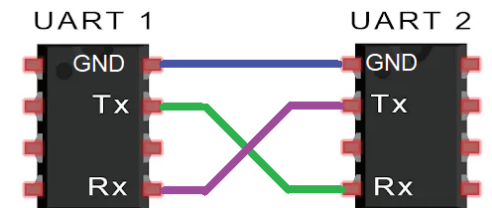




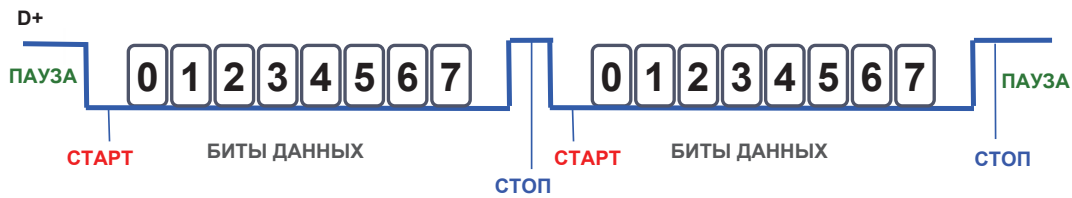
## UART протокол передачи

Universal Asynchronous Receiver-Transmitter,  
Универсальный Асинхронный Приемо-Передатчик.

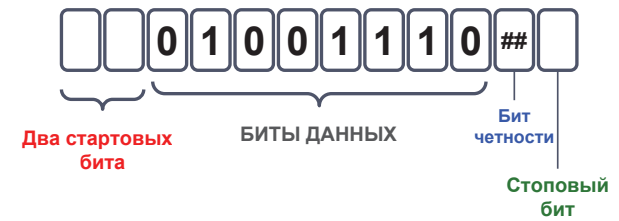
**RX** - Receiver передатчик  
**TX** - Transmitter приемника



## UART протокол передачи



## Служебные сигналы



## Настройка USART

### Управляющие регистры

**UDR0** - входной/выходной регистр данных

**UCSR0A, UCSR0B, UCSR0C** - регистры управления

**UBRR0H, UBRR0L** - регистры управления скоростью передачи

## Регистр UDR0

входной/выходной регистр данных

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



## Регистр UCSR0A

регистры управления

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	RW	R	R	R	R	RW	RW	
Initial Value	0	0	1	0	0	0	0	0	

**Бит 7 RXC0** - флаг завершения приема (1 - если в регистре UDR0 есть непрочитанные данные / 0 - после того, как регистр UDR0 опустошен).

**Бит 6 TXC0** - флаг завершения передачи (1 - после завершения передачи из сдвигового регистра, и если в UDR0 не было загружено нового значения, флаг сбрасывается записью в него 1).

**Бит 5 UDRE0** - флаг опустошения регистра данных UDR0 (устанавливается в 1 после пересылки данных из UDR0 в сдвиговый регистр передатчика и означает что в регистр данных можно загружать новое значение, сбрасывается при записи в UDR0 новых данных).

**Бит 4 FE0** - флаг ошибки кадрирования (при обнаружении ошибки кадрирования (первый стоп-бит равен 0) устанавливается в 1, сбрасывается при приеме стоп-бита равного 1).

**Бит 3 DOR0** - флаг переполнения (устанавливается в 1 если в момент обнаружения нового старт-бита в сдвиговом регистре находится последнее принятое слово, а буфер приемника полон (2 значения)).

**Бит 2 UPEN0** - флаг ошибки контроля четности (устанавливается в 1 при ошибке четности).

**Бит 1 U2X0** - удвоение скорости обмена, если установить в 1 (только в асинхронном режиме, в синхронном следует установить этот бит в 0).

**Бит 0 MPCM0** - режим микропроцессорного обмена (если установлен в 1, режим включен).

## Регистр UCSR0B

регистры управления

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	RW	RW	RW	RW	RW	RW	R	RW	
Initial Value	0	0	0	0	0	0	0	0	

**Бит 7 RXCIE0** - разрешение прерывания при завершении приема если установлен в 1.

**Бит 6 TXCIE0** - разрешение прерывания при завершении передачи если установлен в 1.

**Бит 5 UDRIE0** - разрешение прерывания при очистке регистра данных если установлен в 1.

**Бит 4 RXEN0** - разрешение приема если установлен в 1.

**Бит 3 TXEN0** - разрешение передачи если установлен в 1.

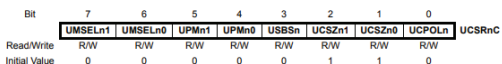
**Бит 2 UCSZ02** - формат посылки данных (используется совместно с битами UCSZ01 и UCSZ00 регистра UCSR0C).

**Бит 1 RXB80** - 8-й разряд принимаемых данных при использовании 9-разрядных слов.

**Бит 0 TXB80** - 8-й разряд передаваемых данных при использовании 9-разрядных слов.

## Регистр UCSR0C

регистры управления



**Биты 7 и 6** UMSEL01 и UMSEL00 отвечают за режим работы USART0:

UMSEL01	UMSEL00	Режим
0	0	Асинхронный режим
0	1	Синхронный режим
1	0	Резерв
1	1	Ведущий SPI

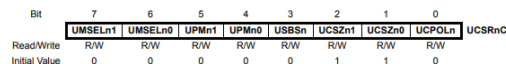
**Биты 5 и 4** UPM01 и UPM00 отвечают за режим работы системы контроля и формирования четности USART0:

UPM01	UPM00	Режим
0	0	Выключен
0	1	Резерв
1	0	Проверка четности
1	1	Проверка нечетности

**Бит 3** USBS0 устанавливает количество стоп битов (1 стоп-бит если сброшен в 0 / 2 стоп-бита если установлен в 1).

## Регистр UCSR0C

регистры управления



**Бит 2** UCSZ02 (2) регистра UCSR0B и **биты 2 и 1** UCSZ01 и UCSZ00 (2, 1) регистра UCSR0C - устанавливают длину передаваемых посылок:

UCSZ02	UCSR01	UCSZ00	Длина передаваемых посылок
0	0	0	5 бит
0	0	1	6 бит
0	1	0	7 бит
0	1	1	8 бит
1	0	0	Резерв
1	0	1	Резерв
1	1	0	Резерв
1	1	1	9 бит

**Бит 0** UCPOL0 устанавливает полярность тактовых сигналов:

- 0 - передача по спаду, прием по нарастанию
- 1 - передача по нарастанию, прием по спаду

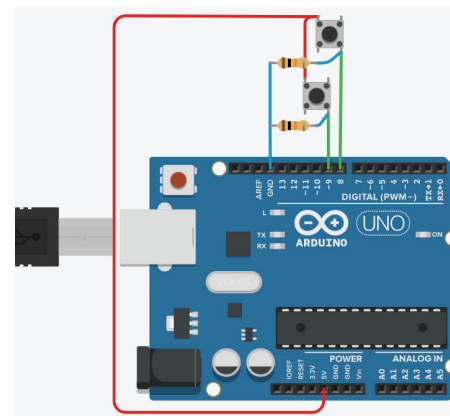
## Регистр UBRRH, UBRR0L

регистры управления скоростью передачи

Bit	15	14	13	12	11	10	9	8	
	-	-	-	-	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

$$UBRR0 = \text{CLK}/(16 \cdot \text{Скорость}) - 1$$

## Экспериментальная установка

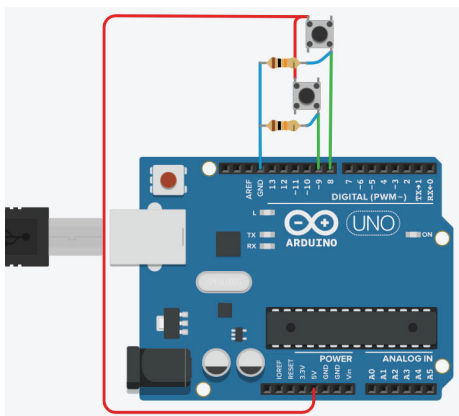


```

1. #include <avr/io.h>
2. #include <util/delay.h>
3. #define F_CPU 16000000 // Рабочая частота контроллера
4. #define BAUD 9600L // Скорость обмена данными
   // Согласно заданной скорости подсчитываем значение для регистра UBRR
1. #define UBRRL_value (F_CPU/(BAUD*16))-1 void init_USART() {
2.   UBRR0L = UBRRL_value; //Младшие 8 бит UBRRL_value
3.   UBRR0H = UBRRL_value >> 8; //Старшие 8 бит UBRRL_value
4.   UCSR0B |= (1 << TXEN0); //Бит разрешения передачи
5.   UCSR0B |= (1 << UCSZ02); //Установиваем формат 8 бит данных
6.   UCSR0C |= (1 << UCSZ00) | (1 << UCSZ01);
7. }
8. void send_UART(char value) {
   // Ожидаем когда очистится буфер передачи
1.   while (!(UCSR0A & (1 << UDRE0)));
2.   UDR0 = value; // Помещаем данные в буфер, начинаем передачу
3. }

```

## Экспериментальная установка

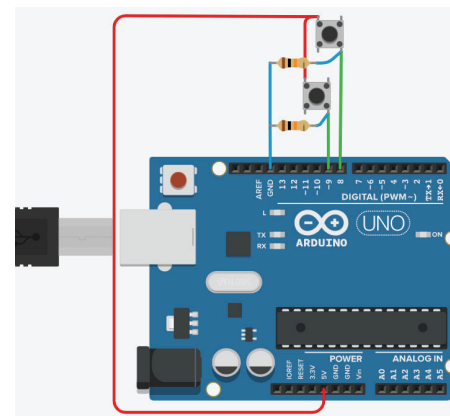


```

17.int main(void)
18.{
19.  init_pin();
20.  init_USART(); //инициализация USART в режиме 9600/8-N-1
21.  send_UART(0x4F); //посылаем ASCII код знака 'O'
22.  send_UART(0x4B); //посылаем ASCII код знака 'K'
23.  send_UART(0x21); //посылаем ASCII код знака '!'
24.  send_UART(0x0D); //переход в начало строки
25.  send_UART(0x0A); //переход на новую строку
26.  while(1)
27.  {
28.    if(PINB&(1<<0))// если кнопка нажата
29.    {
30.      send_UART(0x50); // символ ASCII буквы 'P'
31.      send_UART(0x42); //символ ASCII буквы 'B'
32.      send_UART(0x30); //символ ASCII цифры '0'
33.      send_UART(0x0D);
34.      send_UART(0x0A);
35.      _delay_ms(500);
36.    }

```

## Экспериментальная установка

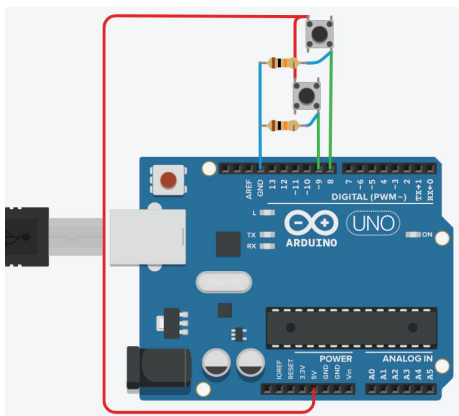


```

37. if(PINB&(1<<1))// если кнопка нажата
38. {
39.  send_UART('P');
40.  send_UART('B');
41.  send_UART('1');
42.  send_UART(0x0D);
43.  send_UART(0x0A);
44.  _delay_ms(500);
45. }
46. }
47.}
48.void init_pin(void)
49.{
50.  DDRB= 0b00000000;//PB0, PB1 input
51.}
52.}
53.}
54.}
55.}
56.}
57.}
58.}
59.}
60.}
61.}
62.}
63.}
64.}
65.}
66.}
67.}
68.}
69.}
70.}
71.}
72.}
73.}
74.}
75.}
76.}
77.}
78.}
79.}
80.}
81.}
82.}
83.}
84.}
85.}
86.}
87.}
88.}
89.}
90.}
91.}
92.}
93.}
94.}
95.}
96.}
97.}
98.}
99.}
100.}

```

## Экспериментальная установка

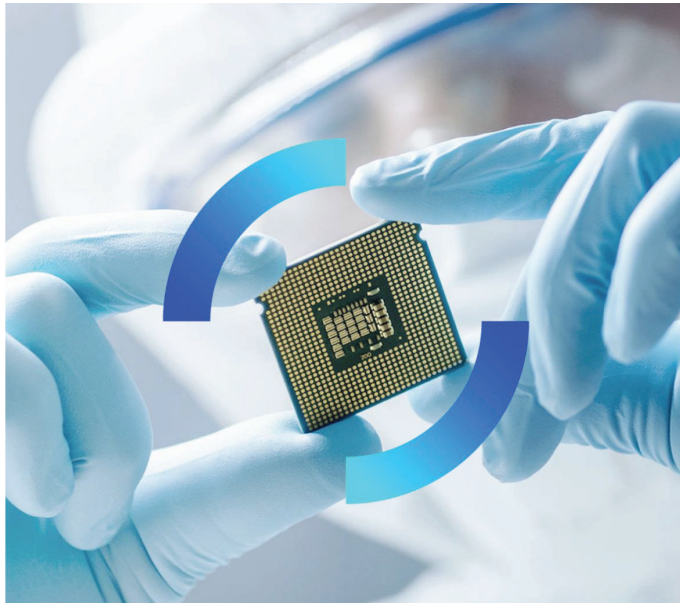


```
1. void setup ()
2. {
3.   Serial.begin(9600);
4.   Serial.println("OK!");
5. }
6. void loop ()
7. {
8.   if(PINB&(1<<0))// если кнопка нажата
9.   {
10.    Serial.println("PB0");
11.    delay(500);
12.  }
13.  if(PINB&(1<<1))// если кнопка нажата
14.  {
15.    Serial.println("PB1");
16.    delay(500);
17.  }
18.}
```

**Спасибо  
за внимание!**

Ленинский проспект, д. 4  
Москва, 119049  
тел. +7 (495) 955-00-32  
e-mail: kancela@misis.ru  
misis.ru





## Прерывание

Прерывания прекращают работу основной программ для того чтобы выполнить более приоритетную, определяемую внутренними или внешними событиями, влияющими на работу микроконтроллера.

## Регистр MCUCR

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	-	<b>BODS</b>	<b>BODSE</b>	<b>PUD</b>	-	-	<b>IVSEL</b>	<b>IVCE</b>	MCUCR
Read/Write	R	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**1 Бит IVCEL** указывает где будет располагаться таблица векторов прерывания:

- 0 - по адресу 0x0002
- 1 - по адресу начала загрузчика + 0x0002

**0 Бит IVCE** разрешает изменение бита IVSEL в течении 4 машинных циклов после установки его в 1.

## Прерывание

### Регистр статуса SREG

Bit	7	6	5	4	3	2	1	0	SREG
0x3F (0x5F)	I	T	H	S	V	N	Z	C	
Read/Write	RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value	0	0	0	0	0	0	0	0	

Таблица векторов прерывания микроконтроллера ATmega328

0x0000 RESET - сброс  
 0x0002 INT0 - внешнее прерывание 0  
 0x0004 INT1 - внешнее прерывание 0  
 0x0006 PCINT0 - прерывание по изменению состояния нулевой группы выводов  
 0x0008 PCINT1 - прерывание по изменению состояния первой группы выводов  
 0x000A PCINT2 - прерывание по изменению состояния второй группы выводов  
 0x000C WDT - прерывание от сторожевого таймера  
 0x000E TIMER2 COMP A - прерывание от таймера/счетчика T2 при совпадении с A  
 0x0010 TIMER2 COMP B - прерывание от таймера/счетчика T2 при совпадении с B  
 0x0012 TIMER2 OVF - прерывание по переполнению таймера/счетчика T2  
 0x0014 TIMER1 CAPT - прерывание от таймера/счетчика T1 по записи  
 0x0016 TIMER1 COMP A - прерывание от таймера/счетчика T1 при совпадении с A  
 0x0018 TIMER1 COMP B - прерывание от таймера/счетчика T2 при совпадении с B

## Прерывание

### Регистр статуса SREG

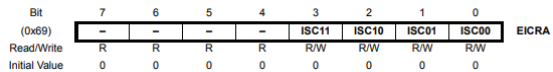
Bit	7	6	5	4	3	2	1	0	SREG
0x3F (0x5F)	I	T	H	S	V	N	Z	C	
Read/Write	RW	RW	RW	RW	RW	RW	RW	RW	
Initial Value	0	0	0	0	0	0	0	0	

Таблица векторов прерывания микроконтроллера ATmega328

0x001A TIMER1 OVF - прерывание по переполнению таймера/счетчика T1  
 0x001C TIMER0 COMP A - прерывание от таймера/счетчика T0 при совпадении с A  
 0x001E TIMER0 COMP B - прерывание от таймера/счетчика T0 при совпадении с B  
 0x0020 TIMER0 OVF - прерывание по переполнению таймера/счетчика T0  
 0x0022 SPI, STC - прерывание по окончании передачи модуля SPI  
 0x0024 USART, RX - прерыванию по окончании приема модуля USART  
 0x0026 USART, UDRE - прерывание по опустошению регистра данных модуля USART  
 0x0028 USART, TX - прерывание по окончании приема модуля USART  
 0x002A ADC - прерывание по завершению преобразования АЦП  
 0x002C EE READY - прерывание по готовности памяти EEPROM  
 0x002E ANALOG COMP - прерывание от аналогового компаратора  
 0x0030 TWI - прерывание от модуля I2C (TWI)  
 0x0032 SPM READY - прерывание по готовности flash памяти

## Внешнее прерывание

### Регистр управления EICRA

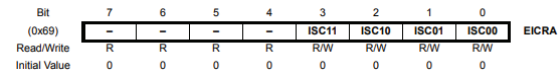


3 Бит ISC11 и 2 Бит ISC10 управляют событиями, в результате которых будет сгенерировано прерывание INT1:

ISC11	ISC10	Событие
0	0	низкий уровень на входе INT1
0	1	любое изменение уровня на входе INT1
1	0	по спадающему сигналу на входе INT1
1	1	по возрастающему сигналу на входе INT1

## Внешнее прерывание

### Регистр управления EICRA



1 Бит ISC01 и 0 Бит ISC00 управляют событиями, в результате которых будет сгенерировано прерывание INTO:

ISC01	ISC00	Событие
0	0	низкий уровень на входе INTO
0	1	любое изменение уровня на входе INTO
1	0	по спадающему сигналу на входе INTO
1	1	по возрастающему сигналу на входе INTO



## Внешнее прерывание

Регистр разрешения внешних прерываний  
**EIMSK**

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	-	-	-	-	-	-	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**1 Бит INT1** разрешает внешние прерывания INT1 при записи в него 1.  
**0 Бит INT0** разрешает внешние прерывания INT0 при записи в него 1.

## Внешнее прерывание

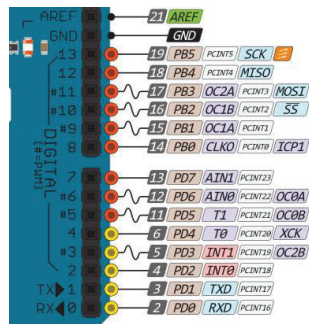
Регистр флагов внешних прерываний **EIFR**

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	-	-	-	-	-	-	INTF1	INTF0	EIFR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

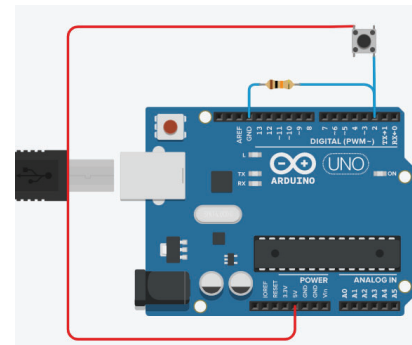
**1 Бит INTF1** устанавливается в 1, если прерывание поступило от INT1.  
**0 Бит INTF0** устанавливается в 1, если прерывание поступило от INT0.

## Внешнее прерывание

Пины 2 и 3 платы разработчика (5 и 6 пины микроконтроллера) обладают дополнительными функциями внешнего прерывания INT0 и INT1 соответственно.



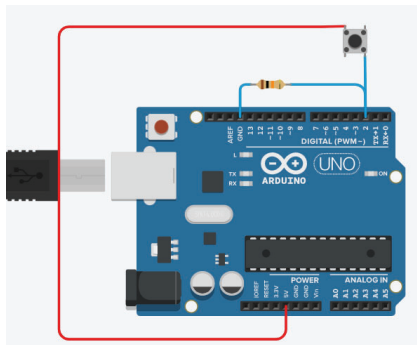
## Внешнее прерывание



```

1. int a=0; // количество импульсов
2. void setup()
3. {
4.   Serial.begin(9600);
5.   int_init();
6.   sei();
7. }
8. void loop()
9. {
10. //выводим количество импульсов
11.Serial.println(" a =");
12.Serial.println(a);
13.}
14.void int_init(void)
15.{
16. //включим прерывания INT0 по нисходящему фронту
17. EICRA |= (1<<ISC00);
18. EICRA |= (1<<ISC01);
19. //разрешаем внешние прерывания INT0
20. EIMSK |= (1<<INT0);
21.}
22. // обработчик прерывания
23.ISR(INT0_vect)
24.{
25. a++;
26.}
    
```

## Внешнее прерывание



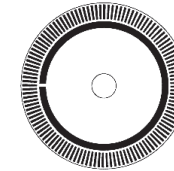
```

1. int a=0; // количество импульсов
2. void setup()
3. {
4.   Serial.begin(9600);
5.   attachInterrupt(0, INT0_vect, FALLING);
6. }
7. void loop()
8. {
9.   //выводим количество импульсов
10.  Serial.print(" a =");
11.  Serial.println(a);
12. }
13. void INT0_vect()
14. {
15.   a++;
16. }

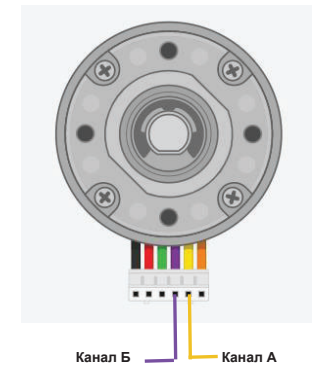
```

## Инкрементный энкодер

**Энкодер (Encoder)** (от англ. encode - преобразовывать) - это устройство (прибор, датчик) для преобразования угловых положений или линейных перемещений в аналоговый или цифровой сигнал.



Главная задача инкрементального энкодера - это расчет единичных импульсов за один цикл. Один цикл равен - одному обороту диска энкодера.



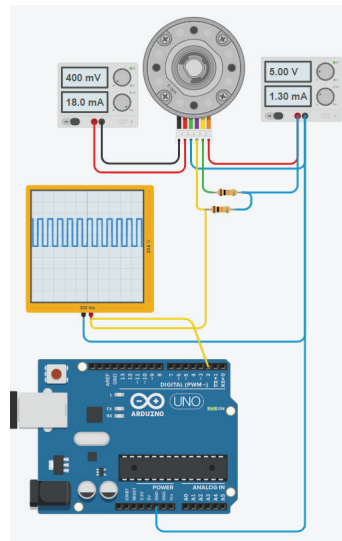
## Определение скорости

Прерывание + Энкодер = скорость

```

1. unsigned long t=0;
2. volatile int a=0; // количество импульсов
3. float speed=0;
4. void setup()
5. {
6.   Serial.begin(9600);
7.   int_ini();
8.   sei();
9. }
10. void loop()
11. {
12. //выводим количество импульсов
13. if (millis() - t>250)
14. {
15.   Serial.print(" a =");
16.   Serial.println(a);

```



misis.ru

13

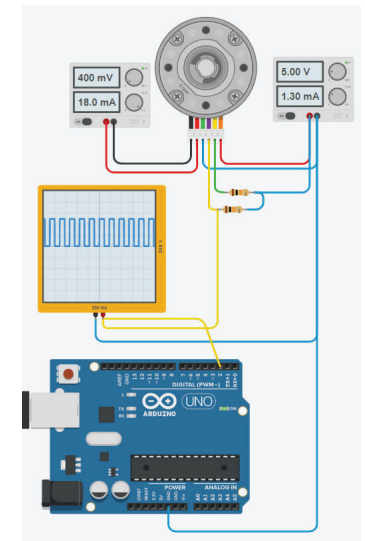
## Определение скорости

Прерывание & Энкодер

```

17. speed = (a/2.5);
18. a=0;
19. t=millis();
20. Serial.print(" speed =");
21.   Serial.println(speed);
22. }
23. }
24. void int_ini(void)
25. {
26. //включим прерывания INT0 по нисходящему фронту
27. EICRA |= (1<<ISC00);
28. EICRA |= (1<<ISC01);
29. //разрешим внешние прерывания INT0
30. EIMSK |= (1<<INT0);
31. }

```



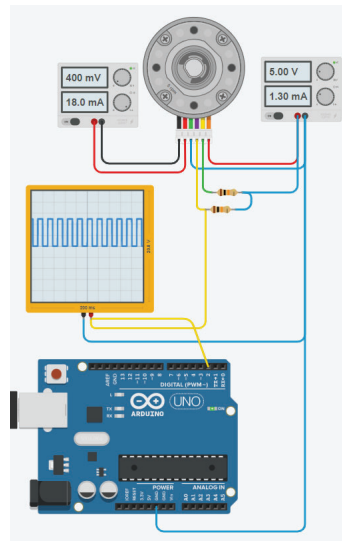
misis.ru

14

## Определение скорости

### Прерывание & Энкодер

```
32. // обработчик прерывания
33. ISR(INT0_vect)
34. {
35.   a++;
36. }
```

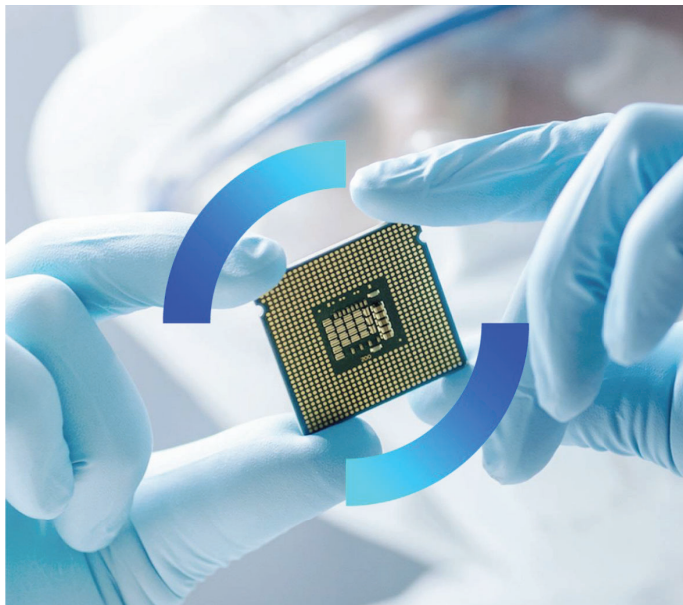


misis.ru

**Спасибо  
за внимание!**

Ленинский проспект, д. 4  
Москва, 119049  
тел. +7 (495) 955-00-32  
e-mail: [kancela@misis.ru](mailto:kancela@misis.ru)  
[misis.ru](http://misis.ru)





## EEPROM постоянная память данных

EEPROM (англ. Electrically Erasable Programmable Read-Only Memory) — электрически стираемое перепрограммируемое ПЗУ- энергонезависимая память данных в которой данные будут храниться даже при отключении питания микроконтроллера.

EEPROM микроконтроллер ATmega328P обладает емкостью 1024 Байта

### Регистры EEPROM

В состав EEPROM микроконтроллера atmega328 входят следующие регистры:

**EEAR** (16 бит) - регистр адреса

**EEDR** - регистр данных

**EECR** - регистр управления

## EEPROM

### EEARH и EEARL -регистры адреса

Bit	15	14	13	12	11	10	9	8	
0x22 (0x42)	-	-	-	-	-	-	-	EEAR8	EEARH
0x21 (0x41)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	X	
	X	X	X	X	X	X	X	X	

## EEPROM

### EEDR - регистр данных

Bit	7	6	5	4	3	2	1	0	
0x20 (0x40)	MSB							LSB	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

## EEPROM

### EECR - регистр управления

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	-	-	EEM1	EEM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

## EEPROM

### EECR - регистр управления

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	-	-	EEM1	EEM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

5 бит EEM1 и 4 бит EEM0 управляют режимами обновления EEPROM:

EEM1	EEM0	Действия
0	0	стирание старого значения и запись нового (3.4 мс)
0	1	только стирание старого значения (1.8 мс)
1	0	только запись нового значения (1.8 мс)
1	1	резерв



# EEPROM

## EECR - регистр управления

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	-	-	EEPМ1	EEPМ0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

**3 Бит EERIE** разрешает прерывания по завершению записи в EEPROM при записи в него 1.

**2 Бит EEMPE** разрешает запись в EEPROM если в него записать 1 (сбрасывается в 0 через 4 машинных цикла).

**1 Бит EEPE** управляет записью в EEPROM (если записать в него 1, то будет произведена запись данных из регистра EEDR в EEPROM по адресу EEAR).

**0 Бит EERE** управляет чтением в EEPROM (если записать в него 1, то будет произведено чтение данных из EEPROM по адресу EEAR в регистр EEDR ).

# EEPROM

## Процедура записи в EEPROM

Процедура записи одного байта в EEPROM состоит из следующих этапов:

1. Дождаться готовности EEPROM к записи данных (ждать пока не сбросится флаг EEPE регистра EECR)
2. Дождаться завершения записи во FLASH-память программ (ждать пока не сбросится флаг SPEN регистра SPMCR)
3. Загрузить байт данных в регистр EEDR, а требуемый адрес - в регистр EEAR (при необходимости)
4. Установить лог. 1 флаг EEMPE регистра EECR для разрешения записи в EEPROM
5. Записать в разряд EEPE регистра EECR лог. 1 в течении 4 машинных циклов. После установки этого разряда процессор пропускает 3 машинных цикла перед выполнением следующей инструкции.

```

1. void EEPROM_write(unsigned int uiAddress, unsigned char ucData){
2. while(EECR & (1<<EEPE)); // проверка готовности EEPROM
3. EEARH = ((uiAddress & 0xF0) << 2); // регистр адреса H
4. EEARL = uiAddress & 0x0F; // регистр адреса L
5. EEDR = ucData; // регистр данных
6. EECR |= (1<<EEMPE); // Разрешение записи в EEPROM
7. EECR |= (1<<EEPE); // Запись в EEPROM
8. }
    
```

## EEPROM

### Процедура чтения из EEPROM

1. Для чтения одного байта из EEPROM необходимо:
2. Проконтролировать состояние флага EEPF. Дело в том, что пока выполняется операция записи в EEPROM память (флаг EEPF установлен), нельзя выполнять ни чтения EEPROM памяти, ни изменения регистра адреса.
3. Загрузить требуемый адрес в регистр EEARH и EEARL.
4. Установить в лог. 1 разряд EERE регистра EECR.
5. Когда запрошенные данные будут помещены в регистр данных EEDR, произойдет аппаратный сброс этого разряда. Однако следить за состоянием разряда EERE для определения момента завершения операции чтения не требуется, т. к. операция чтения из EEPROM всегда выполняется за один машинный цикл. Кроме того, после установки разряда EERE в лог. 1 процессор пропускает 4 машинных цикла перед началом выполнения следующей инструкции.

```
1. unsigned char EEPROM_read(unsigned int uiAddress){
2.   while(EECR & (1<<EEPF)); // проверка готовности EEPROM
3.   EEARH = ((uiAddress & 0xF0) << 2); // регистр адреса H
4.   EEARL = uiAddress & 0x0F; // регистр адреса L
5.   EECR |= (1<<EERE); // чтение EEPROM
6.   return EEDR; // вывод значения
7. }
```

## EEPROM

### Процедура чтения из EEPROM

1. Для чтения одного байта из EEPROM необходимо:
2. Проконтролировать состояние флага EEPF. Дело в том, что пока выполняется операция записи в EEPROM память (флаг EEPF установлен), нельзя выполнять ни чтения EEPROM памяти, ни изменения регистра адреса.
3. Загрузить требуемый адрес в регистр EEARH и EEARL.
4. Установить в лог. 1 разряд EERE регистра EECR.
5. Когда запрошенные данные будут помещены в регистр данных EEDR, произойдет аппаратный сброс этого разряда. Однако следить за состоянием разряда EERE для определения момента завершения операции чтения не требуется, т. к. операция чтения из EEPROM всегда выполняется за один машинный цикл. Кроме того, после установки разряда EERE в лог. 1 процессор пропускает 4 машинных цикла перед началом выполнения следующей инструкции.

```
1. unsigned char EEPROM_read(unsigned int uiAddress){
2.   while(EECR & (1<<EEPF)); // проверка готовности EEPROM
3.   EEARH = ((uiAddress & 0xF0) << 2); // регистр адреса H
4.   EEARL = uiAddress & 0x0F; // регистр адреса L
5.   EECR |= (1<<EERE); // чтение EEPROM
6.   return EEDR; // вывод значения
7. }
```

## EEPROM

### Библиотека EEPROM.h.

**EEPROM.write(адрес, данные)** - пишет данные (только byte!) по адресу

**EEPROM.update(адрес, данные)** - обновляет байт данных, находящийся по адресу.

**EEPROM.read(адрес)** - читает и возвращает байт данных, находящийся по адресу

**EEPROM.put(адрес, данные)** - записывает данные любого типа (типа переданной переменной) по адресу

**EEPROM.get(адрес, данные)** - читает данные по адресу и сам записывает их в данные - указанную переменную

**EEPROM[]** - библиотека позволяет работать с EEPROM памятью как с обычным массивом типа byte (uint8\_t)

```
1. #include <EEPROM.h>
2. void setup() {
3.   Serial.begin(9600);
4.   // пишем 555 по адресу 123
5.   EEPROM.update(555, 123);
6.   Serial.println(EEPROM.read(555)); // выведет 123
7.   Serial.println(EEPROM[555]);    // выведет 123
8. }
9. void loop() { }
```

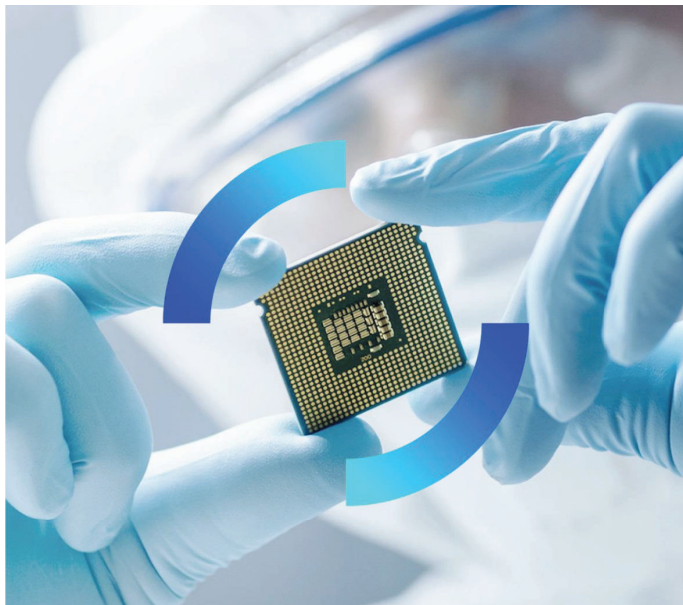
## Спасибо за внимание!

Ленинский проспект, д. 4  
Москва, 119049  
тел. +7 (495) 955-00-32  
e-mail: kancela@misis.ru  
misis.ru

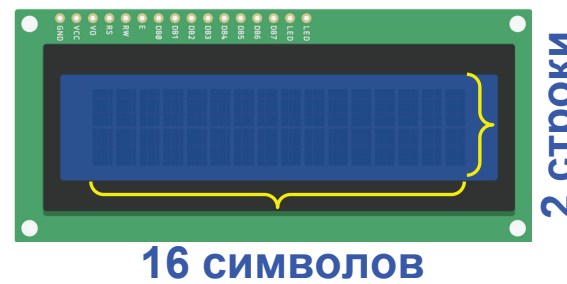


## Программирование микроконтроллеров

Проекты с использованием  
микроконтроллера

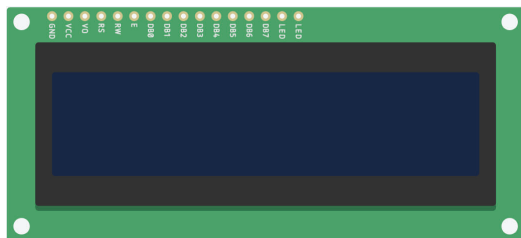


## LCD дисплей 1602



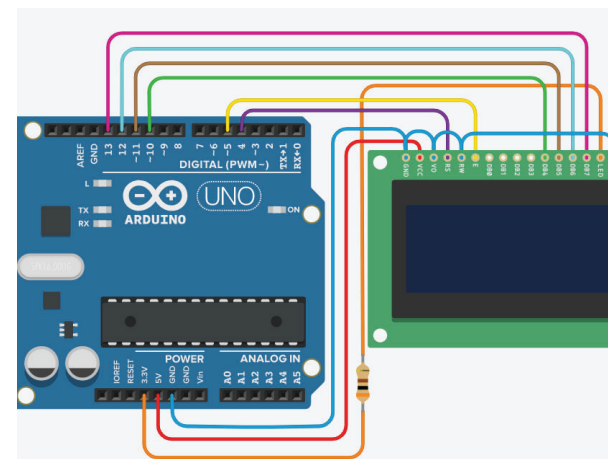
- Символьный тип отображения, есть возможность загрузки символов;
- Светодиодная подсветка;
- Контроллер HD44780;
- Напряжение питания 5В;
- Формат 16x2 символов;

## LCD дисплей 1602



DB0–DB7 – отвечают за входящие/исходящие данные;  
 RS – высокий уровень означает, что сигнал на выходах DB0–DB7 является данными, низкий – командой;  
 R/W – определяет направление данных (чтение/запись). Если только выводим данные устанавливаем лог. 0.  
 E – импульс длительностью не менее 500 мс на этом выводе определяет сигнал для чтения/записи данных с выводов DB0–DB7, RS и W/R;  
 V0 – используется для задания контраста изображения;  
 LEDA, LEDK – питание подсветки (анод и катод),  
 VSS – земля;  
 VDD – питание ЖК-индикатора.

## LCD дисплей 1602



## Библиотека LiquidCrystal

```
#include <LiquidCrystal.h> // подключаем библиотеку
```

```
LiquidCrystal lcd(4, 5, 10, 11, 12, 13);
```

где 4, 5, 10, 11, 12, 13 – номера контактов RS, E, D4, D5, D6, D7.

```
lcd.begin(); // запуск монитора
```

```
lcd.setCursor(); // устанавливает курсор в заданную позицию
```

```
lcd.print(); // вывод информации на экран дисплея
```

## Библиотека LiquidCrystal

пользовательские символы

### createChar()

создает пользовательский символ (глиф) для использования на жидкокристаллическом дисплее. Поддерживаются до восьми символов 5×8 пикселей (нумерация с 0 до 7). Создание каждого пользовательского символа определяется массивом из восьми байтов — один байт для каждой строки. Пять младших значащих битов каждого байта определяют пиксели в этой строке. Чтобы вывести пользовательский символ на экран, используйте функцию write() с номером символа в качестве параметра.

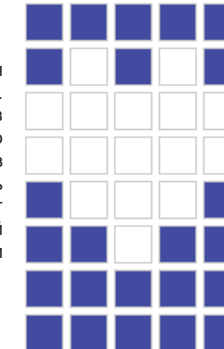
#### lcd.createChar(num, data)

lcd — переменная типа LiquidCrystal

num — номер создаваемого символа (0 to 7)

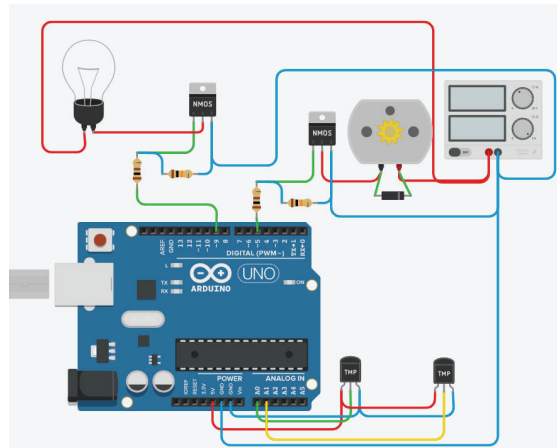
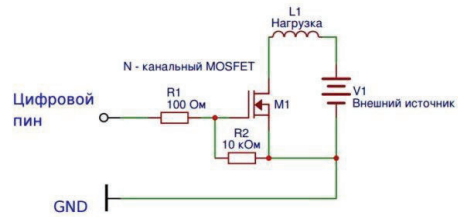
data — данные символьных пикселей

Пиксели



```
byte customChar[8] = {
  0b00000,
  0b01010,
  0b11111,
  0b11111,
  0b01110,
  0b00100,
  0b00000,
  0b00000
};
```

## Проект инкубатор, теплица, климат



## Игра на микроконтроллере

создаем героя

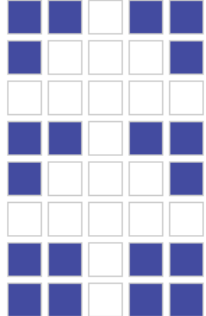


```
byte customChar[8] = {
  0b01110,
  0b01110,
  0b00100,
  0b01110,
  0b10101,
  0b00100,
  0b01010,
  0b10001
};
```

## Игра на микроконтроллере

создаем препятствия

Пиксели



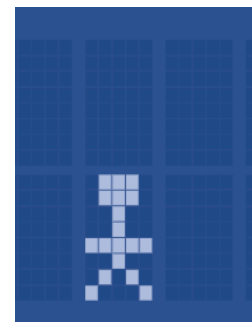
елка

```
byte customChar[8] = {  
  0b00100,  
  0b01110,  
  0b11111,  
  0b00100,  
  0b01110,  
  0b11111,  
  0b00100,  
  0b00100  
};
```

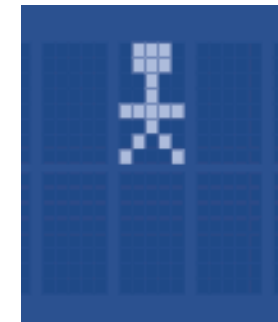
9

## Игра на микроконтроллере

динамика игры прыжок героя



вверх



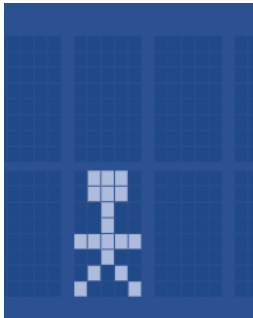
вниз

10



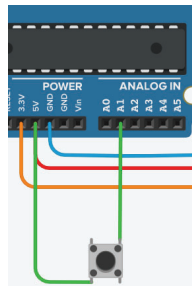
## Игра на микроконтроллере

динамика игры



```
if (digitalRead(A1)) {
  d = 0;
}
if (!digitalRead(A1)) {
  d = 1;
}
```

d - строка 0 или 1



## Игра на микроконтроллере

динамика игры движение

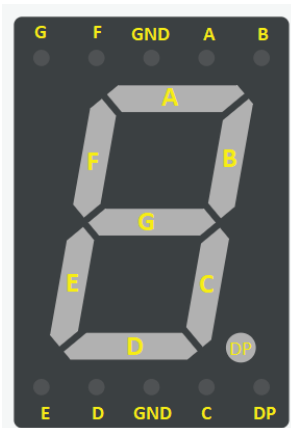
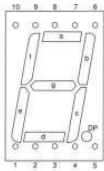
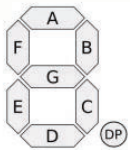
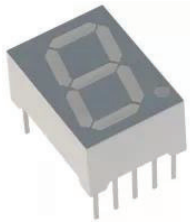


x - движение по оси абсцисс  
y - движение по оси ординат

```
while (x > 0)
{
  LCD.setCursor(x, y);
  LCD.write(byte(i));

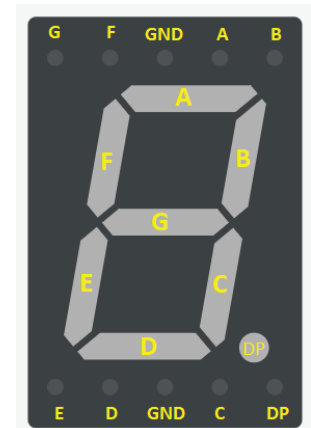
  if (millis() - time2 >= T_period2)
  {
    time2 = millis();
    x--;
  }
}
x = 15;
```

## 7 сегментный индикатор



13

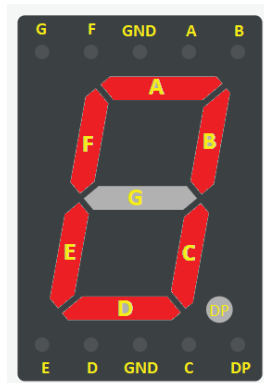
## 7 сегментный индикатор



14

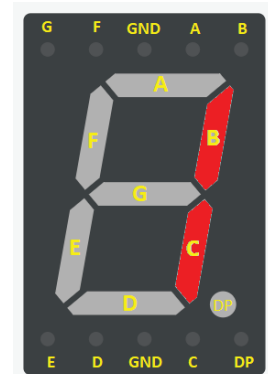
## 7 сегментный индикатор

6	5	4	3	2	1	0
1	1	1	0	1	1	1

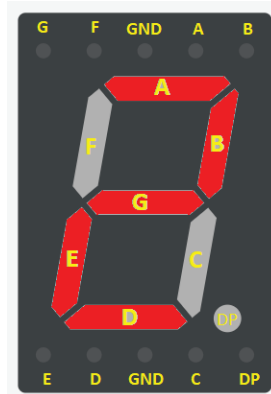
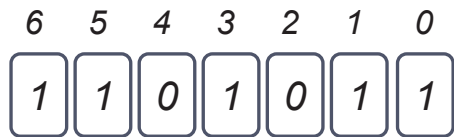


## 7 сегментный индикатор

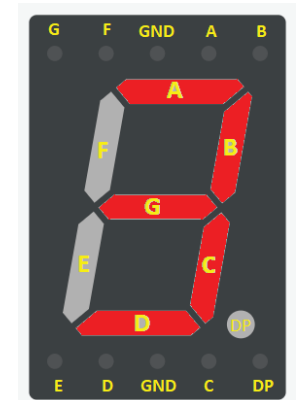
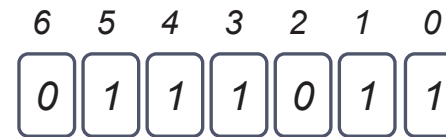
6	5	4	3	2	1	0
0	0	1	0	0	0	1



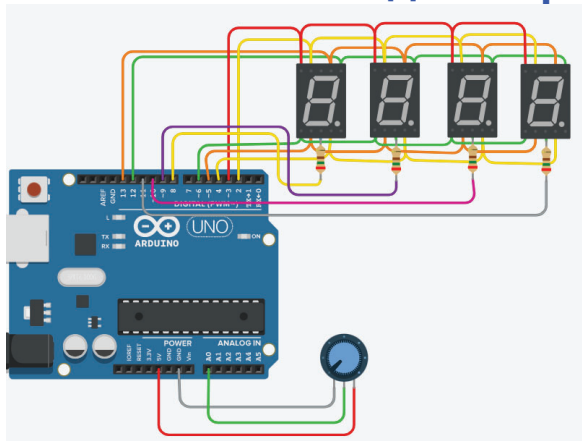
## 7 сегментный индикатор



## 7 сегментный индикатор



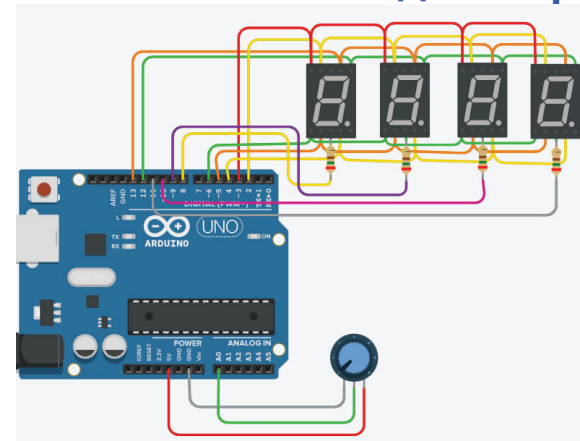
## 7 сегментный индикатор



```

1. byte com[4] = {8, 9, 10, 11};
2. byte seg[7] = {6, 5, 4, 3, 2, 13, 12};
3. int data = 0;
4. byte razr[4] = {0, 0, 0, 0};
5. unsigned long time1 = 0;
6. #define T_timer1 1
7. byte v = 0;
8. byte cifra[10][7] = {
9. { 1, 1, 1, 0, 1, 1, 1 },//0
10. { 0, 0, 1, 0, 0, 0, 1 },//1
11. { 1, 1, 0, 1, 0, 1, 1 },//2
12. { 0, 1, 1, 1, 0, 1, 1 },//3
13. { 0, 0, 1, 1, 1, 0, 1 },//4
14. { 0, 1, 1, 1, 1, 1, 0 },//5
15. { 1, 1, 1, 1, 1, 1, 0 },//6
16. { 0, 0, 1, 0, 0, 1, 1 },//7
17. { 1, 1, 1, 1, 1, 1, 1 },//8
18. { 0, 1, 1, 1, 1, 1, 1 },//9
19.};
    
```

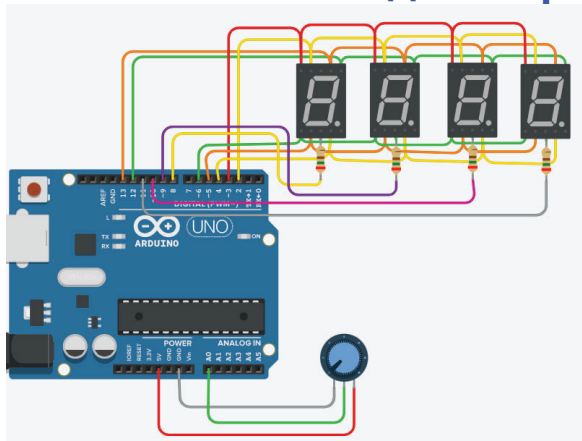
## 7 сегментный индикатор



```

20. void setup()
21. {
22.   for (byte i = 0; i < 4; i++)
23.   {
24.     pinMode(com[i], OUTPUT);
25.     digitalWrite(com[i], 1);
26.   }
27.   byte p = 0;
28.   while (p <= 7)
29.   {
30.     pinMode(seg[p], OUTPUT);
31.     digitalWrite(seg[p], 0);
32.     p++;
33.   }
34. }
    
```

## 7 сегментный индикатор

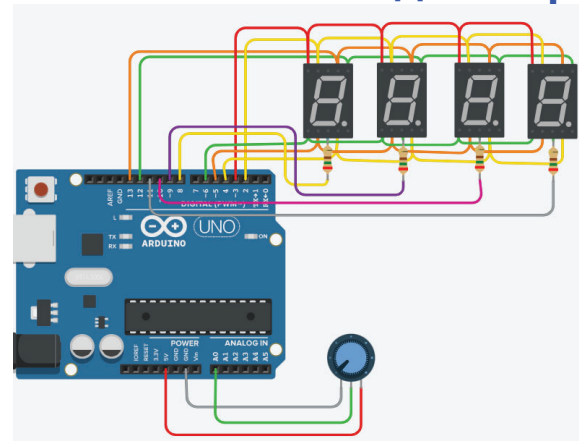


```

35.void loop()
36.{
37.data = analogRead(A0);
38.razr[0] = data / 1000;
39.razr[1] = data / 100 % 10;
40.razr[2] = data / 10 % 10;
41.razr[3] = data % 10;
42.if (millis() - time1 >= T_timer1)
43.{
44.time1 = millis();
45.byte p = 0;
46.while (p <= 7)
47.{
48.pinMode(seg[p], OUTPUT);
49.digitalWrite(seg[p], 0);
50.p++;
51.}

```

## 7 сегментный индикатор

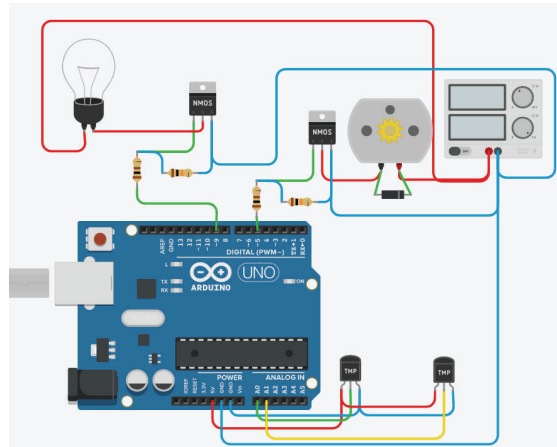
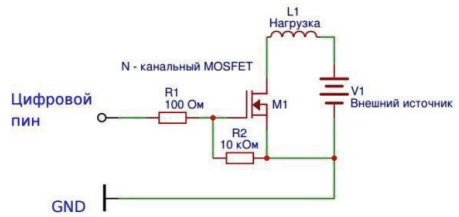


```

52.for (byte i = 0; i < 4; i++)
53.{
54.if (v != i)digitalWrite(com[i], 1);
55.else digitalWrite(com[v], 0);
56.}
57.for (byte j = 0; j <= 9; j++)
58.{
59.if (razr[v] == j)
60.{
61.for (byte z = 0; z <= 7; z++)
62.{
63.digitalWrite(seg[z], cifra[j][z]);
64.}
65.}
66.}
67.v++;
68.}
69.if (v == 4) v = 0;
70.}

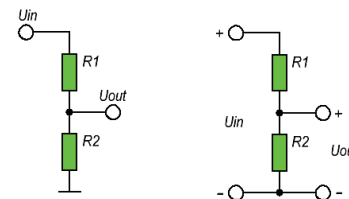
```

## Проект инкубатор, теплица, климат



## Резистивный делитель

Резистивный делитель позволяет получить меньшее из большего напряжение

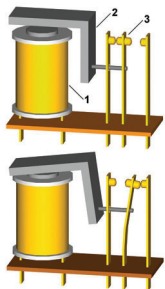


$$U_{out} = U_{in} \times \frac{R_2}{R_1 + R_2}$$

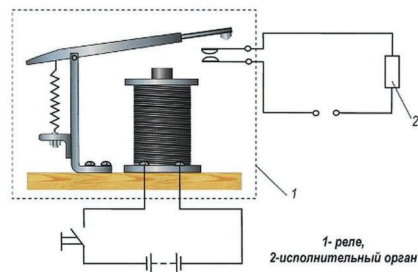
$$I = \frac{U_{in}}{R_1 + R_2}$$

## Электромагнитное реле

Электромагнитное реле — коммутирующее устройство, которое для работы использует электромагнитное поле.



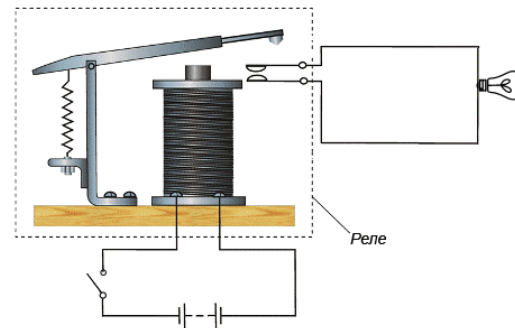
- 1 — электромагнит (обмотка с ферромагнитным сердечником);
- 2 — подвижный якорь;
- 3 — контактная система (переключатель).



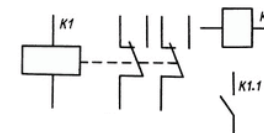
<https://elektrozhatok.ru/info/teoriya/elektromagnitnoe-rele>

## Электромагнитное реле

Электромагнитное реле — коммутирующее устройство, которое для работы использует электромагнитное поле.



<https://www.tria-komm.ru/article/relay-for-automatics/>







**Спасибо  
за внимание!**

Ленинский проспект, д. 4  
Москва, 119049  
тел. +7 (495) 955-00-32  
e-mail: [kancela@misys.ru](mailto:kancela@misys.ru)  
[misys.ru](http://misys.ru)



Текстовый документ, содержащий методические  
рекомендации по подготовке, настройке и  
использованию рекомендованных для занятия  
средств обучения.

## 1. Плата разработчика

Курс построен на изучении 8 битного AVR микроконтроллера ATmega328 фирмы Atmel семейства ATmega. Внешний вид микроконтроллера представлен на рисунке.

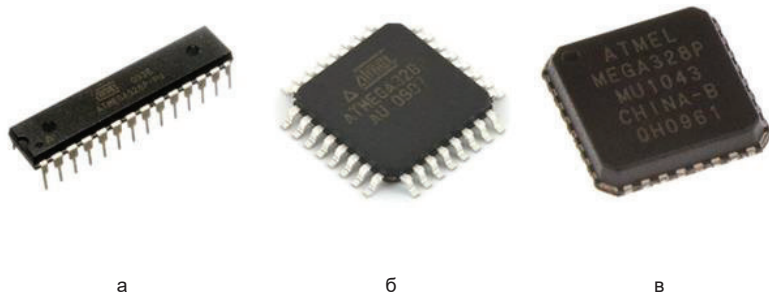


Рисунок 1 Внешний вид микроконтроллера Atmel Atmega328 в различных корпусах

(а - DIP, б - TQFP, в - VQFN)

Рассматриваем микроконтроллер входит в состав платы разработчика Arduino UNO (далее плата разработчика, внешний вид которой представлен на рисунке 2). Данная плата разработчика входит в состав лабораторного оборудования для образовательных школ Москвы проекта "ИТ-класс в московской школе". Внешний вид платы представлен на рисунке.



Рисунок 2 - Внешний вид платы разработчика

Принципиальная схема платы разработчика представлена на рисунке 3.

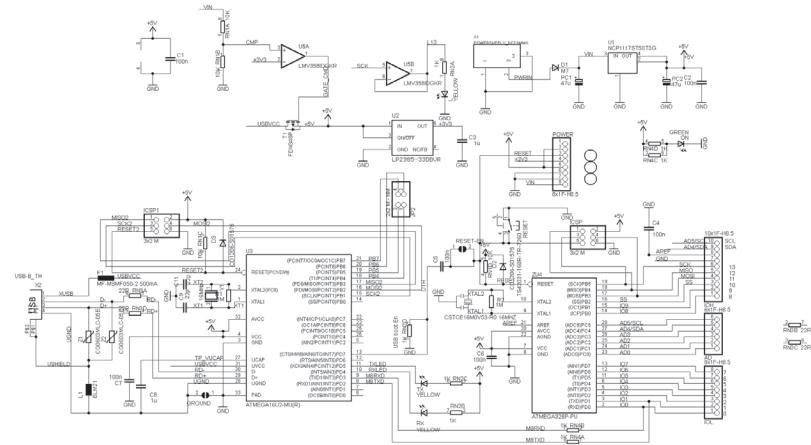


Рисунок 3 Принципиальная схема платы разработчика

Данная плата обладает рядом преимуществ:

- наличие программатора на плате
- возможность подключения к портам микроконтроллера с помощью монтажных проводов, через контактные площадки платы
- наличие DC-DC преобразователя для различного уровня напряжения 3.3 и 5 вольт
- внешний кварцевый резонатор на 16 Мгц.
- питание от USB порта компьютера
- встроенный восстанавливаемый предохранитель.
- разъем для внутрисхемного программирования
- наличие бесплатного программного обеспечения Arduino IDE для программирования микроконтроллера.

Необходимые компоненты для работы на курсе, все компоненты входят в состав лабораторного оборудования проекта "ИТ-класс в московской школе" МАТРЕШКА фирмы Амперка (внешний вид набора представлен на рисунке 4) :

- плата разработчика Arduino Uno;
- резисторы номиналом 250 Ом-10 кОм;
- тактовая кнопка любого типа без фиксации;
- макетная плата;
- LCD экран 1602;
- кабель USB для программирования.



Рисунок 4 - Набор компонентов курса

Для проведения очных занятий и итоговой аттестации используются аудитории с компьютерами, технические требования:

Операционная система:

Windows 7, Windows 8 и Windows 10 (Windows RT не поддерживается) или MacOS (10.6, 10.7, 10.8) или Linux

Аппаратное обеспечение:

- 1) ПЭВМ по количеству учащихся. Минимальные системные требования:
  - Операционная система Windows (XP, Vista, 7, 8) , MacOS (10.6, 10.7, 10.8) , Linux
  - 2 Гб оперативной памяти
  - Процессор 1.5 ГГц
  - 750 Мб свободного дискового пространства
  - Разрешение экрана 1024\*600
  - Microsoft Silverlight 5.0
  - Microsoft.NET 4.0

Доступ в интернет со скоростью не менее 50 Мбит/с и web-браузером предпочтительнее Googlee.Chrome

## 2. Установка программного обеспечения для программирования микроконтроллера

1. Перейти на сайт проекта по ссылке: <https://www.arduino.cc/en/software>
2. Выбрать необходимую операционную систему



## Downloads



### Arduino IDE 2.0.3

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

#### SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

#### DOWNLOAD OPTIONS

**Windows** Win 10 and newer, 64 bits

**Windows** MSI installer

**Windows** ZIP file

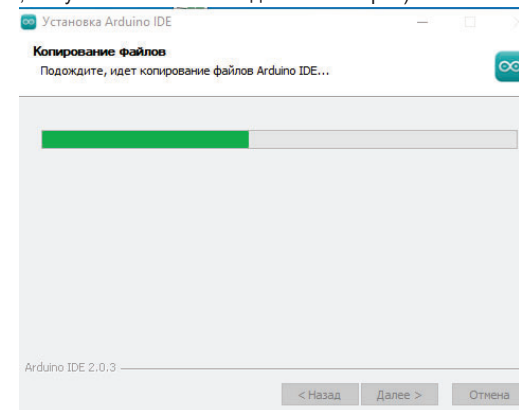
**Linux** AppImage 64 bits (X86-64)

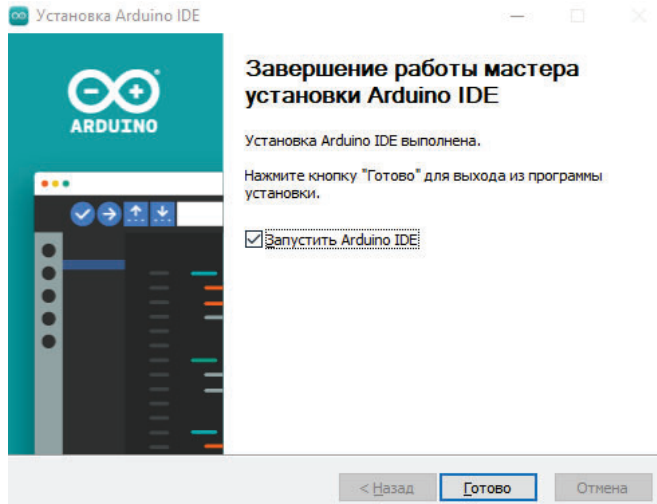
**Linux** ZIP file 64 bits (X86-64)

**macOS** Intel, 10.14: "Mojave" or newer, 64 bits

**macOS** Apple Silicon, 11: "Big Sur" or newer, 64 bits

3. Скачать установочный файл и приступить к установке программы (при возникновении трудностей обратиться к ИТ специалистам образовательного учреждения, т.к. у вас может быть недостаточно прав)

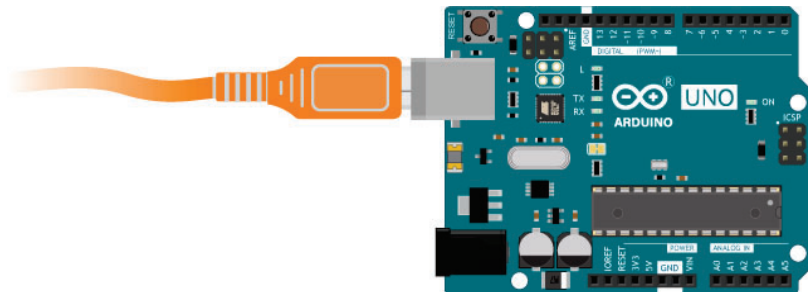




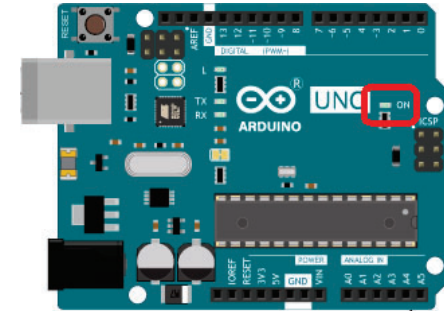
4. После установки программы на вашем компьютере должен появиться ярлык



5. Рекомендуем при запуске программы осуществлять запуск от имени администратора или проверить со специалистами ИТ, чтобы последующие описанные действия вы могли производить самостоятельно.
6. Подключить плату разработчика через USB шнур программирования к компьютеру. Обратите внимание чтобы обратная сторона платы разработчика не соприкасалась с металлическими предметами, во избежания короткого замыкания.

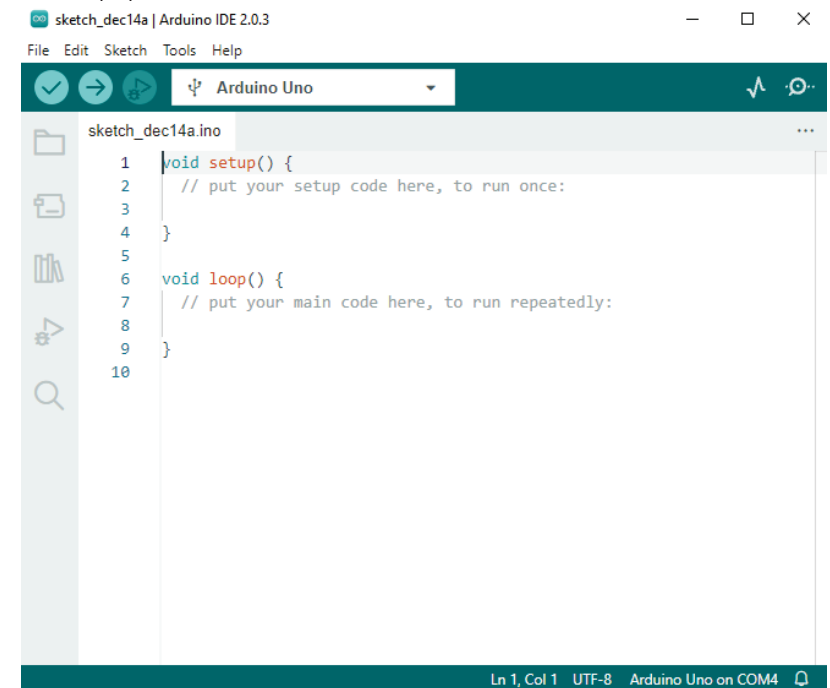


Обратите внимание на индикатор ON, на плате разработчика. Он должен светиться. Если он не светиться, то это может означать что на плате короткое замыкание и сработал предохранитель платы или USB порта. Необходимо оперативно извлечь USB шнур питания платы из компьютера.

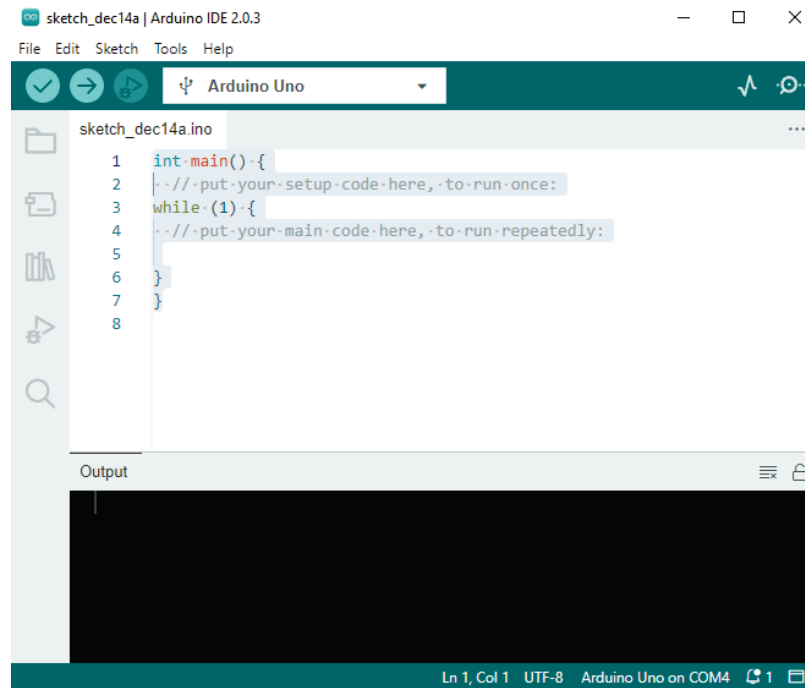


Рекоменуем последовательность действий:

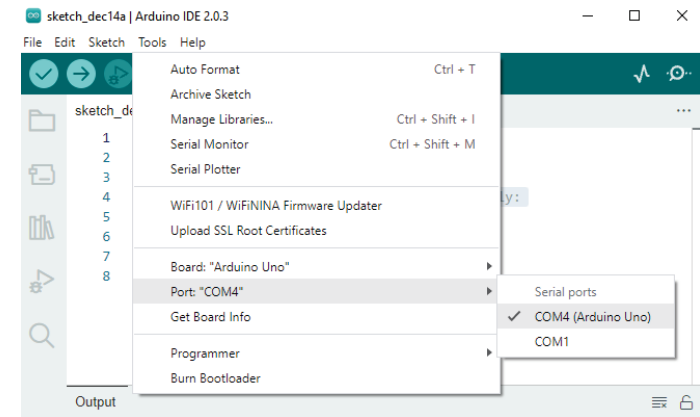
1. Сборка схема
  2. Проверка схемы преподавателем
  3. Подключение схемы к питанию
  4. Выполнение лабораторной работы
  5. Отключение питание
  6. Разборка схемы
7. Перед вами откроется окно с пустым скетчем. По умолчанию стоит возможность писать программный код языком Arduino.



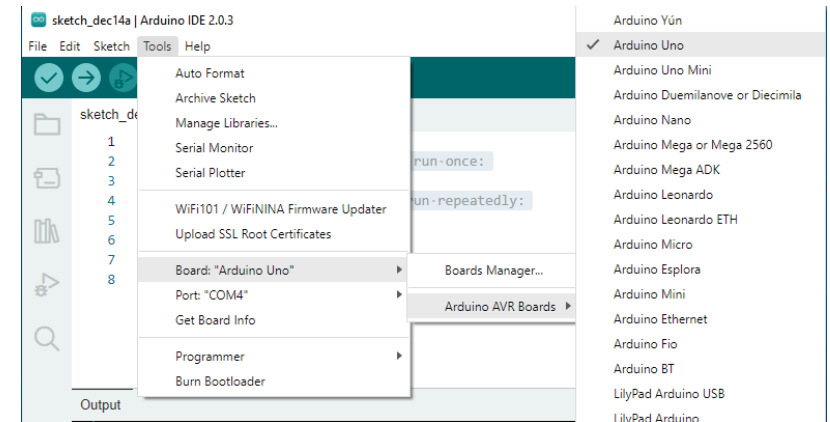
Вы можете при программировании микроконтроллера использовать конструкции языка C++



- Для программирования микроконтроллера необходимо выполнить следующее, установить порт подключения к компьютеру, для этого в меню Tools, выбираем Port и в порту выбираем доступный порт в нашем случае COM4 (Arduino Uno). Смотри рисунок




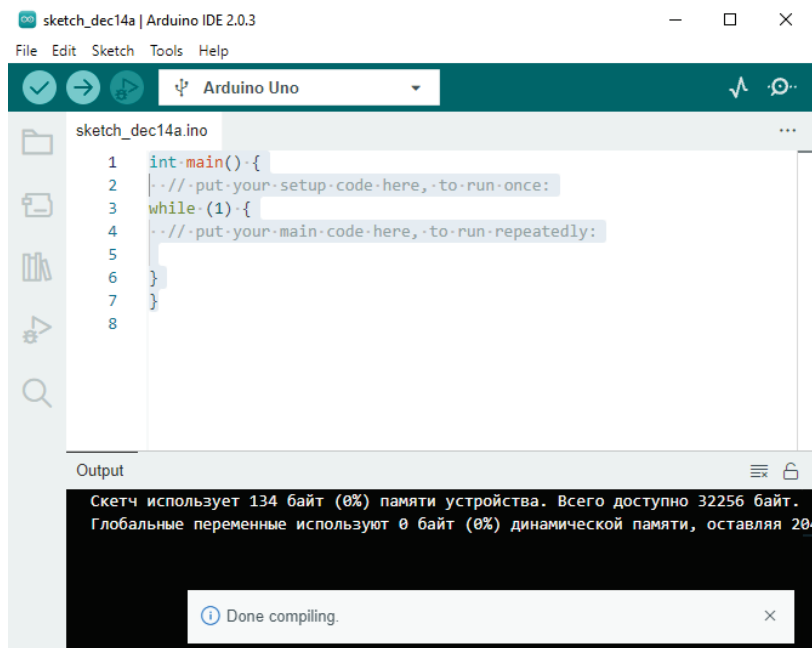
- В разделе Tool в подменю Board выбрать Arduino Uno так как плата разработчика - Arduino Uno



- Если описанные шаги не получилось сделать и у вас не появляется порт, при этом плата подключена. Обратитесь к администратору компьютера необходимо прописать права для данных действий.
- После успешного выполнения подключения вы можете скомпилировать программу:

- без загрузки в микроконтроллер (допустим для проверки кода), для этого

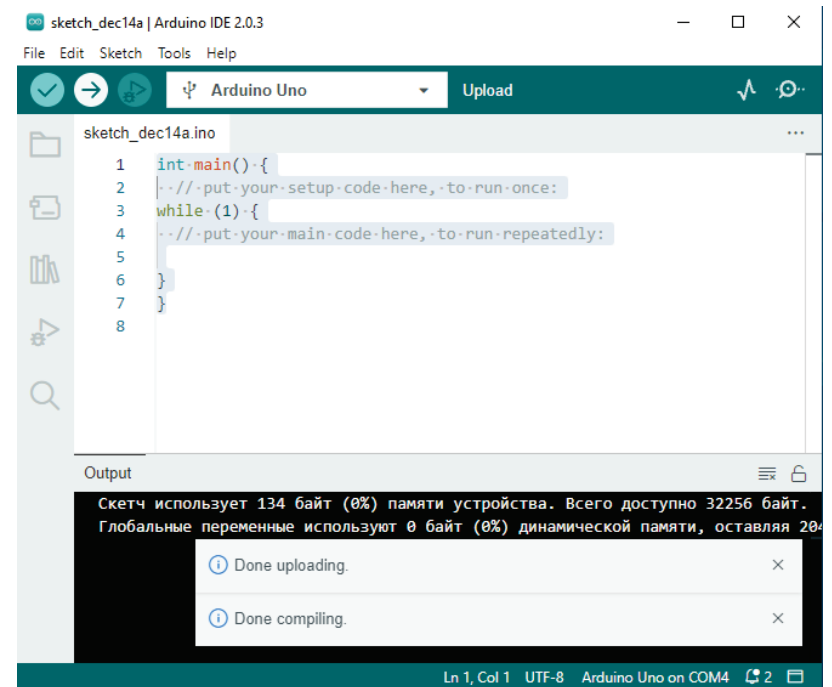
вы должны нажать на кнопку  в поле функциональных кнопок. На экране отобразится сообщение "Done compiling"



- с загрузкой в микроконтроллера, для этого вы должны нажать на кнопку



На экране отобразятся два сообщения “Done compiling” и “Done uploading”

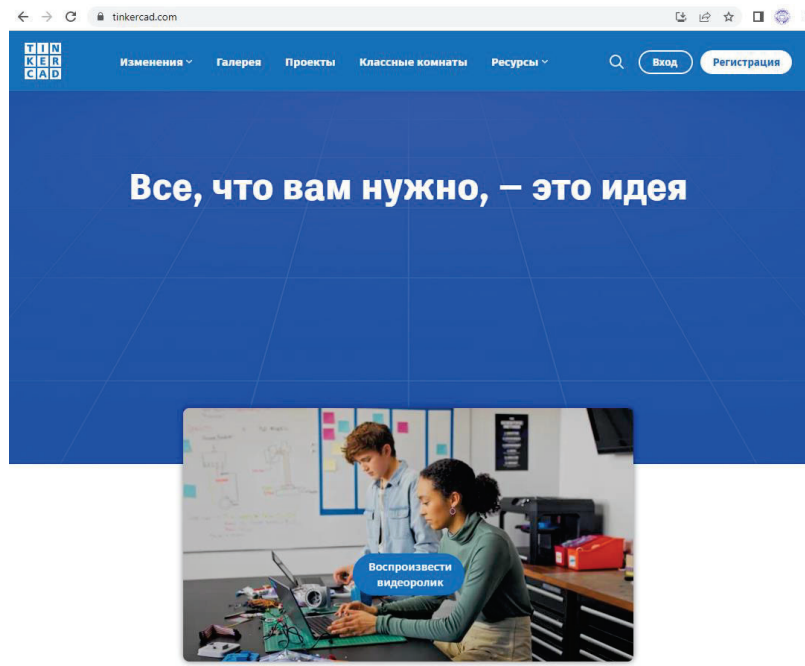


В этом случае ваш программный код загружен на микроконтроллер.

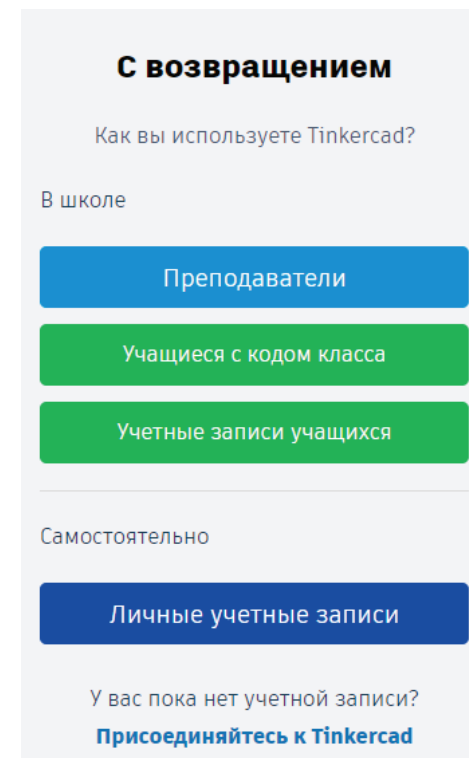
## Программирование микроконтроллера в Tinkercad

Существует еще один способ программирования микроконтроллера с использованием виртуальной среды Tinkercad. Вам понадобится учетка в Google.

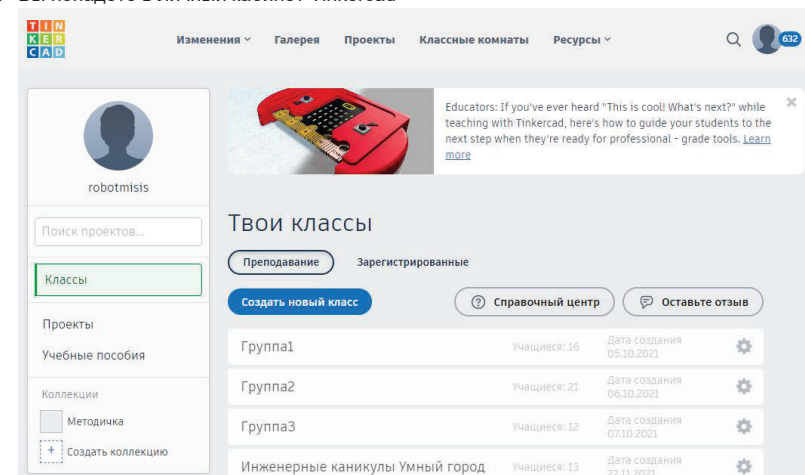
1. Перейдите по ссылке <https://www.tinkercad.com/>



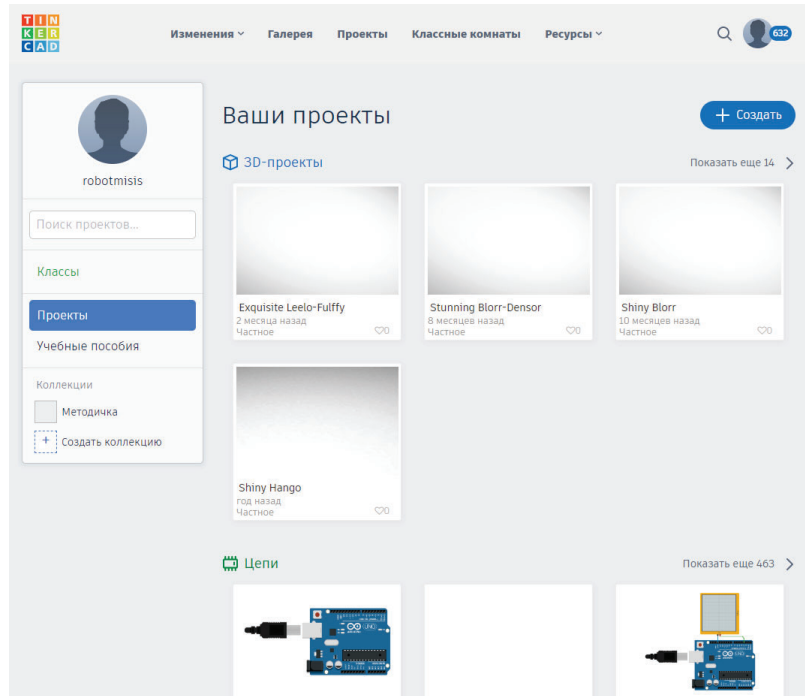
2. Войдите на платформу со своими учетными записями



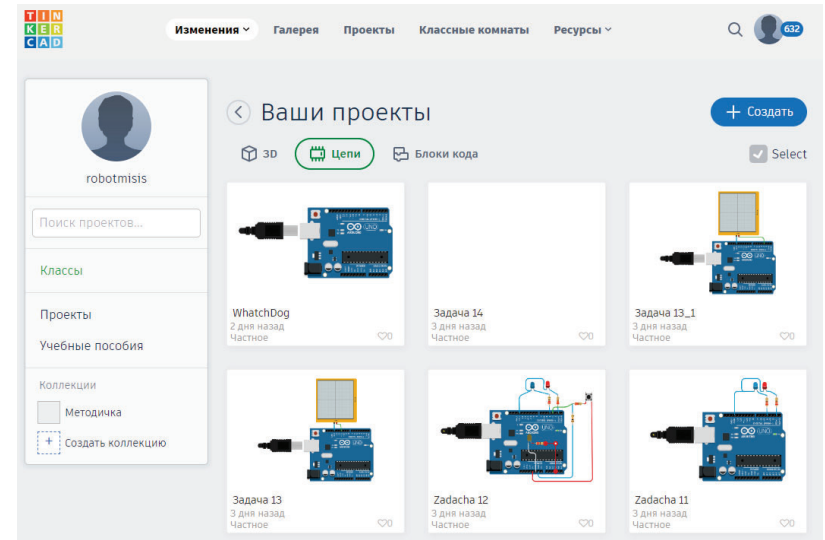
3. Вы попадете в личный кабинет Tinkercad



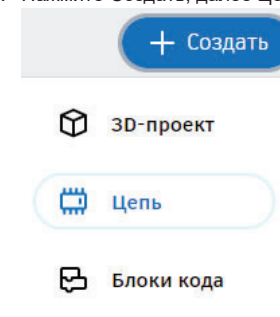
4. Перейдите в раздел проекты



5. Выберите Цепи

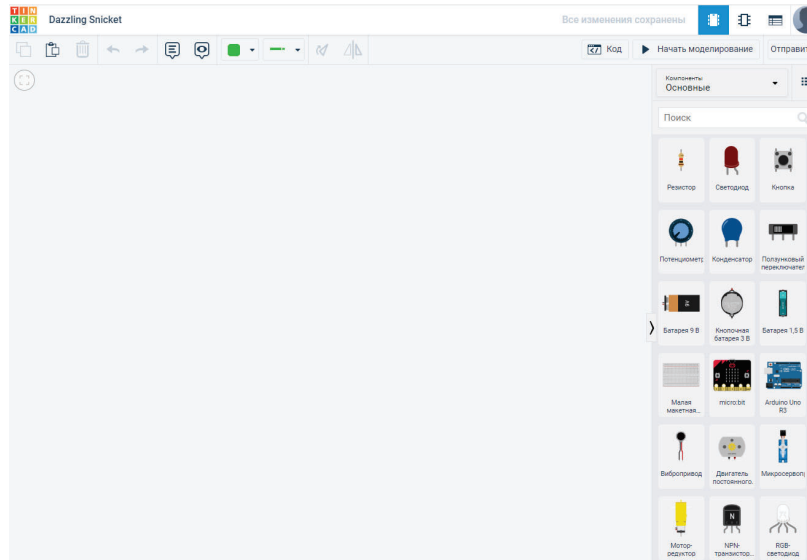


6. Нажмите Создать, далее Цепь



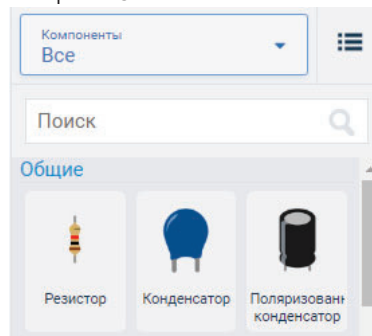
7. Перед вами откроется виртуальная лаборатория электротехники Tinkercad



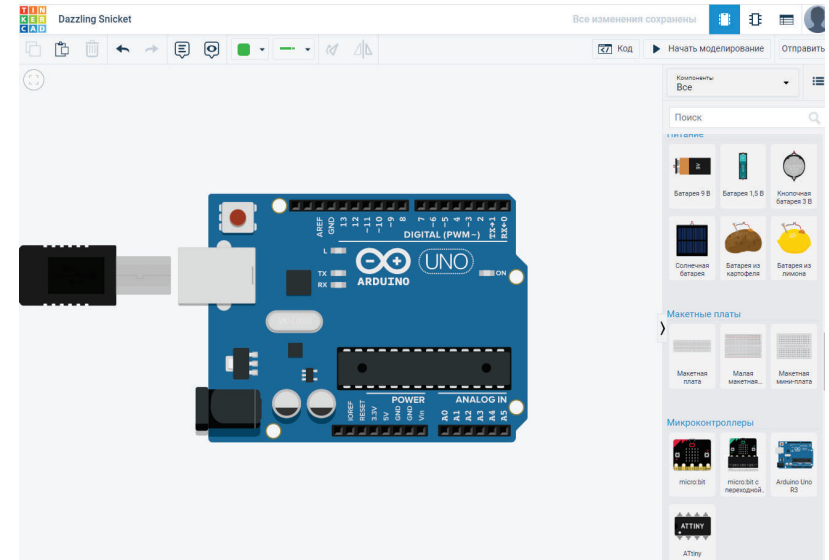


Работа в среде Tinkercad с платой разработчика

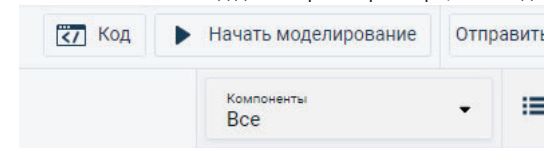
8. В среде Tinkercad в столбце компонентов справа, в разделе компоненты выберите ВСЕ.



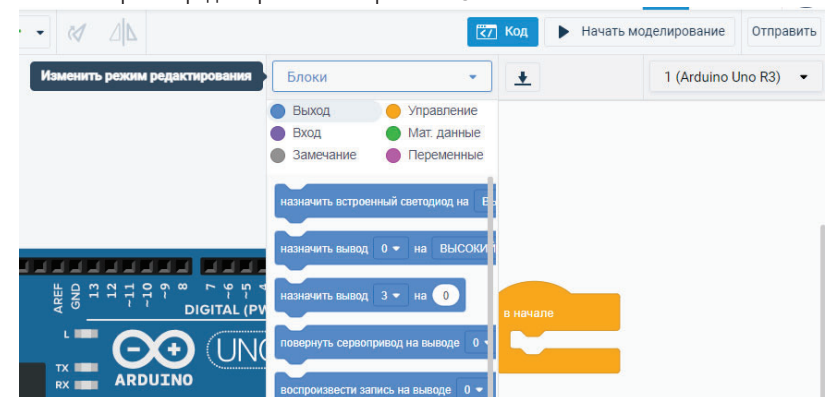
9. В предложенных компонентах найдите плату разработчика Arduino UNO и перетащите на белое поле слева



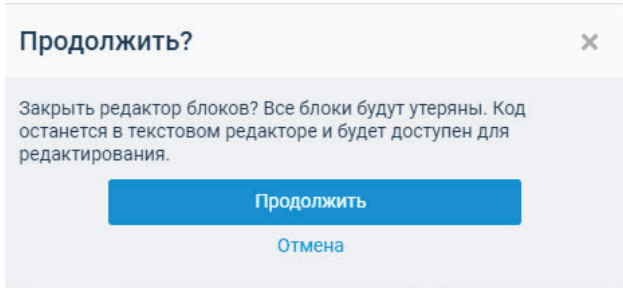
10 Чтобы написать код для микроконтроллера, необходимо зайти в меню КОД



11 В меню режим редактирования выбрать ТЕКСТ



12 Нажать клавишу Продолжить



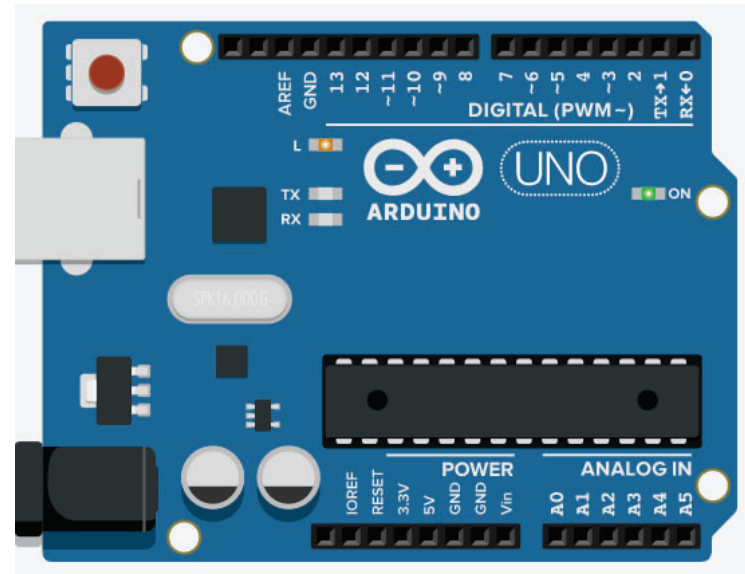
13 Теперь вы можете писать код также как и в приложении Arduino IDE

```
Текст [Download] [Save] [Font] 1 (Arduino Uno R3)
1 // C++ code
2 //
3 void setup()
4 {
5   pinMode(LED_BUILTIN, OUTPUT);
6 }
7
8 void loop()
9 {
10  digitalWrite(LED_BUILTIN, HIGH);
11  delay(1000); // Wait for 1000 millisecond(s)
12  digitalWrite(LED_BUILTIN, LOW);
13  delay(1000); // Wait for 1000 millisecond(s)
14 }
```

Чтобы запустить код, необходимо нажать на кнопку Начать моделировать



14 На экране шнур USB будет помещен в плату разработчика и код начнет исполняться



# Комплект задач с решениями и ответами для организации самостоятельной работы

## Используемые сокращения

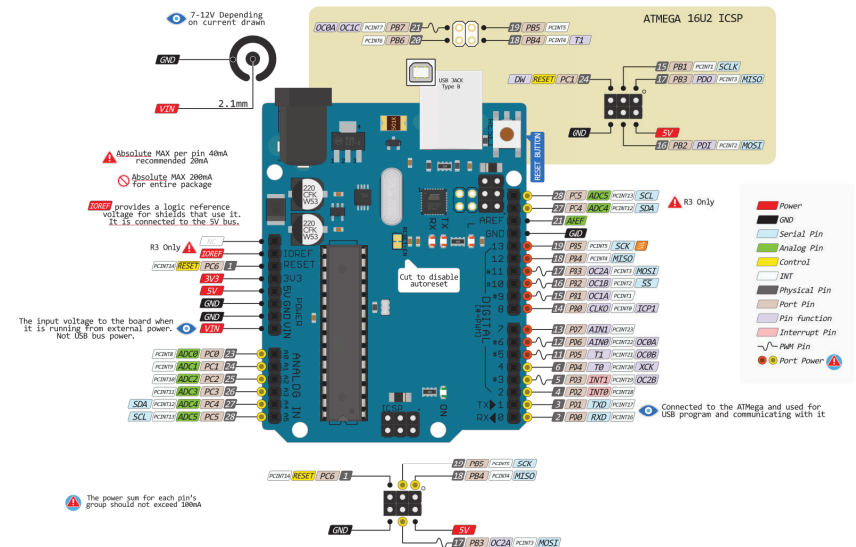
логическая единица - лог. 1

логический ноль - лог. 0

плата Arduino Uno - плата разработчика

Тестирования кода для микроконтроллера Atmega 328 осуществляем на плате разработчика Arduino UNO представленный на рисунке.

## UNO PINOUT



## Раздел 2. Платформа разработчика программируемого микроконтроллера

Порты общего назначения GPIO. Управление нагрузкой на портах микроконтроллера.

12 задач

### БАЗОВЫЙ

Б.1 Разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 1, 2, 4 и 5 порты платы разработчика. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

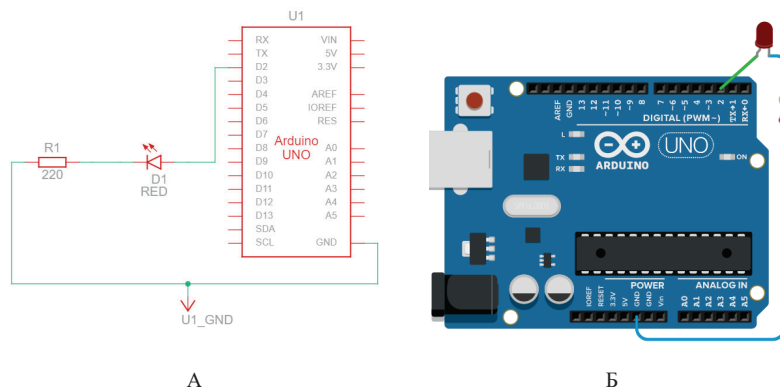
Б.2 Разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 8, 10, 12 и 13 порты платы разработчика. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

Б.3 Разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 1, 2, 8 и 11 порты платы разработчика. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

Б.4 Разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 3, 2, 7 и 12 порты платы разработчика. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

### Типовое решение задачи Б

Используя язык программирования Arduino. Схема для проверки представлена на рисунке. Тестирование наличие лог. 1 на рассматриваем порту осуществляем с помощью светодиода.



Электрическая (А) и эмуляционная (Б) схемы решения задачи

Листинг программы представлен ниже

- ```
1. void setup()
2. {
3.   pinMode(1, OUTPUT); // устанавливаем порты платы в режим вывода сигнала
```

- ```
4.   pinMode(2, OUTPUT);
5.   pinMode(4, OUTPUT);
6.   pinMode(5, OUTPUT);
7.   digitalWrite(1, HIGH); // устанавливаем лог.1 на порту платы
8.   digitalWrite(2, HIGH);
9.   digitalWrite(4, HIGH);
10.  digitalWrite(5, HIGH);
11.
12. }
13. void loop()
14. {
15. }
```

### СРЕДНИЙ

С.1 Разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 1, 2, 4 и 5 порты платы разработчика используя при этом регистры управления портов общего назначения GPIO. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

С.2 Разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 8, 10, 12 и 13 порты платы разработчика используя при этом регистры управления портов общего назначения GPIO. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

С.3 Разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 1, 2, 8 и 11 порты платы разработчика используя при этом регистры управления портов общего назначения GPIO. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

С.4 Разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 3, 2, 7 и 12 порты платы разработчика используя при этом регистры управления портов общего назначения GPIO. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

### Типовое решение задачи С

Используем язык программирования Arduino. Схема для проверки представлена на рисунке. Тестирование наличие лог. 1 на рассматриваем порту осуществляем с помощью светодиода согласно представленной схеме на рисунке. Листинг программы представлен ниже

Листинг программы

- ```
1. void setup() {
2.   DDRD = 0b00110110; // установка пинов 1, 2, 4 и 5 порта D в режим вывода
3.   PORTD = 0b00110110; // установка лог.1 в пины 1, 2, 4 и 5 порта D
4. }
5. void loop()
6. {
7. }
```

### ВЫСОКИЙ

В.1 Используя синтаксис языка C++ разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 1, 2, 4 и 5 порты платы разработчика используя при этом

регистры управления портов общего назначения GPIO. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

В.2 Используя синтаксис языка C++ разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 8, 10, 12 и 13 порты платы разработчика используя при этом регистры управления портов общего назначения GPIO. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

В.3 Используя синтаксис языка C++ разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 1, 2, 8 и 11 порты платы разработчика используя при этом регистры управления портов общего назначения GPIO. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

В.4 Используя синтаксис языка C++ разработайте программу для микроконтроллера Atmega328, которая позволит установить лог. 1 на 3, 2, 7 и 12 порты платы разработчика используя при этом регистры управления портов общего назначения GPIO. Наличие или отсутствие сигнала осуществите с помощью светодиода или вольтметра.

## Типовое решение задачи В

Используем язык программирования C++. Схема для проверки представлена на рисунке. Тестирование наличие лог. 1 на рассматриваем порту осуществляем с помощью светодиода согласно представленной схеме на рисунке. Листинг программы представлен ниже

```
int main(void)
{
  DDRD = 0b00110110; // установка пинов 1, 2, 4 и 5 порта D в режим вывода
  PORTD = 0b00110110; // установка лог.1 в пины 1, 2, 4 и 5 порта D
}
```

## Битовые операции. Приоритеты операций микроконтроллера 12 задач

### БАЗОВЫЙ

Б.1 В устройстве используется микроконтроллер Atmega328. Пины 7, 4 и 2 микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую считывать данные с указанных портов и выводить полученные данные о состоянии данных портов в виде индикации на светодиодах, подключенных к порту С микроконтроллера. Эмулировать работу цифровых датчиков с помощью тумблера.

Б.2 В устройстве используется микроконтроллер Atmega328. Пины 7, 6 и 5 микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую считывать данные с указанных портов и выводить полученные данные о состоянии данных портов в виде индикации на светодиодах, подключенных к порту С микроконтроллера. Эмулировать работу цифровых датчиков с помощью тумблера.

Б.3 В устройстве используется микроконтроллер Atmega328. Пины 5, 4 и 2 микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую считывать данные с указанных портов и выводить полученные данные о состоянии данных портов в виде индикации на светодиодах, подключенных к порту С микроконтроллера. Эмулировать работу цифровых датчиков с помощью тумблера.

Б.4 В устройстве используется микроконтроллер Atmega328. Пины 11, 10 и 9 микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую считывать данные с указанных портов и выводить полученные данные о состоянии данных портов в виде индикации на светодиодах, подключенных к порту С микроконтроллера. Эмулировать работу цифровых датчиков с помощью тумблера.  
БАЗОВЫЙ

## Типовое решение задачи Б

Схема для реализации задачи представлена на рисунке. Электрическая схема состоит из платы разработчика в которой к каждому цифровому порту (0-7) подключен средний вывод ползункового переключателя, крайние выводы переключателя, подключены к потенциалам +5В и 0 соответственно, т.о. эмулируем работы цифровых датчиков. Изменяя положение переключателя можно изменять уровень напряжения на цифровых портах микроконтроллера. Результат битовых операций выводим на порт С микроконтроллера. Для этого к аналоговому выводу А0-А5 платы разработчика подключены светодиоды с резистором. При подаче логической единицы на пин порта С (высокого потенциала) засвечивается светодиод, подключенный к соответствующему выводу порта.

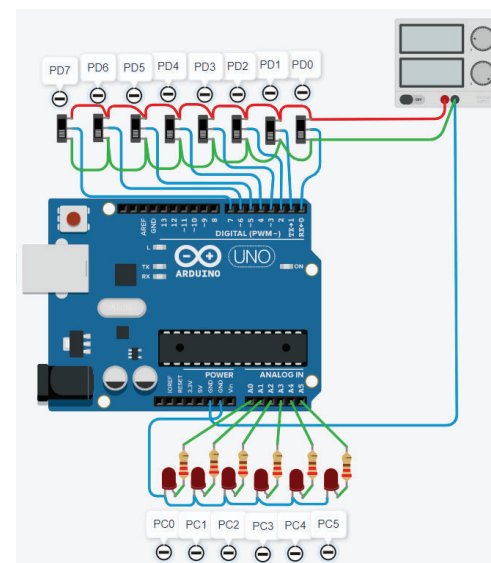


Схема для работы с портами микроконтроллера

1. void setup()
2. {
3. DDRD = 0b00000000; // установка порта D в режим ввода
4. PORTD = 0b11111111; // устанавливаем пины порта D в режим PullUp
5. DDRC = 0b1111111; // установка порта C в режим вывода
6. }
7. void loop()
8. {
9. PORTC = PIND & 0b100001; // наносим битовую маску на 0 и 5 бит порта D
10. // выводим полученный результат в PORTC

**СРЕДНИЙ**

С.1 В устройстве используется микроконтроллер Atmega328. Пины порта D микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую определять порт, в котором установлена лог.1 и вывести в виде двоичного кода номер порта на светодиодах, подключенных к порту С микроконтроллера, используя при этом язык программирования Arduino. Эмулировать работу цифровых датчиков с помощью тумблера.

С.2 В устройстве используется микроконтроллер Atmega328. Пины порта С микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую определять порт, в котором установлена лог.1 и вывести в виде двоичного кода номер порта на светодиодах, подключенных к порту D микроконтроллера, используя при этом язык программирования Arduino. Эмулировать работу цифровых датчиков с помощью тумблера.

С.3 В устройстве используется микроконтроллер Atmega328. Пины порта В микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую определять порт, в котором установлена лог.1 и вывести в виде двоичного кода номер порта на светодиодах, подключенных к порту С микроконтроллера, используя при этом язык программирования Arduino. Эмулировать работу цифровых датчиков с помощью тумблера.

С.4 В устройстве используется микроконтроллер Atmega328. Пины порта С микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую определять порт, в котором установлена лог.1 и вывести в виде двоичного кода номер порта на светодиодах, подключенных к порту В микроконтроллера, используя при этом язык программирования Arduino. Эмулировать работу цифровых датчиков с помощью тумблера.

**Типовое решение задачи С**

Схема для реализации задачи представлена на рисунке. Электрическая схема состоит из платы разработчика в которой к каждому цифровому порту (0-7) подключен средний вывод ползункового переключателя, крайние выводы переключателя, подключены к потенциалам +5В и 0 соответственно, т.о. эмулируем работы цифровых датчиков. Изменяя положение переключателя можно изменять уровень напряжения на цифровых портах микроконтроллера. Результат битовых операций выводим на порт С микроконтроллера. Для этого к аналоговым выводам А0-А5 платы разработчика подключены светодиоды с резистором. При подаче логической единицы на пин порта С (высокого потенциала) засвечивается светодиод, подключенный к соответствующему выводу порта.

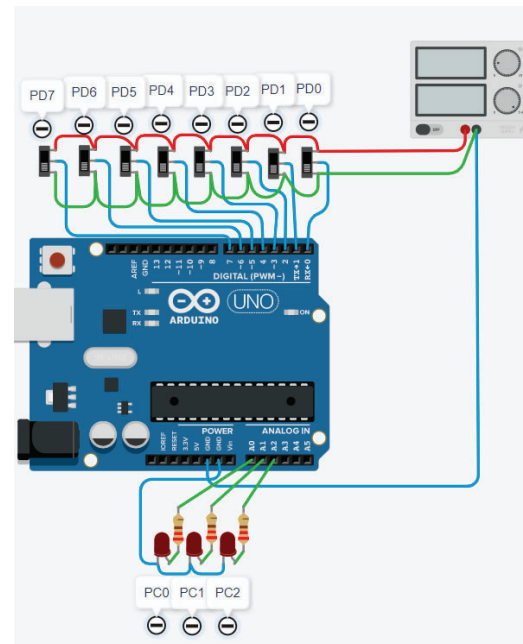


Схема для работы с портами микроконтроллера

```

1. void setup() {
2.   for (byte i = 0; i <= 7; i++) {
3.     pinMode(i, INPUT); // установка пинов порта D в режим ввода
4.   }
5.   for (byte j = 14; j <= 16; j++) {
6.     pinMode(j, OUTPUT); // установка пинов порта С в режим ввода
7.   }
8. }
9. void loop() {
10.  byte data = 0;
11.  for (byte i = 0; i <= 7; i++) {
12.
13.    if (digitalRead(i)) {
14.      data = i;
15.      Serial.println(data);
16.    }
17.  }
18.  switch (data) {
19.    case 0:
20.      digitalWrite(14, 0);
21.      digitalWrite(15, 0);
22.      digitalWrite(16, 0);
23.      break;
24.    case 1:
25.      digitalWrite(14, 1);
26.      digitalWrite(15, 0);
27.      digitalWrite(16, 0);
28.      break;
29.    case 2:

```

```

30. digitalWrite(14, 0);
31. digitalWrite(15, 1);
32. digitalWrite(16, 0);
33. break;
34. case 3:
35. digitalWrite(14, 1);
36. digitalWrite(15, 1);
37. digitalWrite(16, 0);
38. break;
39. case 4:
40. digitalWrite(14, 0);
41. digitalWrite(15, 0);
42. digitalWrite(16, 1);
43. break;
44. case 5:
45. digitalWrite(14, 1);
46. digitalWrite(15, 0);
47. digitalWrite(16, 1);
48. break;
49. case 6:
50. digitalWrite(14, 0);
51. digitalWrite(15, 1);
52. digitalWrite(16, 1);
53. break;
54. case 7:
55. digitalWrite(14, 1);
56. digitalWrite(15, 1);
57. digitalWrite(16, 1);
58. break;
59. default:
60. digitalWrite(14, 0);
61. digitalWrite(15, 0);
62. digitalWrite(16, 0);
63. }
64. }

```

## ВЫСОКИЙ

В.1 В устройстве используется микроконтроллер Atmega328. Пины порта D микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую определять порт, в котором установлена лог.1 и вывести в виде двоичного кода номер порта на светодиодах, подключенных к порту С микроконтроллера, используя при этом регистры управления портов общего назначения GPIO. Эмулировать работу цифровых датчиков с помощью тумблера.

В.2 В устройстве используется микроконтроллер Atmega328. Пины порта С микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую определять порт, в котором установлена лог.1 и вывести в виде двоичного кода номер порта на светодиодах, подключенных к порту D микроконтроллера. Эмулировать работу цифровых датчиков с помощью тумблера.

В.3 В устройстве используется микроконтроллер Atmega328. Пины порта В микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую определять порт, в котором установлена лог.1 и вывести в виде двоичного кода номер порта на светодиодах, подключенных к порту С микроконтроллера. Эмулировать работу цифровых датчиков с помощью тумблера.

В.4 В устройстве используется микроконтроллер Atmega328. Пины порта С микроконтроллера используются в качестве ввода информации с цифровых датчиков. Вам необходимо разработать программу, позволяющую определять порт, в котором установлена лог.1 и вывести в виде двоичного кода номер порта на светодиодах, подключенных к порту В микроконтроллера, используя при этом регистры управления портов общего назначения GPIO. Эмулировать работу цифровых датчиков с помощью тумблера

## Типовое решение задачи В

Схема для реализации задачи представлена на рисунке. Электрическая схема состоит из платы разработчика в которой к каждому цифровому порту (0-7) подключен средний вывод ползункового переключателя, крайние выводы переключателя, подключены к потенциалам +5В и 0 соответственно, т.о. эмулируем работы цифровых датчиков. Изменяя положение переключателя можно изменять уровень напряжения на цифровых портах микроконтроллера. Результат битовых операций выводим на порт С микроконтроллера. Для этого к аналоговому выводу А0-А5 платы разработчика подключены светодиоды с резистором. При подаче логической единицы на пин порта С (высокого потенциала) засвечивается светодиод, подключенный к соответствующему выводу порта.

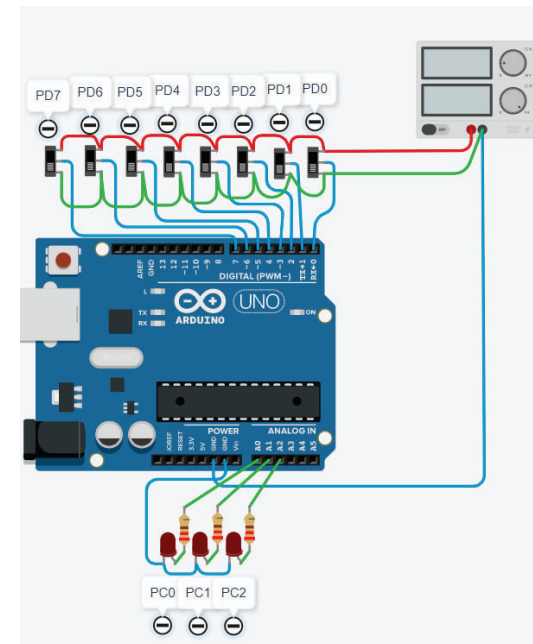


Схема для работы с портами микроконтроллера

```

1. int main(void)
2. {
3.   DDRD = 0b00000000; // установка порта D в режим ввода
4.   PORTD = 0b11111111; // устанавливаем пины порта D в режим PullUp
5.   DDRC = 0b1111111; // установка порта C в режим вывода
6.   while (true)

```

```

7. {
8.   switch (PIND & 0b11111111) {
9.     case 1:
10.      PORTC = 0b000000; //0
11.      break;
12.    case 2:
13.      PORTC = 0b000001; //1
14.      break;
15.    case 4:
16.      PORTC = 0b000010; //2
17.      break;
18.    case 8:
19.      PORTC = 0b000011; //3
20.      break;
21.    case 16:
22.      PORTC = 0b000100; //4
23.      break;
24.    case 32:
25.      PORTC = 0b000101; //5
26.      break;
27.    case 64:
28.      PORTC = 0b000110; //6
29.      break;
30.    case 128:
31.      PORTC = 0b000111; //7
32.      break;
33.    default:
34.      PORTC = 0b000000; // если лог.0 на пине
35.  }
36. }
37. }

```

## Раздел 3. Периферийные модули микроконтроллера

### Ввод информации в микроконтроллер (тактовая кнопка)

#### 12 задач

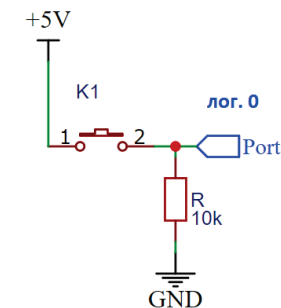
##### БАЗОВЫЙ

Б.1 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 6 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата, то необходимо засветить светодиод, подключенный к 23 пину микроконтроллера Atmega328. Код написать языком программирования Arduino.

Б.2 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 3 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата, то необходимо засветить светодиод, подключенный к 26 пину микроконтроллера Atmega328. Код написать языком программирования Arduino.

Б.3 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 4 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата, то необходимо засветить светодиод, подключенный к 24 пину микроконтроллера Atmega328. Код написать языком программирования Arduino.

Б.4 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 5 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата, то необходимо засветить светодиод, подключенный к 25 пину микроконтроллера Atmega328. Код написать языком программирования Arduino.

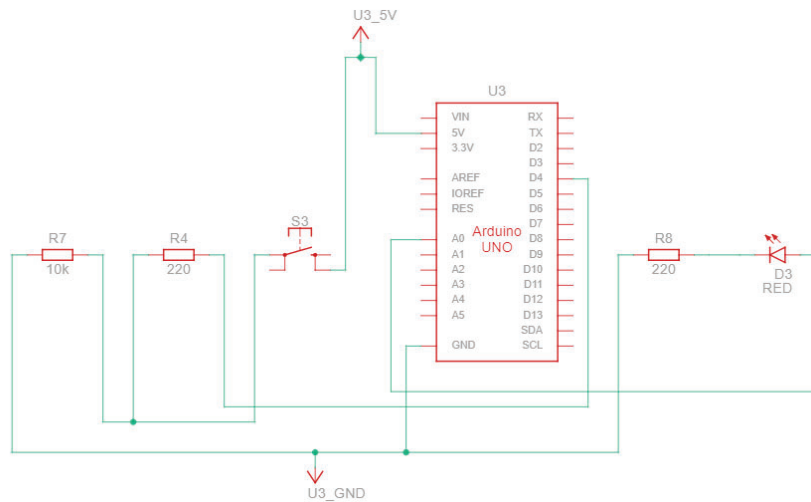


Подключение тактовой кнопки

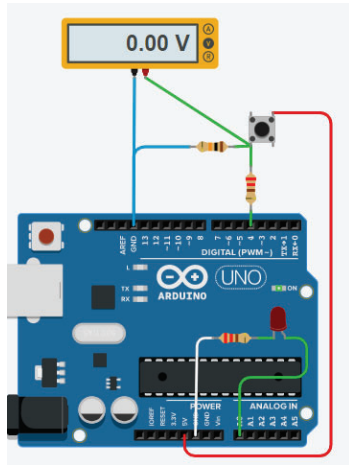
#### Типовое решение задачи Б

Рассмотрим схему, представленную на рисунке. Когда контакт S3 не замкнут, вход D4 (соответствует 6 порту микроконтроллера) подключен через резистор R7 к низкому потенциалу GND и на выводе D4 находится логический 0. Если замкнуть контакт S3, то ток будет протекать через него и попадать на вход D4. Таким образом на входе D4 появится напряжение, соответствующее уровню логической 1. Светодиод VD3 подключен к A0 который соответствует 23 пину микроконтроллера Atmega328. Принципиальная схема представлена на рисунке.





Электрическая схема подключения



Принципиальная схема подключения

#### Листинг программы:

```

1. void setup() {
2.   pinMode(4, INPUT); // установка 4 пин порта D в режим ввода
3.   pinMode(14, OUTPUT); // установка 0 порта C в режим вывода
4. }
5. void loop() {
6.   if (digitalRead(4)) { // считываем состояние 4 пина порта D
7.     digitalWrite(14, 1); // выставляем лог. 1 (светодиод светится) на 14 выход (A0)
8.   } else
9.     digitalWrite(14, 0); // выставляем лог. 0 (светодиод не светится) на 14 выход (A0)
10. }

```

## СРЕДНИЙ

C.1 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 6 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата, то необходимо засветить светодиод, подключенный к 23 пину микроконтроллера Atmega328. Код написать, используя регистры управления портов общего назначения GPIO.

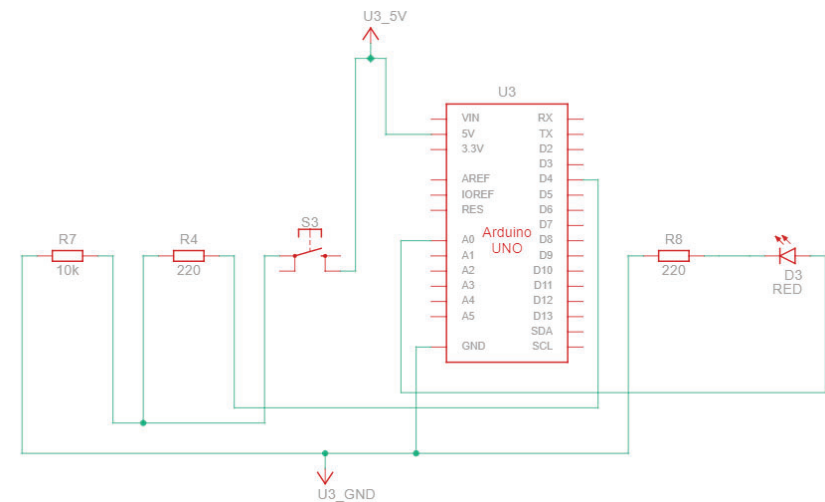
C.2 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 7 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата, то необходимо засветить светодиод, подключенный к 24 пину микроконтроллера Atmega328. Код написать, используя регистры управления портов общего назначения GPIO.

C.3 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 3 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата, то необходимо засветить светодиод, подключенный к 25 пину микроконтроллера Atmega328. Код написать, используя регистры управления портов общего назначения GPIO.

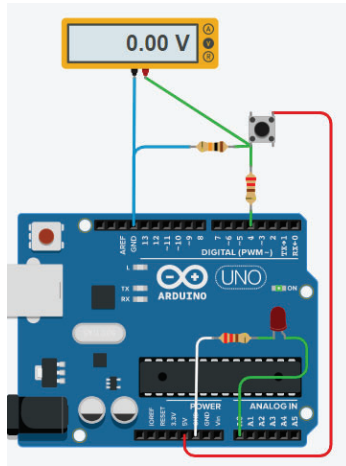
C.4 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 5 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата, то необходимо засветить светодиод, подключенный к 26 пину микроконтроллера Atmega328. Код написать, используя регистры управления портов общего назначения GPIO.

## Типовое решение задачи C

Рассмотрим схему, представленную на рисунке. Когда контакт S3 не замкнут, вход D4 (соответствует 6 порту микроконтроллера) подключен через резистор R7 к низкому потенциалу GND и на выводе D4 находится логический 0. Если замкнуть контакт S3, то ток будет протекать через него и попадать на вход D4. Таким образом на входе D4 появится напряжение, соответствующее уровню логической 1. Светодиод VD3 подключен к A0 который соответствует 23 пину микроконтроллера Atmega328. Принципиальная схема представлена на рисунке.



Электрическая схема подключения



Принципиальная схема подключения

#### Листинг программы

```

1. int main(void) {
2.   DDRD = ~(1 << 4); // установка 4 пин порта D в режим ввода
3.   DDRC |= (1 << 0); // установка 0 порта C в режим вывода
4.
5.   while (true) {
6.     if (PIND & (1 << 4)) {
7.       PORTC |= (1 << 0);
8.     } else {
9.       PORTC &= !(1 << 0);
10.    }
11.  }
12. }

```

#### ВЫСОКИЙ

V.1 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 6 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата 4 раза, то необходимо засветить светодиод, подключенный к 23 пину микроконтроллера Atmega328, при двукратном нажатии на тактовую кнопку светодиод прекращает светиться.

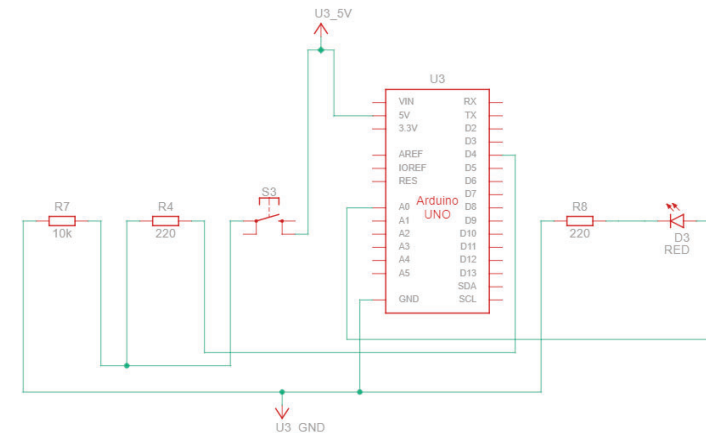
V.2 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 7 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата 6 раз, то необходимо засветить светодиод, подключенный к 24 пину микроконтроллера Atmega328, при двукратном нажатии на тактовую кнопку светодиод прекращает светиться.

V.3 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 3 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата 3 раза, то необходимо засветить светодиод, подключенный к 23 пину микроконтроллера Atmega328, при четырехкратном нажатии на тактовую кнопку светодиод прекращает светиться.

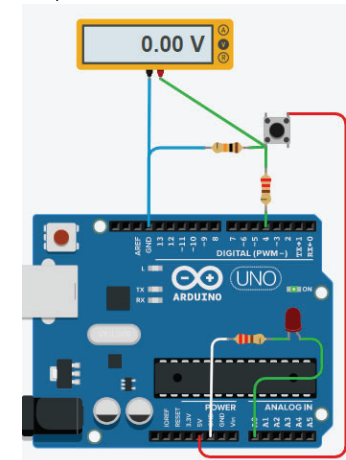
V.4 Ввод информации осуществляется с помощью тактовой кнопки подключенной к 5 порту микроконтроллера согласно электрической схеме, представленной на рисунке. Необходимо определить состояние тактовой кнопки (нажата или отжата). Если кнопка нажата 5 раз, то необходимо засветить светодиод, подключенный к 25 пину микроконтроллера Atmega328, при трехкратном нажатии на тактовую кнопку светодиод прекращает светиться.

#### Типовое решение задачи В

Рассмотрим схему, представленную на рисунке. Когда контакт S3 не замкнут, вход D4 (соответствует 6 порту микроконтроллера) подключен через резистор R7 к низкому потенциалу GND и на выводе D4 находится логический 0. Если замкнуть контакт S3, то ток будет протекать через него и попадать на вход D4. Таким образом на входе D4 появится напряжение, соответствующее уровню логической 1. Светодиод VD3 подключен к A0 который соответствует 23 пину микроконтроллера Atmega328. Принципиальная схема представлена на рисунке.



Электрическая схема подключения



Принципиальная схема подключения

### Листинг программы:

```
1. byte but = 0;
2. void setup() {
3.   pinMode(4, INPUT); // установка 4 пин порта D в режим ввода
4.   pinMode(14, OUTPUT); // установка 0 порта C в режим вывода
5. }
6. void loop() {
7.   if (digitalRead(4)) {
8.     but++;
9.     delay(500);
10.    if (but == 4) {
11.      digitalWrite(14, 1);
12.    }
13.    if (but == 6) {
14.      digitalWrite(14, 0);
15.      but = 0;
16.    }
17.  }
18. }
```

## Таймер-работа со временем. Сторожевой таймер

### 12 задач

#### БАЗОВЫЙ

- Б.1 Используя функцию millis() реализовать мигание светодиода на 7 порту платы разработчика с периодом мигание 500 мс.
- Б.2 Используя функцию millis() реализовать мигание светодиода на 9 порту платы разработчика с периодом мигание 1000 мс.
- Б.3 Используя функцию millis() реализовать мигание светодиода на 10 порту платы разработчика с периодом мигание 1500 мс.
- Б.4 Используя функцию millis() реализовать мигание светодиода на 11 порт платы разработчика с периодом мигание 250 мс.

#### Типовое решение задачи Б

В первой строке программы мы объявляем и обнуляем переменную timer типа uint32\_t - это без знаковый int, который занимает 32 бита, для хранения данных возвращаемой функцией millis(). Во второй строке формируем с помощью функции #define автозамену слова T\_PERIOD на число 500 при компиляции программы. Данное число будет отвечать за длительность импульса. В 5 строке DDRD |= (1<<7) устанавливаем режим вывода на пин порта D на выход. В бесконечном цикле loop() проверяем условие if ((millis()-timer)>= T\_PERIOD), т.е. если разница между текущим временем и зафиксированное временем будет больше T\_PERIOD значит произошло событие, т.е. скобка стала равна true, тогда попадаем в тело условия if. В строке 11 сбрасываем зафиксированное время timer = millis() это делается для того так как время протекает линейно и для следующей обработке необходимо чтобы время time было увеличено в момент срабатывания условия. В 12 строке считываем состояние порта D PIND&(1<<7) если порт будет равен 1, то меняем его на противоположной в 14 строке PORTD &= ~(1<<7), если же состояние порта равен 0, то в 18 строке меняем его на противоположный PORTD |= (1<<7) выставляем лог.1. Далее операции повторяются заново.

Визуально будем наблюдать мигание светодиода подключенного к 7 пину порта D. Данная программ привлекательней для работы микроконтроллера тем что не использует задержки типа delay() при выполнении кода, таким образом микроконтроллер может выполнять параллельно другие операции.

```
1. uint32_t timer =0;
2. #define T_PERIOD 500 //период переключения
3. void setup()
4. {
5.   DDRD |= (1<<7); // установили 7 пин порта D на выход
6. }
7. void loop ()
8. {
9.   if ((millis()-timer)>= T_PERIOD)
10.  {
11.    timer = millis(); //сброс
12.    if ( PIND&(1<<7))
13.    {
14.      PORTD &= ~(1<<7);
15.    }
16.    else
17.    {
18.      PORTD |= (1<<7);
19.    }
20.  }
21. }
```

#### СРЕДНИЙ

- С1. Используя функцию millis() реализовать мигание двух светодиодов с различной частотой (1 светодиод с частотой 100 мс, 2 светодиод с частотой 500 мс), светодиод 1 подключен к 7 порту платы разработчика, а 2 светодиод - к 4 порту.
- С2. Используя функцию millis() реализовать мигание двух светодиодов с различной частотой (1 светодиод с частотой 250 мс, 2 светодиод с частотой 1000 мс), светодиод 1 подключен к 13 порту платы разработчика, а 2 светодиод - к 8 порту.
- С3. Используя функцию millis() реализовать мигание двух светодиодов с различной частотой (1 светодиод с частотой 300 мс, 2 светодиод с частотой 1500 мс), светодиод 1 подключен к 5 порту платы разработчика, а 2 светодиод - к 3 порту.
- С4. Используя функцию millis() реализовать мигание двух светодиодов с различной частотой (1 светодиод с частотой 200 мс, 2 светодиод с частотой 700 мс), светодиод 1 подключен к 7 порту платы разработчика, а 2 светодиод - к 8 порту.

#### Типовое решение задачи С

Чтобы реализовать алгоритм нам потребуется две переменные для хранения времени timer1 и timer2. Периоды мигания запоминаем с помощью функции автозамены define T\_PERIOD1 и T\_PERIOD2 соответственно 100 и 500 мс. В 6 строчке программы устанавливаем режим работы пинов порта 4 и 7 на выход. Для удобства понимания основного кода программы мигание отдельных светодиодов реализовали с помощью подпрограмм: mig\_led1() и mig\_led2(). Подпрограммы выполняют алгоритм описанный для мигания единичного светодиода с той лишь разнице, что время и период мигания настраивается для каждого светодиода индивидуально.

```
1. uint32_t timer1 = 0;
2. uint32_t timer2 = 0;
3. #define T_PERIOD1 100 //период переключения
4. #define T_PERIOD2 500 //период переключения
5. void setup() {
6.   DDRD |= (1 << 7) | (1 << 4); // установили 7,4 пин порта D на выход
7. }
```

```

8. void loop() {
9.  //*****мигаем 7 портом *****
10. mig_led1();
11. //*****
12. //*****мигаем 4 портом *****
13. mig_led2();
14. //*****
15. }
16. void mig_led1() {
17. if ((millis() - timer1) >= T_PERIOD1) {
18.   timer1 = millis(); //сброс
19.   if (PIND & (1 << 7)) {
20.     PORTD &= ~(1 << 7);
21.   } else {
22.     PORTD |= (1 << 7);
23.   }
24. }
25. }
26. void mig_led2() {
27. if ((millis() - timer2) >= T_PERIOD2) {
28.   timer2 = millis(); //сброс
29.   if (PIND & (1 << 4)) {
30.     PORTD &= ~(1 << 4);
31.   } else {
32.     PORTD |= (1 << 4);
33.   }
34. }
35. }

```

## ВЫСОКИЙ

V1. Используя функцию millis() реализовать мигание двух светодиодов с различной частотой (1 светодиод с частотой 100 мс, 2 светодиода с частотой 500 мс), светодиод 1 подключен к 7 порту платы разработчика, а 2 светодиода - к 4 порту. Также необходимо обеспечить опрос тактовой кнопки подключенной к 3 порту платы разработчика при нажатии на которую светодиод подключенный к порту A0 будет светиться при отпускании кнопки светодиода, подключенный к порту A0 перестает светиться.

V2. Используя функцию millis() реализовать мигание двух светодиодов с различной частотой (1 светодиод с частотой 250 мс, 2 светодиода с частотой 1000 мс), светодиод 1 подключен к 5 порту платы разработчика, а 2 светодиода - к 9 порту. Также необходимо обеспечить опрос тактовой кнопки подключенной к 3 порту платы разработчика при нажатии на которую светодиод подключенный к порту A1 будет светиться при отпускании кнопки светодиода, подключенный к порту A1 перестает светиться.

V3. Используя функцию millis() реализовать мигание двух светодиодов с различной частотой (1 светодиод с частотой 300 мс, 2 светодиода с частотой 1500 мс), светодиод 1 подключен к 13 порту платы разработчика, а 2 светодиода - к 7 порту. Также необходимо обеспечить опрос тактовой кнопки подключенной к 3 порту платы разработчика при нажатии на которую светодиод подключенный к порту A2 будет светиться при отпускании кнопки светодиода, подключенный к порту A2 перестает светиться.

V4. Используя функцию millis() реализовать мигание двух светодиодов с различной частотой (1 светодиод с частотой 300 мс, 2 светодиода с частотой 800 мс), светодиод 1 подключен к 8 порту платы разработчика, а 2 светодиода - к 4 порту. Также необходимо обеспечить опрос тактовой кнопки подключенной к 5 порту платы разработчика при нажатии на которую светодиод подключенный к порту A5 будет светиться при отпускании кнопки светодиода, подключенный к порту A5 перестает светиться.

## Типовое решение задачи В

В основную программу листинг который мы рассмотрели выше добавили функцию опроса кнопки button(), которая реализована в 21 по 29 строке программного кода. В функцию button() передаем номер порта, с которого снимаем информацию о кнопке. Далее вызываем функцию button() и переходим в 21 строку. Номер пина порта переданный в функцию button() храниться в локальной переменной X типа byte. В 22 строке if ((PIND & (1 << x))) считываем состояние порта D и проверяем есть ли лог. 1 на указанном порте X. Если лог. 1 установлена, то включаем светодиод в строке 24 PORTC |= (1 << 0), если нет, то в строке 27 отключаем светодиод PORTC &= ~(1 << 0).

Таким образом основой код программы будет состоять из реализации трех подпрограмм: мигание 1 светодиодом строка 12, мигание 2 светодиодом строка 15 и опрос кнопки строка 18. Использование подпрограмм позволяет уменьшить тело основного кода, сделать его удобно читаемым и понятным.

Алгоритм реализован в листинге программы представленном ниже:

```

1. uint32_t timer1 = 0;
2. uint32_t timer2 = 0;
3. #define T_PERIOD1 100 //период переключения
4. #define T_PERIOD2 500 //период переключения
5. void setup() {
6.   DDRD |= (1 << 7) | (1 << 4); // установили 7,4 пин порта D на выход
7.   DDRD &= ~(1 << 3); // 2 пин порта установили на ввод информации
8.   DDRC |= (1 << 0); // 0 пин порта C установили на вывод
9. }
10. void loop() {
11. //*****мигаем 7 портом *****
12. mig_led1();
13. //*****
14. //*****мигаем 4 портом *****
15. mig_led2();
16. //*****
17. //*****Опрос кнопки *****
18. button(3);
19. //*****
20. }
21. void button(byte x) {
22. if ((PIND & (1 << x))) {
23.   // включаем светодиод
24.   PORTC |= (1 << 0);
25. } else {
26.   // выключаем светодиод
27.   PORTC &= ~(1 << 0);
28. }
29. }
30. void mig_led1() {
31. if ((millis() - timer1) >= T_PERIOD1) {
32.   timer1 = millis(); //сброс
33.   if (PIND & (1 << 7)) {
34.     PORTD &= ~(1 << 7);
35.   } else {
36.     PORTD |= (1 << 7);

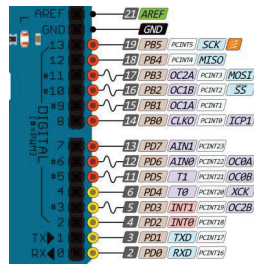
```

```

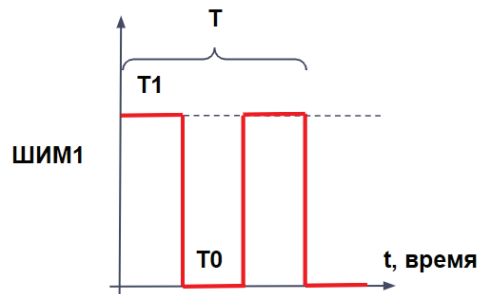
37. }
38. }
39. }
40. void mig_led2() {
41.   if ((millis() - timer2) >= T_PERIOD2) {
42.     timer2 = millis(); //сброс
43.     if (PIND & (1 << 4)) {
44.       PORTD &= ~(1 << 4);
45.     } else {
46.       PORTD |= (1 << 4);
47.     }
48.   }
49. }

```

## Широтно-импульсная модуляция 12 задач



Назначение портов платы разработчика



Широтно-импульсный сигнал

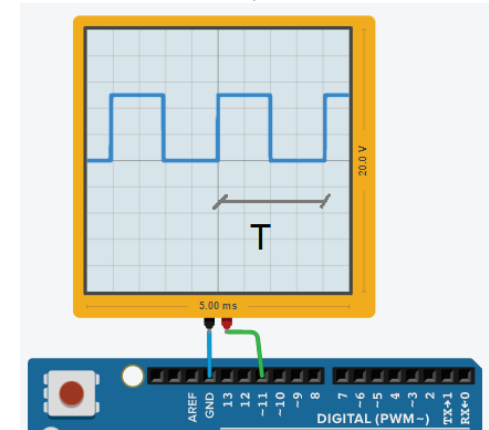
### БАЗОВЫЙ

- Б1. Используя функцию analogWrite() установите ШИМ сигнал на 11 порту платы разработчика, частота которого будет соответствовать 500 Гц, а коэффициент заполнения сигнала - 50%.
- Б2. Используя функцию analogWrite() установите ШИМ сигнал на 6 порту платы разработчика, частота которого будет соответствовать 500 Гц, а коэффициент заполнения сигнала - 25%.
- Б3. Используя функцию analogWrite() установите ШИМ сигнал на 5 порту платы разработчика, частота которого будет соответствовать 500 Гц, а коэффициент заполнения сигнала - 75%.
- Б4. Используя функцию analogWrite() установите ШИМ сигнал на 9 порту платы разработчика, частота которого будет соответствовать 500 Гц, а коэффициент заполнения сигнала - 80%

## Типовое решение задачи Б

1. void setup()
2. { analogWrite(11, 125); }
3. void loop() { }

Результат работы программы представлен на рисунке



Результат эксперимента

На осциллограмме, представленной на рисунке частота сигнала:

$$T = T_0 + T_1$$

$$f = 1/T$$

$$f = 1/T = 1/(4 * 0.5 * 10^{-3}) = 500 \text{ Гц}$$

$$\text{Коэффициент заполнения} =$$

$$D = T_1/T = 2 * 0.5 * 10^{-3} / 4 * 0.5 * 10^{-3} = 0,5$$

### СРЕДНИЙ

- С1. Используя настройки таймера микроконтроллера Atmega328 установите на 3 порту платы разработчика частоту сигнала, соответствующую 122 Гц.
- С2. Используя настройки таймера микроконтроллера Atmega328 установите на 3 порту платы разработчика частоту сигнала, соответствующую 980 Гц.
- С3. Используя настройки таймера микроконтроллера Atmega328 установите на 3 порту платы разработчика частоту сигнала, соответствующую 30 Гц.
- С4. Используя настройки таймера микроконтроллера Atmega328 установите на 3 порту платы разработчика частоту сигнала, соответствующую 31.25 кГц.

## Типовое решение задачи С

Настройка таймера осуществляется регистрами микроконтроллера TCCR2A и TCCR2B. Для этого с помощью битов CS22, CS21, CS20 устанавливаем число 110 в регистр TCCR2B (листинг программы строки 12-13). Режим работы таймера устанавливаем с помощью битов WGM21, WGM20 регистра TCCR2A устанавливаем работу таймера как ШИМ с коррекцией фазы, т.е. записав число 11 в соответствующие биты (листинг программы строки 15-16). Активацию сигнала осуществляем в 17 строке командой analogWrite(11, 125), где 11 - пин порта платы разработчика, 125 - ширина импульса ШИМ сигнала (может быть от 0 - 125). Частота сигнала определяется из выражения:

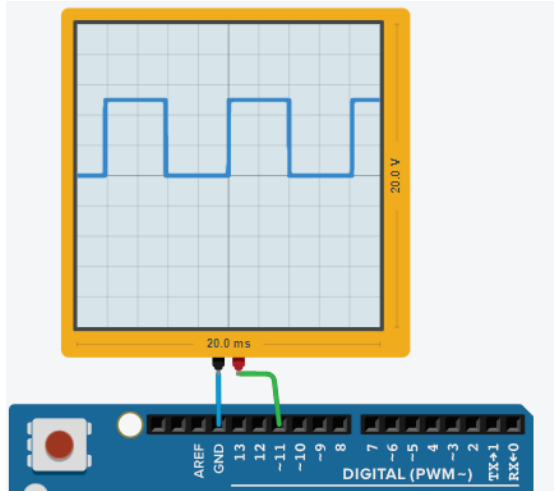
$f = f_{clk\_I/O}/512N$ ,

где  $N$  — коэффициент деления предделителя (расчетна)

```
1. void setup() {
2. // Устанавливаем делитель частоты
3. // 001 делитель 1
4. // 010 делитель 8
5. // 011 делитель 32
6. // 100 делитель 64
7. // 101 делитель 128
8. // 110 делитель 256
9. // 111 делитель 1024
10. // Устанавливаем делитель частоты 110 - что соответствует 256
11. TCCR2B |= (1 << CS22);
12. TCCR2B |= (1 << CS21);
13. TCCR2B &= ~(1 << CS20);
14. // устанавливаем режим работы таймера на 11 выходе как ШИМ
15. TCCR2A &= ~(1 << WGM21);
16. TCCR2A |= (1 << WGM20);
17. analogWrite(11, 125);
18. }
```

```
void loop() { }
```

Результат работы кода представлена на рисунке, где частота сигнала равна 122Гц, т.о можно изменять величину делителя и получить сигналы различной частоты.



Результат эксперимента  $f$  сигнала = 122 Гц (делитель 256)

## ВЫСОКИЙ

V1. При программировании микроконтроллера никто не застрахован от появления таких условий при которых выполнение кода будет невозможно. В результате чего программа закидывается в цикл и микроконтроллер не может самостоятельно выйти из “западни”. С елью безотказной работы и для контроля в микроконтроллере установлен сторожевой таймер. Необходимо

настроить сторожевой таймер так чтобы через каждый 15 мс микроконтроллер сбрасывал сторожевой таймер.

V2. При программировании микроконтроллера никто не застрахован от появления таких условий при которых выполнение кода будет невозможно. В результате чего программа закидывается в цикл и микроконтроллер не может самостоятельно выйти из “западни”. С елью безотказной работы и для контроля в микроконтроллере установлен сторожевой таймер. Необходимо настроить сторожевой таймер так чтобы через каждый 120 мс микроконтроллер сбрасывал сторожевой таймер.

V3. При программировании микроконтроллера никто не застрахован от появления таких условий при которых выполнение кода будет невозможно. В результате чего программа закидывается в цикл и микроконтроллер не может самостоятельно выйти из “западни”. С елью безотказной работы и для контроля в микроконтроллере установлен сторожевой таймер. Необходимо настроить сторожевой таймер так чтобы через каждый 250 мс микроконтроллер сбрасывал сторожевой таймер.

V3. При программировании микроконтроллера никто не застрахован от появления таких условий при которых выполнение кода будет невозможно. В результате чего программа закидывается в цикл и микроконтроллер не может самостоятельно выйти из “западни”. С елью безотказной работы и для контроля в микроконтроллере установлен сторожевой таймер. Необходимо настроить сторожевой таймер так чтобы через каждый 60 мс микроконтроллер сбрасывал сторожевой таймер.

## Типовое решение задачи В

В первой строке подключаем библиотеку `avr/wdt.h` для работы со сторожевым таймером. В функции `setup` настраиваем режим 13 пина на работу в режиме “выход” (4 строка программы). Далее в 5 и 8 строке осуществляет однократное мигание светодиода с периодом в 1 секунду. После этого программа заходит в основной цикл программы `loop`, где в 11 строке разрешает работу сторожевого таймера с частотой обновления 15 мс. Далее программа попадает в бесконечный цикл, где функция сброса сторожевого таймера закомментирована. Т.е. попав во внутренний бесконечный цикл программа закидывается и если бы не активировали сторожевой таймер, то мигание светодиода мы бы не наблюдали. Однако при работе кода будем видеть, что каждые 15 мс у нас будет перезагружаться микроконтроллер и мы будем видеть постоянно мигающий светодиод, подключенный к 13 пину порта, так как код программы будет начинаться с начала раз за разом. Если раскомментировать в 15 строке функцию сброса сторожевого таймера в бесконечном цикле, то произойдет сброс таймера и программа перезагружаться не будет, тем самым мигание светодиода на 13 пине мы не увидим. Таким образом сторожевой таймер позволяет нам обеспечивать стабильную работу программы.

```
1. #include <avr/wdt.h> // подключение библиотеки для работы со сторожевым таймером
2. void setup()
3. {
4.   pinMode(13, OUTPUT); // назначить тип пина
5.   digitalWrite(13, HIGH); // включить светодиод
6.   delay(1000); // ждем 1 с
7.   digitalWrite(13, LOW); // выключить светодиод
8.   delay(1000); // ждем 1 с
9. }
10. void loop(){
11.   wdt_enable(WDTO_15MS); // разрешить watchdog
12.   // функция запустится через 15 мс, если не сбросить таймер
13.   while(1)
14.   {
15.     // wdt_reset(); // раскомментируйте для избежания перезагрузок
16.   }
```

## Раздел 4. Протоколы передачи, память и прерывание микроконтроллера

### Протокол передачи данных USART.

#### 12 задач

##### БАЗОВЫЙ

Б1. Используя библиотеку Serial осуществите следующий алгоритм: при запуске программы по последовательному протоколу передачи данных отправим слово "Hello!". При нажатии на кнопку подключенному к 8 пину платы разработчика выводим слово "Button1", при нажатии на кнопку подключенному к пину 9 - слово "Button2".

Б2. Используя библиотеку Serial осуществите следующий алгоритм: при запуске программы по последовательному протоколу передачи данных отправим слово "Hello, word!". При нажатии на кнопку подключенному к 10 пину платы разработчика выводим слово "Step1", при нажатии на кнопку подключенному к пину 11 - слово "Step2".

Б3. Используя библиотеку Serial осуществите следующий алгоритм: при запуске программы по последовательному протоколу передачи данных отправим слово "Hi!". При нажатии на кнопку подключенному к 3 пину платы разработчика выводим слово "B1", при нажатии на кнопку подключенному к пину 4 - слово "B2".

Б4. Используя библиотеку Serial осуществите следующий алгоритм: при запуске программы по последовательному протоколу передачи данных отправим слово "Ok!". При нажатии на кнопку подключенному к 5 пину платы разработчика выводим слово "But1", при нажатии на кнопку подключенному к пину 4 - слово "But2".

##### Типовое решение задачи Б

```

1. void setup ()
2. {
3.   Serial.begin(9600);
4.   Serial.println("Hello!");
5. }
6. void loop ()
7. {
8.   if(PINB&(1<<0))// если кнопка нажата
9.   {
10.    Serial.println("Button1");
11.    delay(500);
12.   }
13.   if(PINB&(1<<1))// если кнопка нажата
14.   {
15.    Serial.println("Button2");
16.    delay(500);
17.   }
18. }
```

##### СРЕДНИЙ

С1. Используя регистры управления модуля передачи USART микроконтроллера Atmega328 осуществите следующий алгоритм: при запуске программы по последовательному протоколу передачи данных отправим слово "Hi!". При нажатии на кнопку подключенному к 8 пину платы разработчика выводим слово "Bt1", при нажатии на кнопку подключенному к пину 9 - слово "Bt2".

С2. Используя регистры управления модуля передачи USART микроконтроллера Atmega328 осуществите следующий алгоритм: при запуске программы по последовательному протоколу передачи данных отправим слово "Hello!". При нажатии на кнопку подключенному к 7 пину платы разработчика выводим слово "Button1", при нажатии на кнопку подключенному к пину 6 - слово "Button2".

С3. Используя регистры управления модуля передачи USART микроконтроллера Atmega328 осуществите следующий алгоритм: при запуске программы по последовательному протоколу передачи данных отправим слово "Go". При нажатии на кнопку подключенному к 5 пину платы разработчика выводим слово "step1", при нажатии на кнопку подключенному к пину 4 - слово "step2".

С4. Используя регистры управления модуля передачи USART микроконтроллера Atmega328 осуществите следующий алгоритм: при запуске программы по последовательному протоколу передачи данных отправим слово "Run!". При нажатии на кнопку подключенному к 8 пину платы разработчика выводим слово "But1", при нажатии на кнопку подключенному к пину 9 - слово "But2".

##### Типовое решение задачи

В 1 строке подключаем к исполняемому коду файл avr/io.h. В этом файле находятся определения констант, имен регистров и всего необходимого, что может понадобиться для выполнения кода программы. Во 2 строке util/delay.h подключаем файл, позволяющий использовать задержки в коде программы. В 3 строке устанавливаем рабочую частота контроллера. В 4 строке задаем скорость обмена данными. В 5 строке рассчитываем по формуле  $UBRR0 = CLK/(16 \cdot \text{Скорость}) - 1$  согласно заданной скорости подсчитываем значение для регистра UBRR.

С 6 по 12 строки формируем подпрограмму init\_USART() производим настройку протокола USART :

7 и 8 строка заносим рассчитанную частоту передачи данных в регистры UBRR0L и UBRR0H  
9 строке устанавливаем бит разрешения передачи TXEN0 в регистре UCSR0B  
10 и 11 строка битами UCSZ02, UCSZ01 и UCSZ00 в соответствующих регистрах UCSR0B и UCSR0C устанавливаем 8 бит формат данных

В 13 -16 строках формируем подпрограмму send\_UART позволяющая отправить символы по протоколу данных USART:

в 14 строке с помощью цикла while ожидаем, когда очистится буфер передачи при этом произойдет сброс бита UDRE0 регистра UCSR0A

в 15 строке в регистр UDR0 передаем отправляемый символ value

В 48 - 52 строке формируем подпрограмму init\_pin настройки пинов порта B, к которому подключены тактовые кнопки это пины 0 и1 регистра B, что соответствует 8 и 9 вывода платы разработчика. Данные пины настраиваются на ввод данных.

С 17 по 47 строку выполняется основной код:

в 19 строке вызывается подпрограмма настройки пинов init\_pin

в 20 строке вызывается программа инициализации протокола UART

в 21 строке отправляется символ "O" с вызовом подпрограммы send\_UART

в 22 строке отправляется символ "K" с вызовом подпрограммы send\_UART

в 23 строке отправляется символ "!" с вызовом подпрограммы send\_UART

в 24 строке отправляется номер символ в шеснадцатеричном формате 0x0D переход в начало строки с вызовом подпрограммы send\_UART

в 25 строке отправляется номер символ в шеснадцатеричном формате 0x0A переход на новую строку с вызовом подпрограммы send\_UART

с 26 по 46 строку попадаем в бесконечный цикл while где производим опрос тактовых кнопок, подключенных к 0 и 1 пина порта B.

В 28 строке проверяем если кнопка подключенная к пину 0 порта В нажата PINB&(1<<0), то с вызовом подпрограммы send\_UART отправляем посимвольно слово PB0. В 35 строке делаем задержку на 500 мс.

С 37 по 45 производим опрос 1 пина порта В и с вызовом подпрограммы send\_UART отправляем посимвольно слово PB1.

```
1. #include <avr/io.h>
2. #include <util/delay.h>
3. #define F_CPU 16000000 // Рабочая частота контроллера
4. #define BAUD 9600L // Скорость обмена данными
5. #define UBRRL_value (F_CPU/(BAUD*16))-1 //Согласно заданной скорости, подсчитываем значение для регистра UBRR
6. void init_USART() {
7.   UBRR0L = UBRRL_value; //Младшие 8 бит UBRRL_value
8.   UBRR0H = UBRRL_value >> 8; //Старшие 8 бит UBRRL_value
9.   UCSR0B |= (1<<TXEN0); //Бит разрешения передачи
10.  UCSR0B |= (1<<UCSZ02); //Устанавливаем формат 8 бит данных
11.  UCSR0C |= (1<< UCSZ00)|(1<< UCSZ01);
12. }
13. void send_UART(char value) {
14.  while(!( UCSR0A & (1 << UDRE0))); // Ожидаем когда очистится буфер передачи
15.  UDR0 = value; // Помещаем данные в буфер, начинаем передачу
16. }
17. int main(void)
18. {
19.  init_pin();
20.  init_USART(); //инициализация USART в режиме 9600/8-N-1
21.  send_UART(0x48); //посылаем ASCII код знака 'H'
22.  send_UART(0x69); //посылаем ASCII код знака 'i'
23.  send_UART(0x21); //посылаем ASCII код знака '!'
24.  send_UART(0x0D); //переход в начало строки
25.  send_UART(0x0A); //переход на новую строку
26.  while(1)
27.  {
28.    if(PINB&(1<<0))// если кнопка нажата
29.    {
30.      send_UART(0x42); // символ ASCII буквы 'B'
31.      send_UART(0x74); //символ ASCII буквы 't'
32.      send_UART(0x31); //символ ASCII цифры '1'
33.      send_UART(0x0D);
34.      send_UART(0x0A);
35.      _delay_ms(500);
36.    }
37.    if(PINB&(1<<1))// если кнопка нажата
38.    {
39.      send_UART('B');
40.      send_UART('t');
41.      send_UART('2');
42.      send_UART(0x0D);
43.      send_UART(0x0A);
44.      _delay_ms(500);
45.    }
46.  }
47. }
48. void init_pin(void)
```

```
49. {
50.   DDRB= 0b00000000;//PB0, PB1 input
50. }
```

## ВЫСОКИЙ

В1. С помощью протокола UART микроконтроллеру Atmega328 передается пакет, данных вид, которого представлен ниже

\$100, 125, 255;

Числа в пакете — это задание для ШИМ сигналов на 5, 9, 10 выходах платы разработчика соответственно. Составьте программу для микроконтроллера позволяющую принимать данные и устанавливать нужные ШИМ сигналы на указанных выводах микроконтроллера.

В2. С помощью протокола UART микроконтроллеру Atmega328 передается пакет, данных вид, которого представлен ниже

№50, 100, 125;

Числа в пакете — это задание для ШИМ сигналов на 3, 5, 6 выходах платы разработчика соответственно. Составьте программу для микроконтроллера позволяющую принимать данные и устанавливать нужные ШИМ сигналы на указанных выводах микроконтроллера.

В3. С помощью протокола UART микроконтроллеру Atmega328 передается пакет, данных вид, которого представлен ниже

#75, 100, 255;

Числа в пакете — это задание для ШИМ сигналов на 3, 10, 11 выходах платы разработчика соответственно. Составьте программу для микроконтроллера позволяющую принимать данные и устанавливать нужные ШИМ сигналы на указанных выводах микроконтроллера.

В4. С помощью протокола UART микроконтроллеру Atmega328 передается пакет, данных вид, которого представлен ниже

?75, 100, 255;

Числа в пакете — это задание для ШИМ сигналов на 3, 5, 9 выходах платы разработчика соответственно. Составьте программу для микроконтроллера позволяющую принимать данные и устанавливать нужные ШИМ сигналы на указанных выводах микроконтроллера.

## Типовое решение задачи В

В программе реализована пользовательская функция pars() (описанная с 25 по 48 строку кода) позволяющая разбивать принятую строку типа \$100, 125, 255; на отдельные составляющие и записывать переменные в массив intData[]. Данную функцию будем называть парсинг.

С помощью команды Serial.available() > 0 проверяем наличие данных в порту UART, если они есть, то в 27 строке считываем данные с помощью команды Serial.read() и заносим в переменную incomingByte. Далее в 28 строке проверяем флаг getStart, который отвечает за начало функции парсинга. Если этот флаг равен true, то начинаем парсинг. В 29 строке проверяем принятые данные не являются пробелом (' ') или концом данных (';') то полученные символы записываем в строку string\_convert 31 строка. Если же полученный символ не является числом, то строку string\_convert преобразуем в число и в 33 строке преобразованное число записываем в массив intData[ind] , где ind порядковый номер идентификатора. В 35 строке инкрементируем идентификатор и в 34 строке очищаем строку string\_convert .

В 38 строке осуществляем проверку принятого символа с '\$', если происходит совпадение, то флаг getStart устанавливается в true, обнуляется идентификатор массива ind и обнуляется строка string\_convert

В 39 строке осуществляем проверку принятого символа с ';', если происходит совпадение, то флаг getStart устанавливается в false, и устанавливается флаг принятия данных resivData.

Функцию pars()вызывается в основном цикле программы где проверяется флаг принятия данных resivData, если он true, то в 13-16 строках выводим данные массива intData[] в протокол UART, после вывода данных устанавливаем флаг принятия данных resivData в false. В 21 -23 строках с помощью функции analogWrite(port, intData[ind]) устанавливаем ШИМ сигнал в 5, 9, 10 портах платы разработчика.



```

1. #define ch 2
2. bool getStart;
3. bool resivData;
4. String string_convert = "";
5. int intData[ch];
6. byte ind;
7. void setup() {
8.   Serial.begin(9600);
9. }
10. void loop() {
11.   pars();
12.   if (resivData) {
13.     for (byte i = 0; i < ch; i++) {
14.       Serial.print(intData[i]);
15.       Serial.print(" ");
16.     }
17.     Serial.println();
18.     resivData = false;
19.   }
20.   analogWrite(5, intData[0]);
21.   analogWrite(9, intData[1]);
22.   analogWrite(10, intData[2]);
23. }
24. void pars() {
25.   if (Serial.available() > 0) {
26.     char incomingByte = Serial.read(); // читаем входящий символ
27.     if (getStart) {
28.       if (incomingByte != ' ' && incomingByte != ';') //если это не пробел и не конец данных
29.       {
30.         string_convert += incomingByte;
31.       } else {
32.         intData[ind] = string_convert.toInt(); // преобразуем строку в int и вносим в массив
33.         string_convert = ""; //очищаем строку
34.         ind++; // переходим к захвату следующего элемента массива
35.       }
36.     }

```

```

37.   if (incomingByte == '$') {
38.     getStart = true; // начинаем парсинг сигнала
39.     ind = 0;
40.     string_convert = "";
41.   }
42.   if (incomingByte == ';') {
43.     getStart = false; // окончание парсинга
44.     resivData = true; // флаг принятия данные
45.   }
46. }
47. }

```

## Прерывание 12 задач

### БАЗОВЫЙ

B1. С помощью внешнего прерывания определить количество нажатий на тактовую кнопку, подключенную к 2 пину платы разработчика в микроконтроллере atmega328. Количество нажатий выводить в последовательный интерфейс UART. Использовать в коде язык программирования Arduino.

B2. С помощью внешнего прерывания определить количество нажатий на тактовую кнопку, подключенную к 3 пину платы разработчика в микроконтроллере atmega328. Количество нажатий выводить в последовательный интерфейс UART. Использовать в коде язык программирования Arduino.

B3. При нажатии на кнопку 4 раза засветить светодиод подключенному к 13 порту платы разработчика, при нажатии 3 раз светодиод отключить. Реализовать данный алгоритм на внешнем прерывании микроконтроллера atmega328. Использовать в коде язык программирования Arduino.

B4. При нажатии на кнопку 5 раз засветить светодиод подключенному к 12 порту платы разработчика, при нажатии 2 раза светодиод отключить. Реализовать данный алгоритм на внешнем прерывании микроконтроллера atmega328. Использовать в коде язык программирования Arduino.

### Типовое решение задачи B1-B2

В первой строке инициализируем переменную отвечающую за количество нажатий. В 4 строке инициализируем последовательный протокол передачи данных UART. В 5 строке вызывается функция `attachInterrupt` для настройки внешнего прерывания.

**attachInterrupt(interrupt, function, mode)**

**interrupt:** номер прерывания (int)

**function:** функция, вызываемая прерыванием, функция должна быть без параметров и не возвращать значений.

**mode** задает режим обработки прерывания.

Допустимо использование следующих констант:

**LOW** вызывает прерывание, когда на порту LOW

**CHANGE** прерывание вызывается при смене значения на порту, с LOW на HIGH и наоборот

**RISING** прерывание вызывается только при смене значения на порту с LOW на HIGH

**FALLING** прерывание вызывается только при смене значения на порту с HIGH на LOW

При возникновении события внешнего прерывания на 2 порту платы разработчика вызывается обработчик прерывания `INT0_vect()` в котором инкрементируется переменная `a`. В основном цикле программы происходит вывод переменной `a` в последовательный интерфейс UART.

```
1. int a=0; // количество импульсов
2. void setup()
3. {
4.   Serial.begin(9600);
5.   attachInterrupt(0, INT0_vect, FALLING);
6. }
7. void loop()
8. {
9.   //выводим количество импульсов
10.  Serial.print(" a =");
11.  Serial.println(a);
12. }
13. void INT0_vect()
14. {
15.   a++;
16. }
```

### Типовое решение задачи Б3-Б4

```
1. int a=0; // количество импульсов
2. void setup()
3. {
4.   Serial.begin(9600);
5.   pinMode(13, OUTPUT);
6.   attachInterrupt(0, INT0_vect, FALLING);
7. }
8. void loop()
9. {
10.  //выводим количество импульсов
11.  Serial.print(" a =");
12.  Serial.println(a);
13.  if (a==4) digitalWrite(13, 1);
14.  if (a==7)
15.  {
16.    digitalWrite(13, 0);
17.    a=0;
18.  }
19. }
20. void INT0_vect()
21. {
22.   a++;
23. }
```

### СРЕДНИЙ

C1. С помощью внешнего прерывания определить количество нажатий на тактовую кнопку, подключенную к 2 пину платы разработчика в микроконтроллере `atmega328`. Количество нажатий выводить в последовательный интерфейс UART. Использовать в коде язык программирования C++.

C2. С помощью внешнего прерывания определить количество нажатий на тактовую кнопку, подключенную к 3 пину платы разработчика в микроконтроллере `atmega328`. Количество нажатий выводить в последовательный интерфейс UART. Использовать в коде язык программирования C++.

C3. При нажатии на кнопку 4 раза засветить светодиод подключенному к 13 порту платы разработчика, при нажатии 3 раз светодиод отключить. Реализовать данный алгоритм на внешнем прерывании микроконтроллера `atmega328`. Использовать в коде язык программирования C++.

C4. При нажатии на кнопку 5 раз засветить светодиод подключенному к 13 порту платы разработчика, при нажатии 2 раза светодиод отключить. Реализовать данный алгоритм на внешнем прерывании микроконтроллера `atmega328`. Использовать в коде язык программирования C++.

### Типовое решение задачи C1-C2.

В первой строке инициализируем переменную отвечающую за количество нажатий. В 4 строке инициализируем последовательный протокол передачи данных UART. В 5 строке вызывается пользовательскую функцию `int_init()` для настройки внешнего прерывания. 15 и 22 строка содержит код функции `int_init()`. 18 и 19 строках настраивается прерывание по возрастающему сигналу на входе INT0, для этого в биты регистра EICRA записывается лог. 1 в соответствующие биты ISC01 и ISC00. Битом INT0 регистра EIMSK в 21 строке разрешаем прерывание. При возникновении события внешнего прерывания на 2 порту платы разработчика вызывается обработчик прерывания `ISR(INT0_vect)` в котором инкрементируется переменная `a`. 6 строка разрешает все прерывания. В основном цикле программы происходит вывод переменной, `a` в последовательный интерфейс UART.

```
1. int a=0; // количество импульсов
2. void setup()
3. {
4.   Serial.begin(9600);
5.   int_init();
6.   sei();
7. }
8. void loop()
9. {
10.  //выводим количество импульсов
11.  Serial.print(" a =");
12.  Serial.println(a);
13. }
14. }
15. void int_init(void)
16. {
17.  //включим прерывания INT0 по нисходящему фронту
18.  EICRA |= (1<<ISC00);
19.  EICRA |= (1<<ISC01);
20.  //разрешаем внешние прерывания INT0
21.  EIMSK |= (1<<INT0);
22. }
23. // обработчик прерывания
24. ISR(INT0_vect)
25. {
26.   a++;
27. }
```

## Типовое решение задачи С3-С4.

```
1. int a=0; // количество импульсов
2. void setup()
3. {
4.   Serial.begin(9600);
5.   pinMode(13, OUTPUT);
6.   int_ini();
7.   sei();
8. }
9. void loop()
10. {
11. //выводим количество импульсов
12. Serial.print(" a =");
13. Serial.println(a);
14. if (a==4) digitalWrite(13, 1);
15. if (a==7)
16. if (a==7)
17. {
18.   digitalWrite(13, 0);
19.   a=0;
20. }
21. }
22. void int_ini(void)
23. {
24. //включим прерывания INT0 по нисходящему фронту
25. EICRA |= (1<<ISC00);
26. EICRA |= (1<<ISC01);
27. //разрешаем внешние прерывания INT0
28. EIMSK |= (1<<INT0);
29. }
30. // обработчик прерывания
31. ISR(INT0_vect)
32. {
33.   a++;
34. }
```

### ВЫСОКИЙ

В1. С помощью инкрементного энкодера определить скорость вращения мотора. Цифровой вывод энкодера подключен ко 2 порту платы разработчика. Вывести данные в протокол последовательного порта.

В2. С помощью инкрементного энкодера определить скорость вращения мотора. Цифровой вывод энкодера подключен ко 3 порту платы разработчика. Вывести данные в протокол последовательного порта.

В3. В проекте используется двигатель с энкодером. Необходимо разработать программу, которая включает красный светодиод при достижении скорости двигателя больше 100 об/мин, если обороты будут меньше - красный светодиод гасим.

В3. В проекте используется двигатель с энкодером. Необходимо разработать программу, которая включает зеленый светодиод при достижении скорости двигателя больше 500 об/мин, если обороты будут меньше - зеленый светодиод гасим.

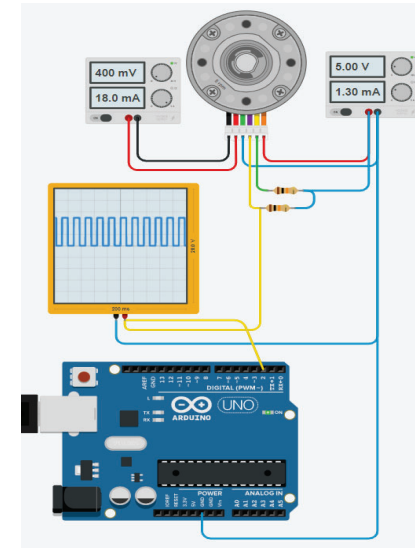


Схема подключения

## Типовое решение задачи В1-В2

Задача заключается в следующем инкрементировать при каждом импульсе от датчика энкодера при вращении двигателя и при достижении определенного значения времени производить расчет скорости. Рассчитанную скорость вывести в последовательный интерфейс UART.

Двигатель подключается через регулируемый источник напряжения, изменяя напряжение будем менять скорость двигателя. Двигатель содержит импульсный энкодер при полном обороте которого на выводах А и В появляются 660 импульсов. Энкодер питается также от независимого источника напряжения. Для отслеживания импульсов от энкодера в цепи измерения подключен осциллограф. Импульсы от энкодера подаются на 2 порт платы разработчика, функцией которого является внешнее прерывание INT0.

Листинг программы реализующий данный алгоритм представлен ниже:

```
1. unsigned long t=0;
2. volatile int a=0; // количество импульсов
3. float speed=0;
4. void setup()
5. {
6.   Serial.begin(9600);
7.   int_ini(); // инициализация внешнего прерывания INT0
8.   sei(); // разрешение глобального прерывания
9. }
10. void loop()
11. {
12. //выводим количество импульсов
13. if (millis() - t>250)
14. {
15.   speed = (a*4*60/660); // расчет скорости 660 им/об
16.   a=0; // сбрасываем счетчик при достижении времени 250 мс
17.   t=millis();
18.   Serial.print(" speed =");
```

```

19. Serial.println(speed);
20. }
21. }
22. void int_ini(void)
23. {
24. //включим прерывания INTO по нисходящему фронту
25. EICRA |= (1<<ISC00);
26. EICRA |= (1<<ISC01);
27. //разрешаем внешние прерывания INTO
28. EIMSK |= (1<<INT0);
29. }
30. // обработчик прерывания
31. ISR(INT0_vect)
32. {
33. a++; // счетчик импульсов от датчика
34. }

```

В строках с 22 по 29 осуществляем настройку внешнего прерывания, которое вызывается по нисходящему фронту сигнала поданного на 2 порт платы разработчика. Для этого в биты ISC00 и ISC01 регистра EICRA устанавливается лог. 1. Установкой лог. 1 в бит INTO регистра EIMSK разрешаем внешнее прерывание. При возникновении внешнего прерывания вызывается программа обработки прерывания ISR(INT0\_vect) в которой инкрементируется счетчик импульсов. В основной программе формируем таймер, который обновляется каждые 250 мс. За каждые 250 мс если двигатель совершает вращение происходит инкрементирование переменной а. Поэтому с сторке 15 используем следующую формулу для расчета скорости:

$$\text{speed} = a^4 \cdot 60 / 660$$

где а - количество импульсов за 250 мс;

4 - коэффициент чтобы получить кол-во импульсов за секунду;  
60 - коэффициент чтобы получить кол-во импульсов за минуту;  
660 - количество импульсов за оборот;  
speed - рассчитанная скорость.

Передаем полученную скорость в последовательный протокол порта.

## Типовое решение задачи В3-В4

```

unsigned long t=0;
volatile int a=0; // количество импульсов
float speed=0;
void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);
  int_ini(); // инициализация внешнего прерывания INTO
  sei(); // разрешение глобального прерывания
}
void loop()
{
  //выводим количество импульсов
  if (millis() - t>250)
  {
    speed = (a^4*60/660); // расчет скорости 660 им/об
    a=0; // сбрасываем счетчик при достижении времени 250 мс
    t=millis();
    Serial.print(" speed =");

```

```

Serial.println(speed);
if (speed > 100) digitalWrite(13, 1); else digitalWrite(13, 0);
}
}
void int_ini(void)
{
  //включим прерывания INTO по нисходящему фронту
  EICRA |= (1<<ISC00);
  EICRA |= (1<<ISC01);
  //разрешаем внешние прерывания INTO
  EIMSK |= (1<<INT0);
}
// обработчик прерывания
ISR(INT0_vect)
{
  a++; // счетчик импульсов от датчика
}

```

## EEPROM постоянная память данных 12 задач БАЗОВЫЙ

Б1. Записать данные в энергонезависимую память EEPROM по адресу 523 с температурного датчика, подключенного к аналоговому порту A0 платы разработчика. Вывести данные в интерфейс последовательного порта UART. Используя библиотеку EEPROM.

Б2. Записать данные в энергонезависимую память EEPROM по адресу 524 с газового датчика, подключенного к аналоговому порту A1 платы разработчика. Вывести данные в интерфейс последовательного порта UART. Используя библиотеку EEPROM.

Б3. Записать данные в энергонезависимую память EEPROM по адресу 525 с потенциометра, подключенного к аналоговому порту A2 платы разработчика. Вывести данные в интерфейс последовательного порта UART. Используя библиотеку EEPROM.

Б4. Записать данные в энергонезависимую память EEPROM по адресу 526 с датчика освещенности, подключенного к аналоговому порту A3 платы разработчика. Вывести данные в интерфейс последовательного порта UART. Используя библиотеку EEPROM.

## Типовое решение задачи Б.

Для упрощения работы с EEPROM используется библиотека EEPROM.h. В данной библиотеке используются следующие функции:

**EEPROM.write(адрес, данные)** - – пишет данные (только byte!) по адресу

**EEPROM.update(адрес, данные)** – обновляет байт данных, находящийся по адресу.

**EEPROM.read(адрес)** – читает и возвращает байт данных, находящийся по адресу

**EEPROM.put(адрес, данные)** – записывает данные любого типа (типа переданной переменной) по адресу

**EEPROM.get(адрес, данные)** – читает данные по адресу и сам записывает их в данные – указанную переменную

**EEPROM[]** – библиотека позволяет работать с EEPROM памятью как с обычным массивом типа byte (uint8\_t)

1. #include <EEPROM.h>
2. int int\_data =0;

```

3. byte byte_data =0;
4. void setup() {
5.   Serial.begin(9600);
6.   int_data = analogRead(A0);
7.   byte_data = map (int_data, 0, 1024, 0, 255);
8.   // пишем byte_data по 523
9.   EEPROM.update(523, byte_data);
10.  Serial.println(EEPROM.read(523)); // выведет 123
11. }
12. void loop() {}

```

### СРЕДНИЙ

C1. Записать данные в энергонезависимую память EEPROM по адресу 523 с температурного датчика, подключенного к аналоговому порту A0 платы разработчика. Вывести данные в интерфейс последовательного порта UART. В коде использовать регистры управления энергонезависимой памяти EEPROM.

C2. Записать данные в энергонезависимую память EEPROM по адресу 524 с газового датчика, подключенного к аналоговому порту A1 платы разработчика. Вывести данные в интерфейс последовательного порта UART. В коде использовать регистры управления энергонезависимой памяти EEPROM.

C3. Записать данные в энергонезависимую память EEPROM по адресу 525 с потенциометра, подключенного к аналоговому порту A2 платы разработчика. Вывести данные в интерфейс последовательного порта UART. В коде использовать регистры управления энергонезависимой памяти EEPROM.

C4. Записать данные в энергонезависимую память EEPROM по адресу 526 с датчика освещенности, подключенного к аналоговому порту A3 платы разработчика. Вывести данные в интерфейс последовательного порта UART. В коде использовать регистры управления энергонезависимой памяти EEPROM.

### Типовое решение задачи С.

```

1. int int_data =0;
2. byte byte_data =0;
3. void setup() {
4.   Serial.begin(9600);
5.   int_data = analogRead(A0);
6.   byte_data = map (int_data, 0, 1024, 0, 255);
7.   // пишем byte_data по адресу 523
8.   EEPROM_write (523, byte_data);
9.   Serial.println(EEPROM_read(523)); // выведет 123
10. }
11. void loop() {}
12. unsigned char EEPROM_read(unsigned int uiAddress){

```

```

13. while(EECR & (1<<EEPE)); // проверка готовности EEPROM
14.  EEARH = ((uiAddress & 0xF0) << 2); // регистр адреса H
15.  EEARL = uiAddress & 0x0F; // регистр адреса L
16.  EECR |= (1<<EERE); // чтение EEPROM
17.  return EEDR; // вывод значения
18. }
19. void EEPROM_write(unsigned int uiAddress, unsigned char ucData){
20.  while(EECR & (1<<EEPE)); // проверка готовности EEPROM
21.  EEARH = ((uiAddress & 0xF0) << 2); // регистр адреса H
22.  EEARL = uiAddress & 0x0F; // регистр адреса L
23.  EEDR = ucData; // регистр данных
24.  EECR |= (1<<EEMPE); // Разрешение записи в EEPROM
25.  EECR |= (1<<EEPE); // Запись в EEPROM
26. }

```

### ВЫСОКИЙ

V1. Записать в энергонезависимую память EEPROM по адресу 523 данные с температурного датчика, подключенного к аналоговому порту A0 платы разработчика, а в память по адресу 524 с газового датчика, подключенного к аналоговому порту A1 платы разработчика. Вывести данные в интерфейс последовательного порта UART. В коде использовать регистры управления энергонезависимой памяти EEPROM.

V2. Записать в энергонезависимую память EEPROM по адресу 525 данные с температурного датчика, подключенного к аналоговому порту A2 платы разработчика, а в память по адресу 526 с газового датчика, подключенного к аналоговому порту A3 платы разработчика. Вывести данные в интерфейс последовательного порта UART. В коде использовать регистры управления энергонезависимой памяти EEPROM.

V3. Записать в энергонезависимую память EEPROM по адресу 527 данные с температурного датчика, подключенного к аналоговому порту A4 платы разработчика, а в память по адресу 528 с газового датчика, подключенного к аналоговому порту A5 платы разработчика. Вывести данные в интерфейс последовательного порта UART. В коде использовать регистры управления энергонезависимой памяти EEPROM.

V4. Записать в энергонезависимую память EEPROM по адресу 528 данные с температурного датчика, подключенного к аналоговому порту A4 платы разработчика, а в память по адресу 529 с датчика освещенности, подключенного к аналоговому порту A5 платы разработчика. Вывести данные в интерфейс последовательного порта UART. В коде использовать регистры управления энергонезависимой памяти EEPROM.

### Типовое решение задачи В.

```

1. int int_data =0;
2. byte byte_data =0;
3. void setup() {
4.   Serial.begin(9600);
5.   int_data = analogRead(A0);
6.   byte_data = map (int_data, 0, 1024, 0, 255);
7.   // пишем byte_data по адресу 523
8.   EEPROM_write (523, byte_data);

```

```

9. Serial.println(EEPROM_read(523)); // выведет byte_data
10. int_data = analogRead(A1);
11. byte_data = map (int_data, 0, 1024, 0, 255);
12. // пишем byte_data по адресу 524
13. EEPROM_write (524, byte_data);
14. Serial.println(EEPROM_read(524)); // выведет byte_data
15. }
16. void loop() {}
17. unsigned char EEPROM_read(unsigned int uiAddress){
18. while(EECR & (1<<EEPE)); // проверка готовности EEPROM
19.  EEARH = ((uiAddress & 0xF0) << 2); // регистр адреса H
20.  EEARL = uiAddress & 0x0F; // регистр адреса L
21.  EECR |= (1<<EERE); // чтение EEPROM
22.  return EEDR; // вывод значения
23. }
24. void EEPROM_write(unsigned int uiAddress, unsigned char ucData){
25. while(EECR & (1<<EEPE)); // проверка готовности EEPROM
26.  EEARH = ((uiAddress & 0xF0) << 2); // регистр адреса H
27.  EEARL = uiAddress & 0x0F; // регистр адреса L
28.  EEDR = ucData; // регистр данных
29.  EECR |= (1<<EEMPE); // Разрешение записи в EEPROM
30.  EECR |= (1<<EEPE); // Запись в EEPROM
31. }

```

## Раздел 5. Проекты с использованием микроконтроллера

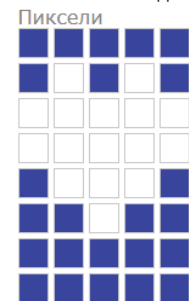
### Проекты с использованием микроконтроллера.LCD экран 1602.

#### Создание игры на микроконтроллере

12 задач

##### БАЗОВЫЙ

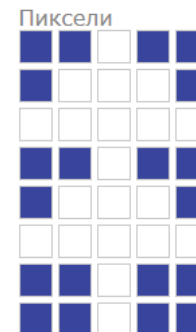
Б1. Вывести на дисплеи LCD1602 пользовательский глиф представленный на рисунке.



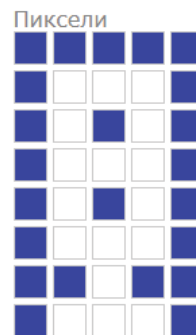
Б2. Вывести на дисплеи LCD1602 пользовательский глиф представленный на рисунке.



Б3. Вывести на дисплеи LCD1602 пользовательский глиф представленный на рисунке.



Б4. Вывести на дисплеи LCD1602 пользовательский глиф представленный на рисунке.



#### Типовое решение задачи Б.

Для работы с дисплеем воспользуемся командами библиотеки **LiquidCrystal.h**:

##### LiquidCrystal lcd(4, 5, 10, 11, 12, 13)

Создает объект lcd библиотеки LiquidCrystal, где 4, 5, 10, 11, 12, 13 – номера контактов платы разработчика подключенные соответственно к LCD дисплею: RS, E, D4, D5, D6, D7.

**lcd.begin()** - команда инициализирует стар дисплея

**lcd.setCursor()** - команда устанавливает курсор в заданную позицию

**lcd.print()** - команда выводит информацию на экран дисплея

```
1. #include <LiquidCrystal.h>
2. LiquidCrystal lcd(4, 5, 10, 11, 12, 13); // инициализация объекта lcd библиотеки
3. byte customChar1[8] = {
4. 0b00000,
5. 0b01010,
6. 0b11111,
7. 0b11111,
8. 0b01110,
9. 0b00100,
10. 0b00000,
11. 0b00000
12. };
13. void setup()
14. {
15. lcd.begin(16, 2);
16. lcd.createChar(0, customChar1);
17. lcd.setCursor(0,0);
18. lcd.write(byte(0));
19. }
20. void loop() { }
```

#### СРЕДНИЙ

C1. Вывести одновременно на дисплеи LCD1602 данные с аналоговых датчиков температуры, подключенных к портам A0, A1, A2, A3 платы разработчика. Данные с датчиков должны обновляться при изменении их параметров.

C2. Вывести одновременно на дисплеи LCD1602 данные с аналоговых датчиков освещенности, подключенных к портам A2, A3, A4, A5 платы разработчика. Данные с датчиков должны обновляться при изменении их параметров.

C3. Вывести одновременно на дисплеи LCD1602 данные с аналоговых датчиков CO2, подключенных к портам A1, A3, A4, A5 платы разработчика. Данные с датчиков должны обновляться при изменении их параметров.

C3. Вывести одновременно на дисплеи LCD1602 данные с потенциометров, подключенных к портам A1, A3, A4, A5 платы разработчика. Данные с потенциометров должны обновляться при изменении их параметров.

#### Типовое решение задачи С.

Для работы с дисплеем воспользуемся командами библиотеки **LiquidCrystal.h**:

##### **LiquidCrystal lcd(4, 5, 10, 11, 12, 13)**

Создает объект lcd библиотеки LiquidCrystal, где 4, 5, 10, 11, 12, 13 – номера контактов платы разработчика подключенные соответственно к LCD дисплею: RS, E, D4, D5, D6, D7.

**lcd.begin()** - команда инициализирует стар дисплея

**lcd.setCursor()** - команда устанавливает курсор в заданную позицию

**lcd.print()** - команда выводит информацию на экран дисплея

В 1 строке программы подключаем библиотеку LiquidCrystal. Во 2 строке задаем переменную time необходимую для настройки таймера для очистки дисплея. В 3 строке используя функцию define вводим параметр T\_period1 который отвечает за период обновления дисплея. В 4 строке

производим инициализацию объекта lcd библиотеки LiquidCrystal, указав какие пины платы разработчика 4, 5, 10, 11, 12, 13 будут подключены к пинам дисплея RS, E, D4, D5, D6, D7. С 5 по 14 строку формируем одномерный массив customChar1[8] из 8 символов заполненный параметрами представленный в битовой форме записи.

В функции setup производим активацию дисплея строка 17 lcd.begin(16, 2) задействуем 16 символов и 2 строки. В основной цикл loop, где в 11 строке используя функцию if ...else... сформирован таймер настроенный на обновление с периодом T\_period1, также после обновления таймера происходит очистку экрана (14 строка программы). Данное очищение экрана сделано для корректного отображения информации. Убедится в этом можно если закоментировать 14 строку и запустив программы изменить положение потенциометра от максимального значения до минимального. Пока таймер не сбросится, то выполняем строки программы с 18 по 42. Перед выводом информации в 18 строке программы устанавливаем курсор в позицию 0 символ 0 строка lcd.setCursor(0,0). Далее в 19 строке вводим на экран t1. Далее в строке 20 lcd.setCursor(2,0) передвигаем курсор в позицию 2 символ 0 строка и в 21 строке программы выводим символ "=" и после этого в строке 22 lcd.setCursor(3,0) передвигаем курсор в позицию 3 символ 0 строка и в 23 строке выводим данные считанные с аналогового порта A0 lcd.print(analogRead(A0)). Аналогичные перемещения курсора по дисплею и вывод информации на экран осуществляется в строка 24-42.

Рассмотрим листинг программы, отображающий информацию на LCD дисплее.

```
1. #include <LiquidCrystal.h>
2. unsigned long time1 =0;
3. #define T_period1 250
4. LiquidCrystal lcd(4, 5, 10, 11, 12, 13); // инициализация объекта lcd библиотеки
5. void setup()
6. {
7. lcd.begin(16, 2);
8. }
9. void loop()
10. {
11. if (millis()-time1>=T_period1)
12. {
13. time1 = millis();
14. lcd.clear();
15. }
16. else
17. {
18. lcd.setCursor(0,0);
19. lcd.print("t1");
20. lcd.setCursor(2,0);
21. lcd.print("=");
22. lcd.setCursor(3,0);
23. lcd.print(analogRead(A0));
24. lcd.setCursor(8,0);
25. lcd.print("t3");
26. lcd.setCursor(10,0);
27. lcd.print("=");
28. lcd.setCursor(11,0);
29. lcd.print(analogRead(A2));
30. lcd.setCursor(0,1);
31. lcd.print("t2");
32. lcd.setCursor(2,1);
33. lcd.print("=");
34. lcd.setCursor(3,1);
35. lcd.print(analogRead(A1));
```

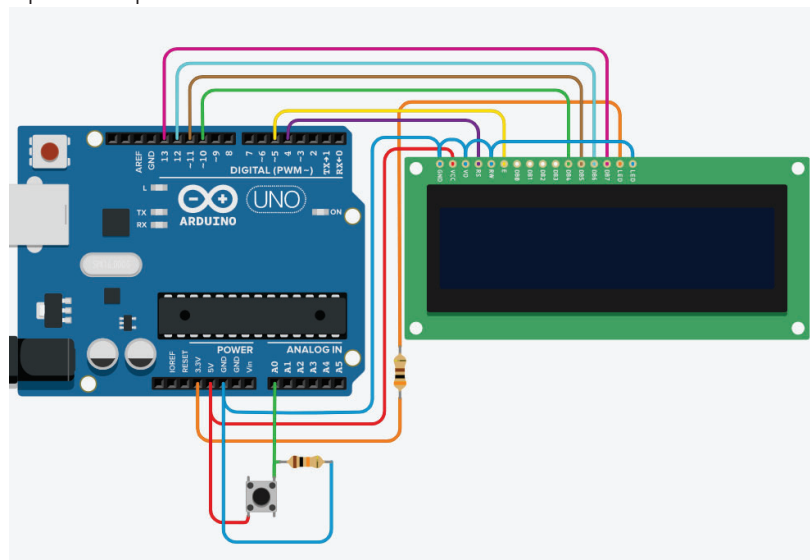
```

36. lcd.setCursor(8,1);
37. lcd.print("t4");
38. lcd.setCursor(10,1);
39. lcd.print("=");
40. lcd.setCursor(11,1);
41. lcd.print(analogRead(A3));
42. }
43. }

```

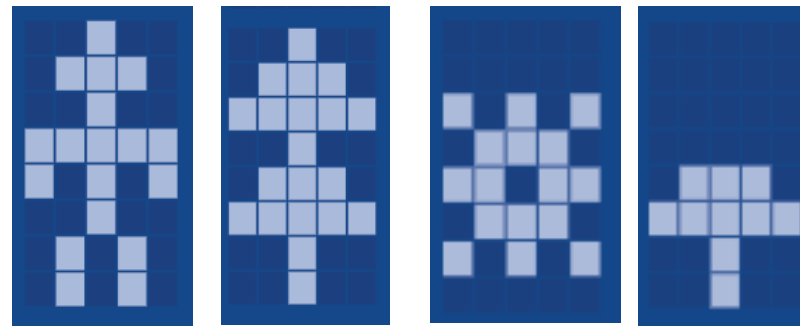
## ВЫСОКИЙ

В1. Разработать игру с использованием микроконтроллера, дисплея LCD1602 и тактовой кнопки. Электрическая схема компонентов представлена на рисунке. Герою необходимо перепрыгнуть через препятствия, если герой перепрыгнул через препятствие или миновал его, ведем счет. Увеличиваем счет каждый раз когда герой миновал препятствие. Если герой минует 5 преград увеличиваем скорость его перемещения. Если герой столкнулся с препятствием то необходимо вывести слово GAME OVER , обнулить счет и вернуть скорость на начальную. Управлять роботом можно с помощью одной кнопкой: если кнопка нажата герой прыгает, если кнопка не нажата герой перемещается без прыжка. Препятствия в игре меняются после того как они миновали героя и скрылись с экрана.



Электрическая схема аппаратной части игры

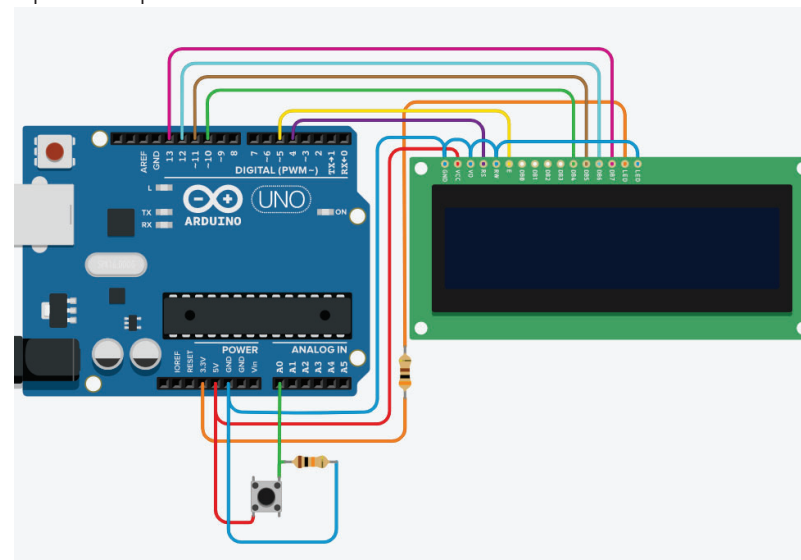
Герой - в нашем примере это робот. Реализованный в виде глифа. Препятствия: дерево, гриб и снежинка, также реализованы в виде глифов, при этом дерево и гриб перемещаются вниз экрана и снежинка вверх экрана.



а б в г

Действующие лица игры (а - робот, б - дерево, в - снежинка, г - гриб)

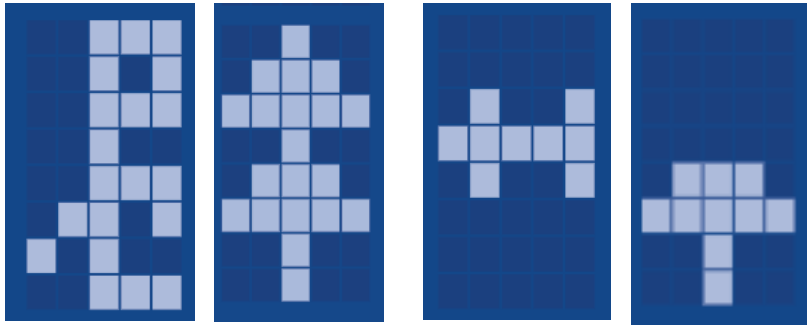
В2. Разработать игру с использованием микроконтроллера, дисплея LCD1602 и тактовой кнопки. Электрическая схема компонентов представлена на рисунке. Герою необходимо перепрыгнуть через препятствия, если герой перепрыгнул через препятствие или миновал его, ведем счет. Увеличиваем счет каждый раз когда герой миновал препятствие. Если герой минует 5 преград увеличиваем скорость его перемещения. Если герой столкнулся с препятствием то необходимо вывести слово GAME OVER , обнулить счет и вернуть скорость на начальную. Управлять роботом можно с помощью одной кнопкой: если кнопка нажата герой прыгает, если кнопка не нажата герой перемещается без прыжка. Препятствия в игре меняются после того как они миновали героя и скрылись с экрана.



Электрическая схема аппаратной части игры

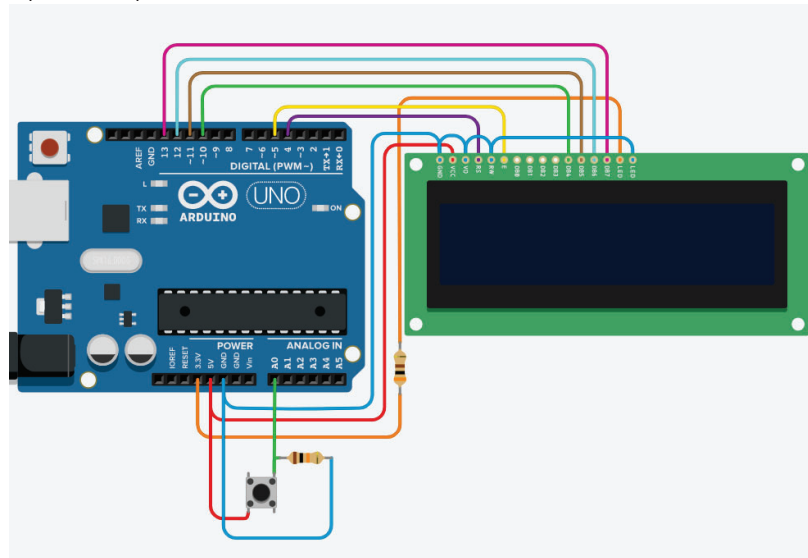
Герой - в нашем примере это дракон. Реализованный в виде глифа. Препятствия: дерево, гриб и стрела, также реализованы в виде глифов, при этом дерево и гриб перемещаются вниз экрана и стрела вверх экрана.





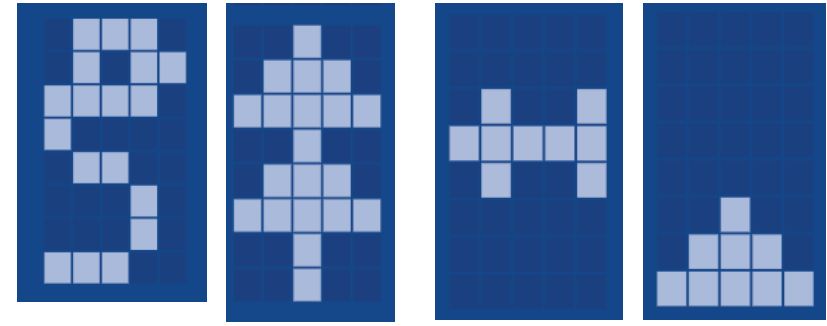
а б в г  
 Действующие лица игры (а - дракон, б - дерево, в - стрела, г - гриб)

В3. Разработать игру с использованием микроконтроллера, дисплея LCD1602 и тактовой кнопки. Электрическая схема компонентов представлена на рисунке. Герою необходимо перепрыгивать через преграды, если герой перепрыгнул через препятствие или миновал его, ведем счет. Увеличиваем счет каждый раз, когда герой миновал препятствие. Если герой минует 5 преград увеличиваем скорость его перемещения. Если герой столкнулся с препятствием то необходимо вывести слово GAME OVER , обнулить счет и вернуть скорость на начальную. Управлять роботом можно с помощью одной кнопкой: если кнопка нажата герой прыгает, если кнопка не нажата герой перемещается без прыжка. Преграды в игре меняются после того как они миновали героя и скрылись с экрана.



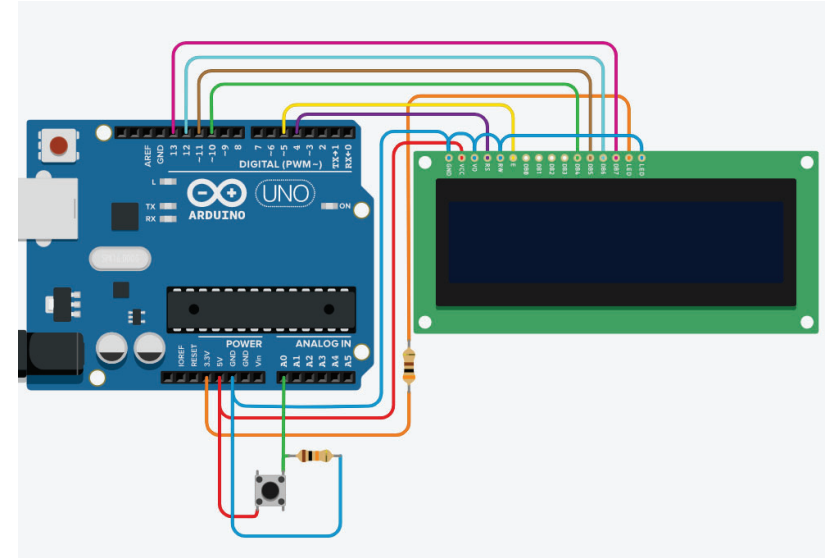
Электрическая схема аппаратной части игры

Герой - в нашем примере это змейка. Реализованный в виде глифа. Преграды: дерево, снежинка и стрела, также реализованы в виде глифов, при этом дерево и камень перемещаются вниз экрана и стрела вверх экрана.



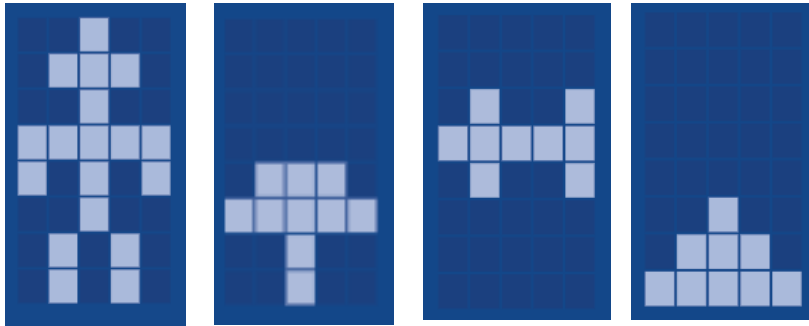
а б в г  
 Действующие лица игры (а - робот, б - дерево, в - снежинка, г - камень)

В4. Разработать игру с использованием микроконтроллера, дисплея LCD1602 и тактовой кнопки. Электрическая схема компонентов представлена на рисунке. Герою необходимо перепрыгивать через преграды, если герой перепрыгнул через препятствие или миновал его, ведем счет. Увеличиваем счет каждый раз, когда герой миновал препятствие. Если герой минует 5 преград увеличиваем скорость его перемещения. Если герой столкнулся с препятствием то необходимо вывести слово GAME OVER , обнулить счет и вернуть скорость на начальную. Управлять роботом можно с помощью одной кнопкой: если кнопка нажата герой прыгает, если кнопка не нажата герой перемещается без прыжка. Преграды в игре меняются после того как они миновали героя и скрылись с экрана.



Электрическая схема аппаратной части игры

Герой - в нашем примере это робот. Реализованный в виде глифа. Преграды: гриб, стрела и камень, также реализованы в виде глифов, при этом дерево и камень перемещаются вниз экрана и стрела вверх экрана.

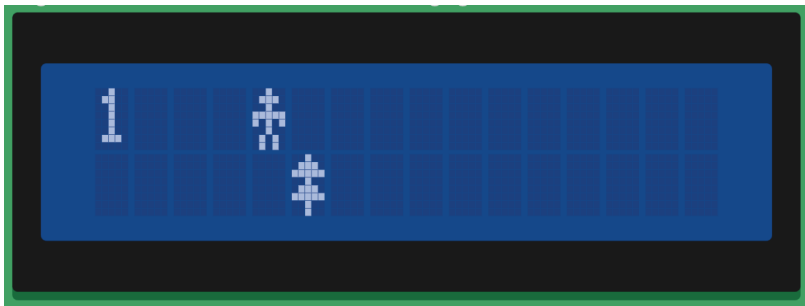


а б в г

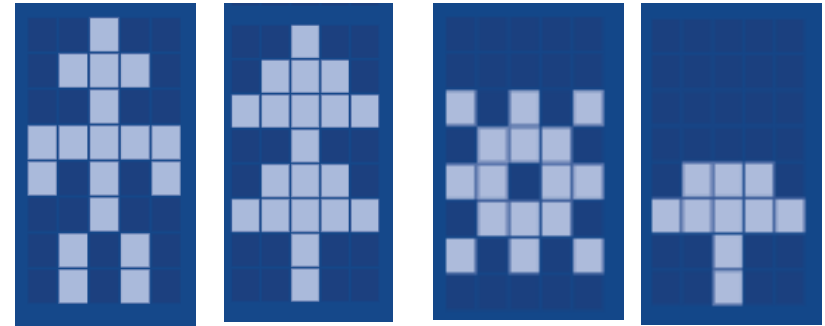
Действующие лица игры (а - робот, б - гриб, в - снежинка, г - камень)

### Типовое решение задачи В.

Рассмотрим основные моменты игры.



Герой - в нашем примере это робот. Реализованный в виде глифа. Преграды: дерево, гриб и снежинка, также реализованы в виде глифов, при этом дерево и гриб перемещаются вниз экрана и снежинка вверх экрана.



а б в г

Действующие лица игры (а - робот, б - дерево, в - снежинка, г - гриб)

### Реализация движения в игре

Осуществляется двумя способами:

#### Прыжок героя

Переход героя из 1 строки дисплея в 0 строку дисплея при нажатии на кнопку, если кнопка не нажата герой располагается в 0 строке дисплея.

Данный алгоритм реализуется в 74 и 81 строках программы. Считываем данные с А0 порта и проверяем если параметр равен лог. 1 то в переменную d отвечающую за ординату героя присваиваем номер 0 строки, если на порт подан лог. 0 - то в d присваивается лог. 0.

```

74. if(digitalRead(A0))
75.   {
76.     d = 0;
77.   }
78.   if(!digitalRead(A0))
79.     {
80.       d = 1;
81.     }

```

#### Движение препятствий

Герой неподвижен по отношению к преградам и располагается всегда на 5 символе дисплея 1 или 0 строки. Перемещаются преграды из 15 символа в 0 символ, т.о. создается эффект движения героя.

Для управления героем и препятствиями в программе вводятся переменные

byte d =0; // ордината героя

byte x =15; // абсцисса препятствия

byte y =0; // ордината препятствия

изменение x начинается с 15 символа и пока значения больше 0 происходит уменьшение на единицу с периодом указанным в переменной T\_period2.

```

68. while (x>0)
69.   {

```

```

...
107.     if (millis()-time1 >= T_period2 )
108.     {
109.         time1= millis();
110.         x--;
111.     }
...
117.     }
118.     x=15;

```

Создание объектов реализуется в виде глифа с помощью матрицы элементов описанных с 35 по 54 строках программы. Сами пользовательские символы создаются в 58 и 61 строках:

```

58. LCD.createChar(0,robot);
59. LCD.createChar(1,grib);
60. LCD.createChar(2,tree);
61. LCD.createChar(3,snej);

```

Причем герою присваивается 0 порядковый номер. Это сделано для того чтобы при произвольном выборе препятствия не был выбран 0 элемент.

```

65.  i = random(1,4);
66.  if(i==3) y=0;
67.  else y=1;

```

Произвольный выбор препятствия осуществляется в 65 строке, где переменная *i* отвечает за номер выбранного препятствия (1 - гриб, 2 - дерево и 3 - снежинка). 66 и 67 строчка программы отвечают за положение препятствия если номер выбранного препятствия равен 3 т.е. это снежинка, то она параметру *y* отвечающего за номер строки дисплея размещения препятствия присваивается 1, а если это гриб или дерево - присваивается 0.

## Проверка столкновения

Проверка столкновения осуществляется в 89 строке программы по условию `if ((x==4) && (d==y))`:

где *x* - абсцисса препятствия;  
*d* - ордината героя;  
*y* - ордината препятствия.

таким образом фиксируем столкновение если ординаты и абсциссы героя и препятствия совпадают, тогда выводим надпись GAME OVER в 91 и 92 строке программы. Кроме этого необходимо сбросить счет обнуляем переменную *p* в 93 строке программы, возвращаем значение 15 ординате препятствия в строке 97 и возвращаем значение периода `T_period2` к 300 (данный параметр отвечает за скорость перемещения объекта). Делаем паузу в 1 секунду (95 строка) и очищаем дисплей (96 строка).

```

89. if ((x==4) && (d==y))
90. {
91.     LCD.setCursor(4,1);
92.     LCD.print("GAME OVER");
93.     p =0;
94.     T_period2 = 300;
95.     delay(1000);
96.     LCD.clear();

```

```

97.     x=15;
98. }

```

## Проверка удачного выполнения миссии и увеличение скорости

Проверка удачного выполнения миссии осуществляется в 100 строке программы по условию `if ((x==4) && !(d==y)&& (millis()-time2 >= T_period1+10 ))`

где *x* - абсцисса препятствия;  
*d* - ордината героя;  
*y* - ордината препятствия.

таким образом фиксируем успешное выполнение миссии если абсциссы героя и препятствия совпадают, а ординаты - не совпадают, тогда переменную отвечающую за счет *p* увеличиваем на единицу. Кроме этого в условии проверки миссии добавлено условие проверки таймера и период обработки таймера настраивается из условия что `T_period1+10 >T_period2`. Это сделано для того чтобы в момент анализа успешного освоения миссии переменная *p* не успевала многократно инкрементировать.

Сделайте эксперимент самостоятельно: уберите в 100 строке проверку `(millis()-time2 >= T_period1+10` и вы увидите что в момент прыжка переменная *p* успевает увеличиться более чем в 1 раз!

В 104 и 105 строке при значениях равных 2 и 4 соответственно происходит уменьшение периода `T_period2`, отвечающего за скорость движения препятствия.

## Вывод героя, препятствия и счета на дисплей

Прорисовка героя и других элементов осуществляется параметрически. За вывода героя на экран отвечают 137 и 138 строки программы

```

137.     LCD.setCursor(4,d);
138.     LCD.write(byte(0));

```

где 4 - абсцисса героя она всегда равна 4, так герой у нас находится всегда на 4 символе дисплея;  
*d* - ордината героя изменяется программно при анализе прыжка героя;  
0 - герой в программе это 0 пользовательский символ

За вывод препятствия на экран отвечают 125 и 126 строки программы:

```

124.     //Рисуем препятствие
125.     LCD.setCursor(x,y);
126.     LCD.write(byte(i));

```

где *x* - абсцисса препятствия;  
*y* - ордината препятствия;  
*i* - номер пользовательского символа (1 - гриб, 2 - дерево , 3 - снежинка )

## Обновление экрана

После того как проверили координаты нарисовали героев и препятствия на в основном цикле движения необходимо обязательно обновить экран. Т.е. наша игра перемещается по кадрам. Таким образом смену экрана необходимо осуществлять с определенной периодичностью, для этого в строках 112 и 116 запускается таймер при срабатывании которого осуществляется очистка экрана в 115 строке программы

```
112.     if (millis()-time >= T_period )
113.     {
114.         time= millis();
115.         LCD.clear();
116.     }
```

Полный листинг программы

```
1. #include <LiquidCrystal.h>
2. unsigned long time =0;
3. #define T_period 50
4. unsigned long time1 =0; // переменная для настройки Таймера 1
5. unsigned long time2 =0; // переменная для настройки Таймера 2
6. #define T_period1 300
7. int T_period2 = 300; // период отвечающий за скорость в игре
8. LiquidCrystal LCD(4, 5, 10, 11, 12, 13);
9. int data =0;
10. byte p=0; // баллы
11. byte d =0; // ордината героя
12. byte x =15; // абсцисса препятствия
13. byte y =0; // ордината препятствия
14. byte i =0; // выбор препятствия
15. byte robot[8] = {
16. 0b00100,
17. 0b01110,
18. 0b00100,
19. 0b11111,
20. 0b10101,
21. 0b00100,
22. 0b01010,
23. 0b01010
24. };
25. byte grib[8] = {
26. 0b00000,
27. 0b00000,
28. 0b00000,
29. 0b00000,
30. 0b01110,
31. 0b11111,
32. 0b00100,
33. 0b00100
34. };
35. byte tree[8] = {
36. 0b00100,
37. 0b01110,
38. 0b11111,
39. 0b00100,
40. 0b01110,
41. 0b11111,
```

```
42. 0b00100,
43. 0b00100
44. };
45. byte snej[8] = {
46. 0b00000,
47. 0b00000,
48. 0b10101,
49. 0b01110,
50. 0b11011,
51. 0b01110,
52. 0b10101,
53. 0b00000
54. };
55. void setup()
56. {
57. LCD.begin(16, 2);
58. LCD.createChar(0,robot);
59. LCD.createChar(1,grib);
60. LCD.createChar(2,tree);
61. LCD.createChar(3,snej);
62. }
63. void loop()
64. {
65. i = random(1,4);
66. if(i==3) y=0;
67. else y=1;
68. while (x>0)
69. {
70. //Рисуем препятствие
71. LCD.setCursor(x,y);
72. LCD.write(byte(i));
73. // управление героем
74. if(digitalRead(A0))
75. {
76. d = 0;
77. }
78. if(!digitalRead(A0))
79. {
80. d = 1;
81. }
82. // Рисуем героя
83. LCD.setCursor(4,d);
84. LCD.write(byte(0));
85. //вывод счета
86. LCD.setCursor(0,0);
87. LCD.print(p);
88. // проверяем столкновение
89. if ((x==4) && (d==y))
90. {
91. LCD.setCursor(4,1);
92. LCD.print("GAME OVER");
93. p =0;
94. T_period2 = 300;
95. delay(1000);
```

```
96.   LCD.clear();
97.   x=15;
98.   }
99.   //подсчет баллов
100.      if ((x==4) && !(d==y)&& (millis()-time2 >= T_period1+10 ))
101.      {
102.          time2 = millis();
103.          p++;
104.          if (p==2) T_period2=200;
105.          if (p==4) T_period2=150;
106.      }
107.      if (millis()-time1 >= T_period2 )
108.      {
109.          time1= millis();
110.          x--;
111.      }
112.      if (millis()-time >= T_period )
113.      {
114.          time= millis();
115.          LCD.clear();
116.      }
117.      }
118.      x=15;
119.      }
```