

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ  
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ  
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

**ПРОГРАММИРОВАНИЕ  
МИКРОКОНТРОЛЛЕРОВ**  
**Часть 1**

Практикум для вузов

Составители:  
А.П. Трифонов,  
Ю.Э. Корчагин,  
В.К. Маршаков,  
К.А. Зимовец

Издательско-полиграфический центр  
Воронежского государственного университета  
2013

Утверждено научно-методическим советом физического факультета «12» марта 2013 года, протокол №3

Рецензент д-р физ.-мат. наук, проф. Е.Н. Бормонтов

Практикум подготовлен на кафедре радиофизики физического факультета Воронежского государственного университета.

Рекомендуется для студентов 4-го курса очной и 5-го курса очно-заочной форм обучения.

Для направления 011800 – Радиофизика, специальности 010801 – Радиофизика и электроника

## СОДЕРЖАНИЕ

Введение .....	4
Общие сведения об устройстве микроконтроллеров .....	5
Принцип работы и структурная схема микроконтроллера .....	5
Структура памяти микроконтроллеров Atmel .....	8
Назначение выводов .....	10
Внутрисхемное программирование .....	11
Лабораторная работа № 1. Операции с параллельными портами ввода-вывода .....	14
Параллельные порты ввода-вывода .....	14
Описание лабораторного макета .....	14
Практические задания .....	16
Пример выполнения работы .....	19
Лабораторная работа № 2. Генератор сигналов .....	25
Описание лабораторного макета .....	25
Прямой цифровой синтез .....	26
Практические задания .....	28
Пример выполнения работы .....	31
Литература .....	38

## **ВВЕДЕНИЕ**

Как известно, подавляющее большинство современных устройств радиотехники и электроники спроектировано на базе цифровых и микропроцессорных систем. Поэтому для студентов-радиофизиков очень актуальными являются знания в области цифровой техники и умение программировать встраиваемые микропроцессорные системы, например микроконтроллеры и сигнальные процессоры.

В данном пособии рассмотрены две лабораторные работы, посвящённые началам программирования микроконтроллеров. Приведены краткие теоретические сведения, необходимые для выполнения лабораторных работ.

Выполнив первую работу, студенты получают навыки программирования микроконтроллера, умение работать с портами ввода-вывода. Вторая работа посвящена синтезу сигнала заданной формы и частоты методом прямого цифрового синтеза.

## ОБЩИЕ СВЕДЕНИЯ ОБ УСТРОЙСТВЕ МИКРОКОНТРОЛЛЕРОВ

### Принцип работы и структурная схема микроконтроллера

Микроконтроллер представляет собой однокристалльную микроЭВМ, выполненную в виде одной микросхемы. В состав этой микросхемы, как правило, входят: процессор, память программ, память данных, тактовый генератор, набор периферийных устройств (порты, таймеры, АЦП и т.д.) Довольно популярными являются микроконтроллеры фирм Atmel, Microchip, Intel и Motorola. В данном пособии рассматривается программирование 8-разрядных микроконтроллеров AVR фирмы Atmel, которые подразделяются на три семейства:

- Classic AVR — основная линия микроконтроллеров с производительностью отдельных модификаций до 16 MIPS, FLASH ROM программ 2–8 Кбайт, EEPROM данных 64–512 байт, SRAM 128–512 байт;
- Mega AVR с производительностью 1–16 MIPS для сложных приложений, требующих большого объёма памяти, FLASH ROM программ 4–128 Кбайт, EEPROM данных 64–512 байт, SRAM 2–4 Кбайт, SRAM 4 Кбайт, встроенный 10-разрядный 8-канальный АЦП, аппаратный умножитель  $8 \times 8$ ;
- Tiny AVR — низкопроизводительные микроконтроллеры в 8-выводном исполнении с низкой стоимостью.

Структурная схема обобщённой микропроцессорной системы изображена на рис.1. Она состоит из процессора, памяти и периферийных устройств, соединённых с шинами данных, адреса и управления. Если память разделена на два устройства — память программ и память данных, — то говорят, что такая микропроцессорная система имеет гарвардскую архитектуру. Если память представляет собой единое устройство для хранения и программ, и данных, то такая микропроцессорная система имеет архитектуру

фон Неймана. Микроконтроллеры AVR построены по гарвардской архитектуре.

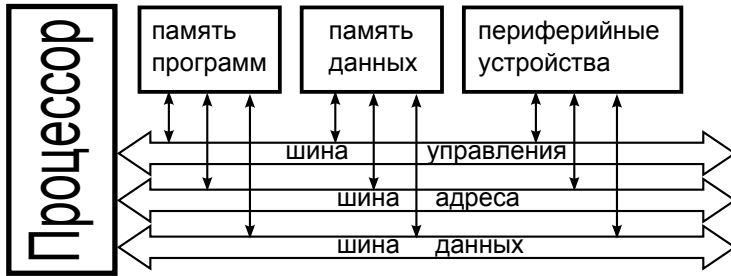


Рис. 1. Блок-схема микропроцессорной системы

Работу процессора обычно иллюстрируют циклом фон Неймана, графическое изображение которого приведено на рис. 2. Здесь обозначено 1 — выборка команды из памяти; 2 — увели-

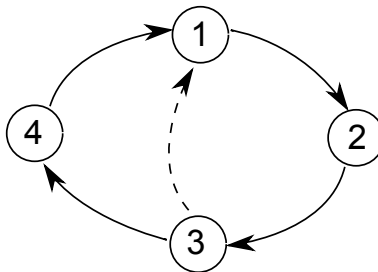
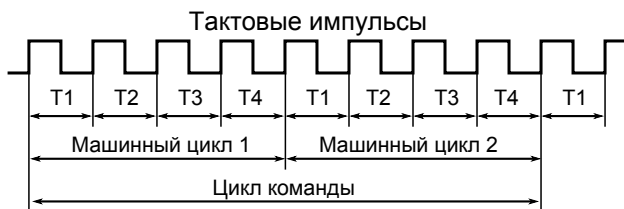


Рис. 2. Цикл фон Неймана

чение на единицу счётчика команд; 3 — дешифрирование кода команды, 4 — выполнение команды. Если код команды содержит более одного байта (двухбайтные, трехбайтные), то есть команда хранится в нескольких ячейках памяти, то процессор может обращаться к памяти 1, 2 и более раз. По первому байту микропроцессор узнаёт, какая это команда. Цикл (1, 2, 3) выполняется столько раз, сколько байтов занимает команда.

Для работы микроконтроллера требуются тактовые импульсы. За время одного тактового импульса процессор выполняет какое-либо элементарное действие (к примеру, выставляет адрес на шину адреса или считывает данные с шины данных). Частота тактовых импульсов определяет быстродействие микроконтроллера. Из нескольких тактовых импульсов складываются машинные циклы. За время машинного цикла происходит одно обращение процессора к памяти или схемам ввода-вывода при выборке и выполнении команды. В зависимости от типа процессора машинные циклы могут содержать либо одинаковое количество тактов, либо разное. Время, в течение которого выбирается и выполняется команда, называется циклом команд. Цикл команд состоит из одного или нескольких машинных циклов. Пример временной диаграммы работы процессора показан на рис. 3. Источниками



*Рис. 3. Временная диаграмма*

тактовых импульсов для микроконтроллера могут служить внешний или внутренний генераторы. Внутренний генератор собран на одном кристалле с микроконтроллером. Частота тактовых импульсов определяется времязадающими цепями генератора, в качестве которых могут быть внутренняя или внешняя RC-цепь, подключенный к микроконтроллеру внешний кварцевый резонатор. По умолчанию используется внутренний тактовый генератор с внутренней RC-цепью. При этом частота тактовых импульсов зависит от модели микроконтроллера. Для ATmega8, используемых в лабораторных макетах, частота примерно равна 1 МГц.

## Структура памяти микроконтроллеров Atmel

**Память программ** микроконтроллеров AVR имеет «плоскую» структуру. Она представляет собой электрически перепрограммируемое постоянное запоминающее устройство (FlashROM) и предназначена для хранения кодов команд программы и констант. Для адресации ячеек памяти программ используется 16-разрядная шина адреса.

**Память данных** микроконтроллеров AVR структурно состоит из трёх разделов, имеющих общую адресацию. Структура памяти данных изображена на рис. 4. В начале находятся 32 восьмиразрядных регистра общего назначения (РОН). К ним можно обращаться из программы по именам r0, r1, ..., r31 и по адресам 0000h – 001Fh в общем адресном пространстве. Здесь и далее приставка 0x или суффикс h означает, что число записано в шестнадцатеричной системе. Далее располагаются 64 регистра ввода-вывода с адресами 0020h – 005Fh. К ним тоже можно обращаться как по адресу, так и по именам регистров. Имена сопоставлены с адресами в файлах описания микроконтроллера, которыми комплектуются системы программирования. Последним регистром ввода/вывода является регистр состояния SREG с адресом 005Fh.

Третий раздел представляет собой оперативное запоминающее устройство (ОЗУ) статического типа SRAM. Каждая ячейка содержит 8 бит, а объём памяти разный у разных микроконтроллеров. Адрес последней ячейки ОЗУ будем обозначать именем RAMEND.

В составе микроконтроллеров имеется также энергонезависимое постоянное запоминающее устройство EEPROM. Оно предназначено для хранения данных, записанных при программировании, а также данных, получаемых в процессе выполнения программы. Ёмкость EEPROM различна у разных микроконтроллеров. EEPROM имеет своё отдельное адресное пространство. Доступ к EEPROM осуществляется через порты ввода-вывода.

В качестве примера в данном пособии будут рассматривать-



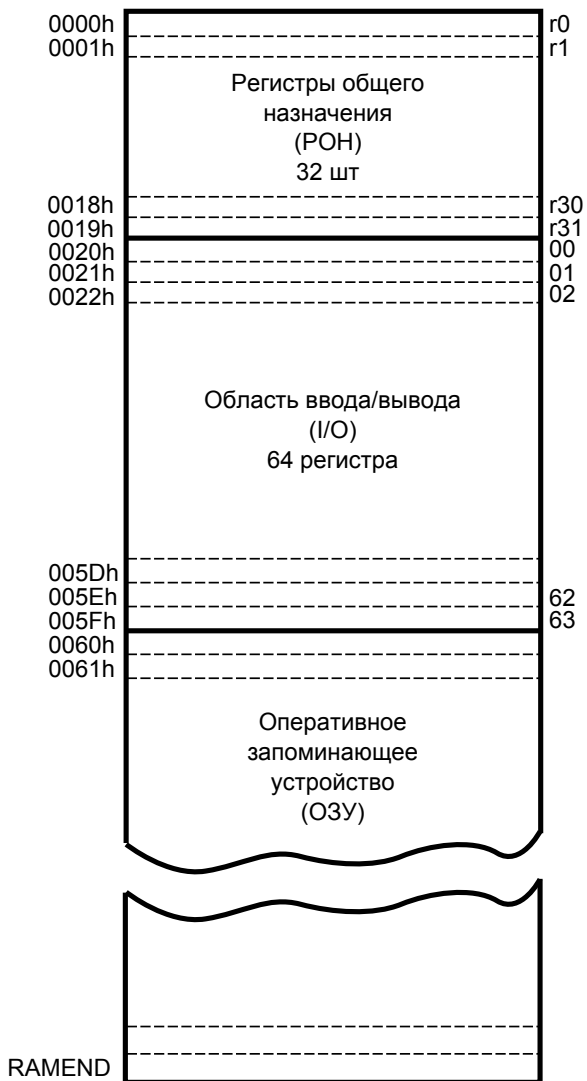


Рис. 4. Структура памяти данных

ся микроконтроллеры ATmega8. Ёмкость ОЗУ МК ATmega8 составляет 1024 байт, ёмкость EEPROM — 512 байт, ёмкость памяти программ — 8 Кбайт.

### Назначение выводов

Рассмотрим назначение выводов микросхемы ATmega8. На рис. 5 показана цоколёвка микросхемы. На контакты «VCC» и

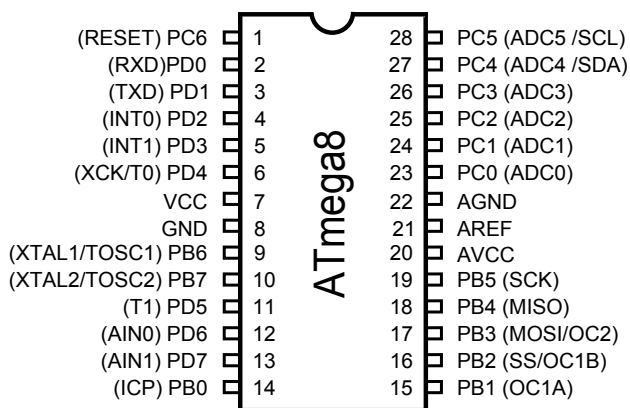


Рис. 5. Цоколёвка микросхемы ATmega8

«GND» подаются «+» и «-» питания (5 В) цифровой части микроконтроллера. Аналоговая часть микросхемы питается отдельными контактами «AVCC» и «AGND». К аналоговой части, например, относится встроенный в микроконтроллер АЦП. Разделение питания аналоговой и цифровой частей необходимо для того, чтобы уменьшить влияние импульсных помех цифровой схемы на функционирование аналоговой. К контакту «AREF» можно подключать выход источника внешнего опорного (образцового) напряжения, необходимого для работы АЦП. Остальные ножки микросхемы являются контактами портов ввода-вывода. В микроконтроллере ATmega8 реализованы порты: В (8 контактов PB0 – PB7), С (7 контактов PC0 – PC6) и D (8 контактов PD0 – PD7). Помимо

основного, контакты портов имеют одно или два вспомогательных назначения (на рис. 5 написаны в скобках). Перечислим наиболее важные из них:

- ADC0 – ADC5 — входы АЦП;
- SCK, MISO, MOSI, SS — выводы последовательного порта SPI, также используются для внутрисхемного программирования;
- INT0, INT1 — входы сигналов внешних прерываний;
- XTAL1, XTAL2 — контакты для подключения внешнего кварцевого резонатора или внешнего источника тактового сигнала;
- RXD, TXD — выводы последовательного порта UART для взаимодействия с другими контроллерами и COM портом компьютера;
- RESET — вывод для подключения кнопки сброса (если она необходима).

### **Внутрисхемное программирование**

Запись программы в память микроконтроллера осуществляется с помощью программатора. Для прошивки современных микроконтроллеров нет необходимости извлекать их из схемы. Программатор использует функции внутрисхемного программирования, взаимодействуя с микроконтроллером посредством SPI порта. Существует много программаторов, выпускаемых промышленностью. Они соединяются с программируемым микроконтроллером, с одной стороны, и с одним из интерфейсов персонального компьютера (ПК) — с другой. Программа на ПК отправляет прошивку программатору, который загружает её в память микроконтроллера.

Одним из возможных вариантов является подключение программируемого микроконтроллера непосредственно к LPT порту ПК, подключение также возможно через согласующие цепи. Тогда программа на ПК сама осуществляет загрузку программы в

память микроконтроллера через LTP интерфейс. Именно такой способ предусмотрен для прошивки микроконтроллеров в макетах, используемых на лабораторных работах. Разъём «LPT порт» лабораторного макета нужно соединить кабелем с LPT портом ПК. Для прошивки используется программа PonyProg. Её внешний вид изображён на рис. 6.

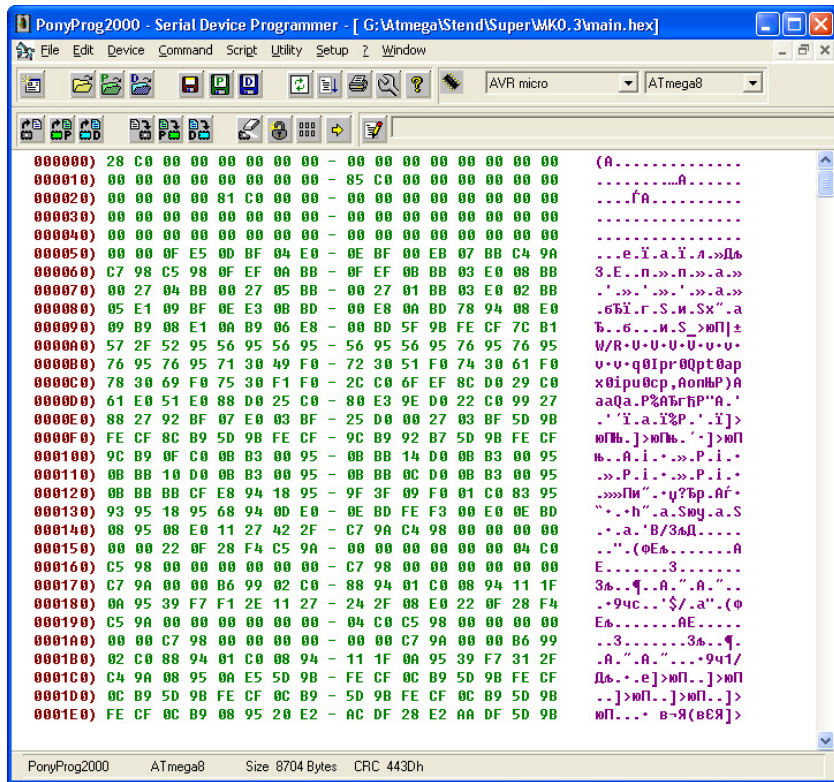






Рис. 6. Окно программатора PonyProg

Примерный алгоритм работы с программой заключается в следующем:

- выбираем модель программируемого микроконтроллера



- кнопкой  стираем имеющуюся в памяти микроконтроллера программу;
- открываем файл прошивки (кнопка );
- записываем программу в микроконтроллер (кнопка );

Прочитать программу из памяти контроллера можно с помощью кнопки .

# ЛАБОРАТОРНАЯ РАБОТА № 1. ОПЕРАЦИИ С ПАРАЛЛЕЛЬНЫМИ ПОРТАМИ ВВОДА-ВЫВОДА

## Параллельные порты ввода-вывода

Порты ввода-вывода В, С, D предназначены для связи микроконтроллера с «внешним миром». Биты портов подключены к ножкам микросхемы. Каждый из них может работать в режиме либо ввода, либо вывода. В микроконтроллерах AVR каждому порту поставлены в соответствие три восьмиразрядных регистра в области ввода-вывода:  $DDR_x$ ,  $PORT_x$ ,  $PIN_x$ , где  $x$  — буква В, С или D. Регистр  $DDR_x$  является конфигурационным и служит для настройки режима — ввод или вывод, — причём можно управлять каждым разрядом отдельно. Когда биты регистра  $DDR_x$  содержат логическую 1, то соответствующие биты порта являются выходами. Следовательно, чтобы настроить порт на вывод, нужно записать единицы в регистр  $DDR_x$ . При этом данные, содержащиеся в регистре  $PORT_x$ , будут выводиться на контакты микросхемы.

Если записать в регистр  $DDR_x$  нули, то соответствующие линии порта будут настроены как входные. Но их поведение зависит от содержимого регистра  $PORT_x$ . Если соответствующий бит порта  $PORT_x$  содержит 1, то к входной линии подключается внутренний подтягивающий резистор. Он соединяет линию порта с плюсом питания и обеспечивает на входной линии потенциал, близкий к логической единице. Когда бит порта  $PORT_x$  содержит 0, то входная линия находится в Z-состоянии. Её потенциал в этом случае задаёт устройство, к которому она подключена (например, другой микроконтроллер). Прочитать данные с входной линии можно из регистра  $PIN_x$ .

## Описание лабораторного макета

Лабораторная работа выполняется с использованием макета №1, блок-схема которого изображена на рис. 7.

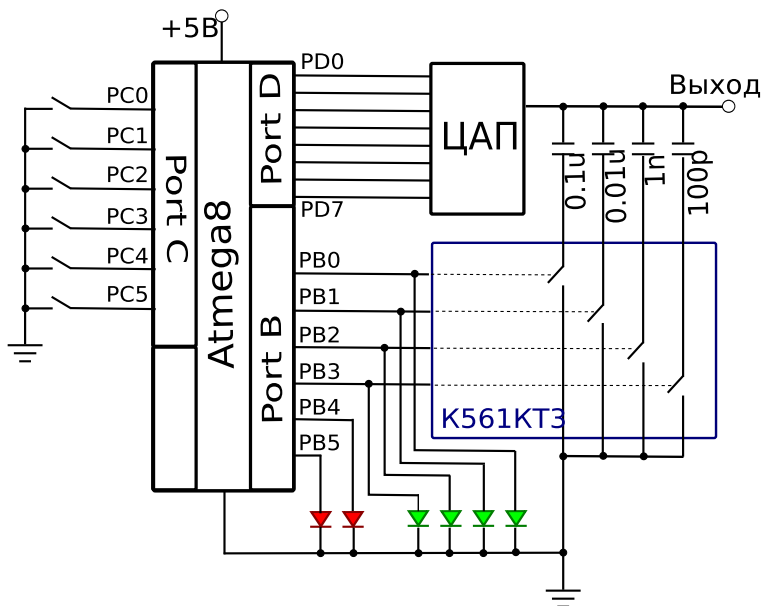


Рис. 7. Блок-схема лабораторного макета № 1

Главным элементом макета является микроконтроллер ATmega8. К линиям 0–5 порта С микроконтроллера подключены кнопки. Это позволяет запрограммировать реакцию устройства на внешние воздействия. В отпущенном состоянии кнопок на линиях порта поддерживается напряжение, близкое к +5 В (логическая 1). Это достигается включением «подтягивающих» резисторов номиналом 10 кОм между ножками порта и линией питания (при соответствующей настройке порта). В нажатом состоянии напряжение на линии порта равно 0 В, что соответствует логическому нулю. Считывание данных из порта С и последующий их анализ позволяет проанализировать состояние кнопок.

К линиям 0–3 порта В подключены зелёные, а к линиям 4 и 5 — красные светодиоды. Если сконфигурировать порт В как выходной, то при выводе в порт логических единиц на выводах микросхемы поддерживается близкое к +5 В напряжение. При

этом подключённый к линии порта светодиод горит. Если в порт выведен логический ноль, то напряжение на линии порта близко к 0 и светодиод погашен. Таким образом, управление горением светодиодов можно осуществить с помощью своевременного вывода в порт В двоичных данных.

Лабораторный макет питается от внешнего источника питания +5 В. Остальные элементы макета не используются в первой лабораторной работе и будут рассмотрены позже.

Внешний вид макета изображён на рис. 8, 9.

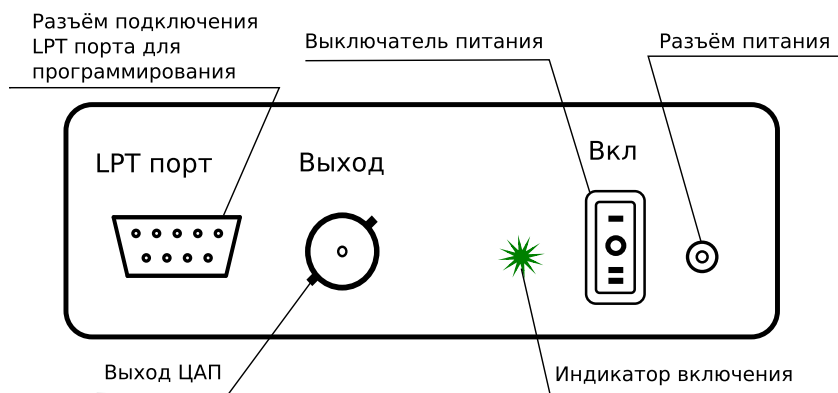


Рис. 8. Внешний вид лабораторного макета № 1 спереди

## Практические задания

Во всех вариантах заданий необходимо написать программу для микроконтроллера, которая реализует описанное в варианте поведение лабораторного макета. Работоспособность программы необходимо проверить с помощью эмулятора ATmega8 в программном комплексе AVR Studio. Отлаженную программу в виде бинарного файла прошивки необходимо загрузить в микроконтроллер с помощью программатора PonyProg.



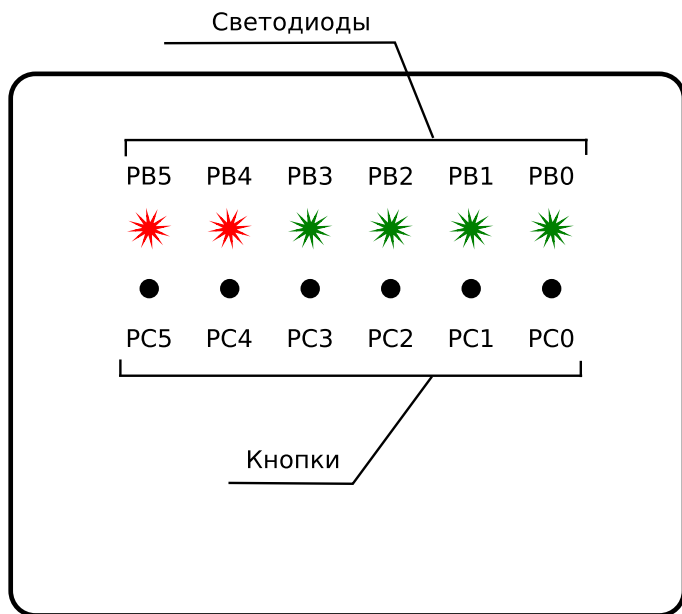


Рис. 9. Внешний вид лабораторного макета № 1 сверху

### Вариант 1.

Один из светодиодов макета мигает с частотой 1 Гц. Нажатием на одну кнопку можно остановить, а потом запустить мигание.

### Вариант 2.

Один из светодиодов макета мигает с некоторой частотой. Нажатием на одну кнопку можно изменять частоту мигания. Например, нажатие на кнопку приводит к удвоению частоты, повторное нажатие — возвращает частоту мигания в исходное состояние.

### Вариант 3.

Два светодиода на макете мигают попеременно с частотой 0,5 Гц (когда один гаснет, второй загорается). Нажатием на одну кнопку можно остановить, а потом запустить мигание.

*Вариант 4.*

Два светодиода на макете попеременно мигают с некоторой частотой (когда один гаснет, второй загорается). Нажатием на одну кнопку можно изменять частоту мигания. Например, нажатие на кнопку приводит к удвоению частоты, повторное нажатие — возвращает частоту мигания в исходное состояние.

*Вариант 5.*

Линейка светодиодов работает в режиме бегущего огня. Нажатием на кнопку можно остановить, а потом запустить бегущий огонь.

*Вариант 6.*

Линейка светодиодов работает в режиме бегущего огня. Нажатием на кнопку можно изменить скорость движения бегущего огня.

*Вариант 7.*

Три светодиода на макете работают в режиме светофора (в качестве жёлтого света можно выбрать любой светодиод).

*Вариант 8.*

Макет иллюстрирует работу счётчика. Светодиодами отображается двоичный код количества нажатий на одну из кнопок.

*Вариант 9.*

Макет иллюстрирует работу шифратора. Светодиодами отображается двоичный код номера нажатой кнопки.

*Вариант 10.*

Макет иллюстрирует работу дешифратора. Одновременное нажатие на несколько кнопок представляет собой номер нужного светодиода. Этот номер декодируется, и диод загорается.

## Пример выполнения работы

В качестве примера реализуем на лабораторном макете переключатель ёлочных гирлянд. Будем считать, что каждый светодиод представляет собой гирлянду. Реализуем два возможных режима работы: режим бегущего огня, режим псевдослучайного включения гирлянд. Кнопка PC0 будет запускать и останавливать работу макета, а кнопки PC1 и PC2 — осуществлять переключение режимов работы. Будем писать текст программы, сопровождая его пояснениями.

Программа начинается включением в текст программы файла описания микроконтроллера `m8def.inc`. Этот файл необходимо расположить в папке программы или указать к нему полный путь. Далее присваиваются символические имена `tmp`, `mode`, `outreg`, `vk1`, `Nrandom` регистрам `r16`, `r17`, `r19`, `r20`, `r21` соответственно.

```
.include "m8def.inc"
.def tmp = r16
.def mode = r17
.def outreg = r19
.def vk1 = r20
.def Nrandom = r21
```

Регистр `tmp` используется как временный, регистр `mode` содержит номер режима работы. Будем считать, что `mode = 1` соответствует режиму бегущего огня, а `mode = 2` — режиму псевдослучайного включения. Регистр `vk1` отвечает за состояние макета: включен (`vk1 = FFh`) или выключен (`vk1 = 00h`). Кнопка включения/выключения присоединена в контакту PC0 (бит 0 порта C). К контактам PC1, PC2 подключены кнопки смены режима: бегущий огонь и случайное включение. В программе для кнопок предусмотрены названия `pusk`, `beg`, `rand`.

```
.equ pusk = PC0
.equ beg = PC1
```

```
.equ rand = PC2
```

После директив начинается сама программа. Вначале располагается таблица векторов прерываний. Из всех прерываний используются два: прерывание начального старта **RESET** и прерывание переполнения таймера-счётчика **T1 change**. Остальные адреса заменены командой **nor**.

```
rjmp RESET  
nor  
nor  
nor  
nor  
nor  
nor  
nor  
nor  
rjmp change  
nor  
nor  
nor  
nor  
nor  
nor  
nor  
nor  
nor  
nor
```

Далее располагается подпрограмма обработки прерывания **RESET** или подпрограмма начального старта. Здесь конфигурируются стек и порты ввода-вывода

```
RESET:  
; конфигурируем стек  
ldi tmp, low(RAMEND)  
out SPL, tmp
```

```

ldi tmp, high(RAMEND)
out SPH, tmp
; настраиваем порты
clr tmp
out DDRC,tmp ; линии порта C - входы.
ser tmp
out DDRB,tmp ; линии порта B - выходы
out PORTC,tmp ; включение подтягивающих резисторов

```

Включение и выключение гирлянд будем осуществлять посредством вывода в порт В данных, заранее подготовленных в регистре `outreg`. Регистр `vk1` будем использовать как параметр состояния макета. При `vk1 = 00h` работа приостановлена, при `vk1 = FFh` — включена;

```

; включаем первую гирлянду
clr outreg
inc outreg
out PORTB,outreg
; разрешаем прерывание таймера
ldi tmp, (1<<TOIE1)
out TIMSK,tmp
; включаем режим работы 1
ldi mode,1
; разрешаем прерывания
sei ; I=1
; приостанавливаем работу макета
clr vk1 ;vk1 = 00h
; присваиваем начальное значение
; датчику случайных чисел
ldi Nrandom, 156
; конец подпрограммы начального старта

```

Основная часть программы начинается меткой `main` и представляет собой бесконечный цикл. В теле цикла происходит анализ

состояния кнопок. Если нажата кнопка `pusk`, происходит инверсия регистра `vk1`. При нажатии одной из кнопок `beg` или `rand` в регистр `mode` записывается значение 1 или 2. Далее происходит сравнение регистра `vk1` с нулём. Если работа макета приостановлена `vk1 = 00h`, то команда `breq main` обеспечивает возврат к метке `main` сразу после проверки нажатия кнопок

```
main:
; отслеживаем нажатия кнопок
sbis PINC, pusk
com vk1
sbis PINC, beg
ldi mode, 1
sbis PINC, rand
ldi mode, 2
cpi vk1, 0
breq main
```

Если работа макета включена `vk1 = FFh`, то после проверки состояния кнопок организуется задержка. Для этого запускается таймер T1 и происходит ожидание переполнения таймера. В процессе ожидания дополнительно проверяется состояние кнопки `pusk`.

```
; присваиваем начальное значение таймера
ldi tmp, $f8
out TCNT1H, tmp
ldi tmp, $5e
out TCNT1L, tmp
; запускаем таймер
ldi tmp, $04
out TCCR1B, tmp
; ожидание срабатывания таймера
clr r18 ; очищаем временный регистр
wait: ; пока r18 равен 0, ждём
    sbis PINC, pusk
```

```

        com vkl
        cpi r18,0
breq wait
; r18 перестанет быть равным 0 после обработки
; прерывания переполнения таймера
rjmp main ; конец главного цикла

```

Подпрограмма обработки прерывания переполнения таймера T1 начинается с метки `change`. Команда `ser r18` присваивает ненулевое значение регистру `r18`, что позволяет остановить ожидание в главном цикле программы. Дальнейшие действия зависят от режима работы. Если `mode = 2`, то начиная с метки `mode2` генерируется очередное псевдослучайное число `Nrandom` и выводится в порт В. Если `mode = 1`, то содержимое регистра `outreg` сдвигается влево для создания бегущего огня и тоже выводится в порт В.

```

change:
ser r18

cpi mode,2 ; если режим = 2
breq mode2 ; переходим на метку mode2

; иначе режим = 1
lsl outreg ; сдвиг влево
sbrc outreg,5 ; если дошли до 5-го бита
ldi outreg,1 ; перескакиваем на 1-й бит
rjmp endmode ; обход метки mode2

mode2:
mov tmp,N_random ; временно сохраняем Nrandom
lsr N_random ; умножаем на 2
lsr N_random ; умножаем на 4
add N_random,tmp ; умножаем на 5
inc N_random ; прибавили с = 1

```

```
mov outreg, N_random
```

```
endmode:
```

```
out PORTB, outreg ; выводим в порт
```

```
clr tmp
```

```
out TCCR1B, tmp ; останавливаем таймер T1
```

```
reti ; конец обработки прерывания
```



## ЛАБОРАТОРНАЯ РАБОТА № 2. ГЕНЕРАТОР СИГНАЛОВ

### Описание лабораторного макета

Лабораторная работа выполняется с использованием макета № 1, блок-схема которого изображена на рис. 7.

К линиям 0–7 порта D подключен цифроаналоговый преобразователь (ЦАП). ЦАП построен по простой резистивной схеме. На его вход с порта D через интервалы времени  $\Delta t$  подаются отсчёты сигнала в виде восьмибитных двоичных чисел (диапазон значений от 0 до 255). На выходе ЦАП каждый отсчёт представляется уровнем напряжения от 0 до +5 В, поддерживаемым в течение интервала  $\Delta t$ . Таким образом, выходной сигнал ЦАП представляет собой ступенчатую функцию с величиной ступеньки, кратной  $5/255 \approx 0.02$  В. Пример сигнала на выходе ЦАП изображён на рис. 10.

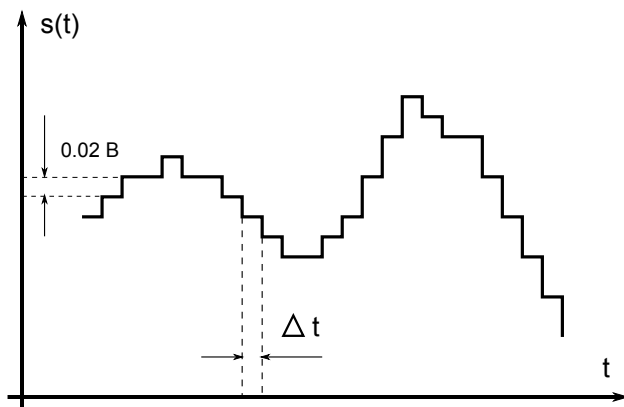


Рис. 10. Пример сигнала на выходе ЦАП

Для сглаживания ступенек в лабораторном макете используются конденсаторы, выполняющие роль фильтра низких частот. Предусмотрено подключение конденсаторов разных номиналов.

Оно осуществляется с помощью управляемых ключей, реализованных на микросхеме К561КТ3. Управляющие линии ключей соединены с линиями 0–3 порта В. Вывод логической единицы в нужный разряд порта В приводит к подключению соответствующего конденсатора.

### Прямой цифровой синтез

Для генерации сигнала воспользуемся методом прямого цифрового синтеза. Пусть необходимо сгенерировать сигнал, описываемый периодической функцией  $f(\varphi)$ , то есть  $\forall k \in Z$  справедливо равенство  $f(\varphi + k\Psi) = f(\varphi)$ . Величина  $\Psi$  — период функции  $f(\varphi)$ . В частном случае гармонического сигнала  $f(\varphi) = \cos \varphi$ , период  $\Psi = 2\pi$ .

Генерация сигнала методом прямого цифрового синтеза заключается в том, что через равные промежутки времени  $\Delta t$ , определяемые частотой дискретизации  $F = 1/\Delta t$ , на вход ЦАП поступают отсчёты (мгновенные значения) генерируемого сигнала. Эти отсчёты может вычислять программа микроконтроллера и выводить их в порт D. Но чаще всего ввиду малой мощности микроконтроллера для вычисления очередного отсчёта требуется время, большее, чем  $\Delta t$ . Поэтому отсчёты вычисляются заранее и хранят в памяти микроконтроллера в виде массива чисел. Поскольку функция  $f(\varphi)$  периодическая, имеет смысл сохранять отсчёты, взятые на одном периоде. Их количество зависит от объёма памяти микроконтроллера и выбирается, как правило, кратным степени двойки. Обозначим  $P = 2^p$  — количество отсчётов на периоде,  $f_i = f(i\Psi/P)$  — отсчёты. Для вывода отсчётов в ЦАП используется 8-битный порт D. Поэтому величины  $f_i$  будем хранить в 8-разрядных ячейках. Следовательно,  $f_i$  необходимо представить в виде целых чисел  $x_i$  из диапазона 0...255:

$$x_i = \text{trunc} \left\{ 255 \frac{f_i - \min(f_i)}{\max(f_i) - \min(f_i)} \right\},$$

где  $\text{trunc} \{ \cdot \}$  — целая часть числа.

Выводить поочерёдно отсчёты  $x_i$  можно следующим образом. Будем использовать переменную — счётчик  $i$ . Через интервал времени  $\Delta t$  счётчик  $i$  инкрементируется и отсчёт  $x_i$  выводится в порт. Величину  $i$  можно интерпретировать как нормированное целочисленное время  $i = t/\Delta t$ , которое измеряется в количестве отсчётов или интервалов дискретизации. Если разрядность переменной  $i$  равна  $p$ , то она пробегает значения от 0 до  $P - 1 = 2^p - 1$ . Достигнув максимально возможного значения, переменная  $i$  переполняется и становится равна нулю. Следовательно, за время  $P\Delta t$  успевает переполниться переменная  $i$  и выводится в порт полный период генерируемого сигнала. Значит частота сигнала на выходе ЦАП равна  $\nu_0 = F/P$ .

Пусть теперь необходимо сгенерировать сигнал с другой частотой. Обозначим  $s(t) = f(\omega t)$ , где величина  $\omega$  характеризует частоту генерируемого сигнала. Аргумент  $\phi = \omega t$  будем называть фазой сигнала. Она является монотонно растущей величиной. Прирост фазы за время  $\Delta t$  равен  $\Delta\phi = \omega\Delta t$ . Отсюда видно, что чем выше частота, тем больше прирост фазы за время одного интервала дискретизации.

Чтобы обеспечить генерацию сигнала с разной частотой, модифицируем алгоритм работы устройства. Во-первых, вместо переменной счётчика  $i$  с разрядностью  $p$  будем использовать накопительную переменную  $k$  с разрядностью  $n > p$ . Она может принимать значения от 0 до  $N - 1 = 2^n - 1$ . Поскольку  $N > P$ , переменная  $k$  не подходит для нумерации отсчётов  $x_i$ , хранимых в памяти. Нумеровать отсчёты будем с помощью  $p$  старших битов переменной  $k$ . Тогда для вывода в ЦАП полного периода хранящегося в памяти сигнала потребуется время  $N\Delta t > P\Delta t$ . Сигнал будет иметь частоту  $\nu = F/N$ . Например, если разрядность накопительной переменной равна 24 бит, а частота дискретизации  $F = 0.1$  МГц, то  $N = 16777216$ ,  $\nu = 0.00596$  Гц.

Во-вторых, разрешим увеличивать накопительную переменную более чем на единицу. Будем на каждом интервале дискретизации  $\Delta t$  прибавлять к переменной  $k$  целую величину  $\omega$ . Тогда  $k$

будет достигать переполнения в  $\omega$  раз быстрее, а значит частота сигнала увеличится в  $\omega$  раз и будет равна  $\nu = \omega F/N$ .

Величину  $k$  можно интерпретировать как нормированную на  $\Delta t$  фазу сигнала, и поэтому её называют аккумулятором фазы. Отношение  $F/N = \Delta F$  характеризует частотное разрешение, то есть показывает, насколько меняется частота сигнала при изменении  $\omega$  на единицу. Заметим также, что величина  $\omega$  по смыслу не является частотой в обычном понимании. Она показывает, во сколько раз частота генерируемого сигнала больше минимальной частоты  $\Delta F = F/N$ , которая в свою очередь определяется частотой дискретизации  $F$  и разрядностью аккумулятора фазы.

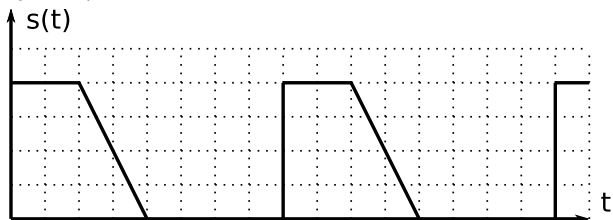
Осталось обсудить, чему равна и как выбирается частота дискретизации  $F$ . Поскольку мгновенные значения сигнала для вывода в ЦАП подготавливаются микроконтроллером, то частота  $F$  зависит от тактовой частоты  $F_{clk}$ , на которой работает микроконтроллер. На один цикл своей работы (инкремент аккумулятора фазы, извлечение отсчёта из памяти, вывод его в порт) он тратит  $m$  машинных тактов. При оптимально написанной программе  $m \approx 7...12$ . Следовательно частота, с которой отсчёты попадают в порт D, равна  $F = F_{clk}/m$ . Таким образом, при тактовой частоте микроконтроллера  $F_{clk} = 1$  МГц и  $m = 10$  частота дискретизации  $F = 0.1$  МГц.

### Практические задания

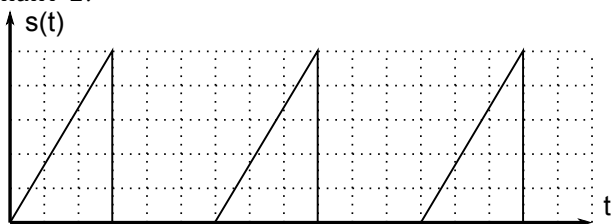
1. На лабораторном макете реализуйте функцию генератора периодического сигнала, форма и скважность которого определяется вашим вариантом. Выходной сигнал макета наблюдайте на экране осциллографа.
2. Исследуйте влияние ёмкости сглаживающего конденсатора на внешний вид выходного сигнала.
3. Выясните, с каким временным интервалом  $\Delta t$  микроконтроллер выдаёт отсчёты сигнала.

4. Исследуйте, какой максимальной частоты можно добиться для вашей формы сигнала.

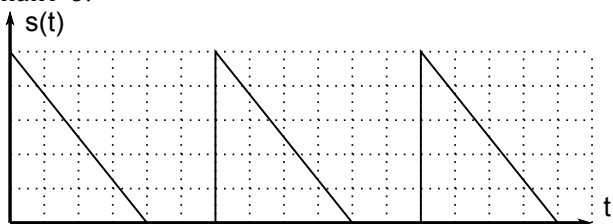
Вариант 1.



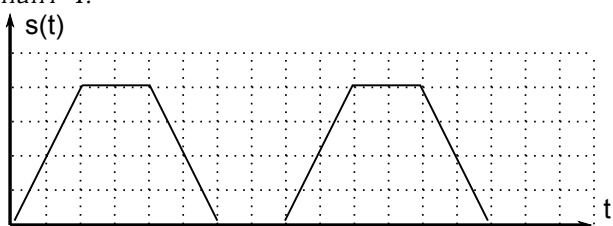
Вариант 2.



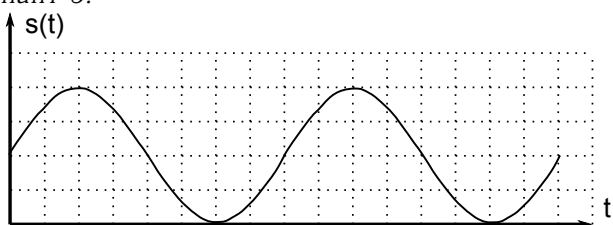
Вариант 3.



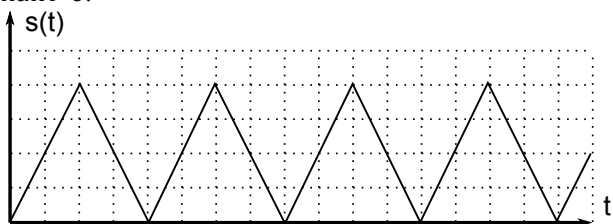
Вариант 4.



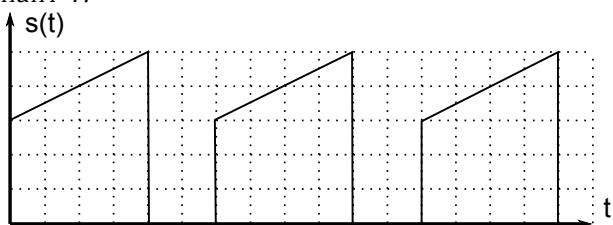
Вариант 5.



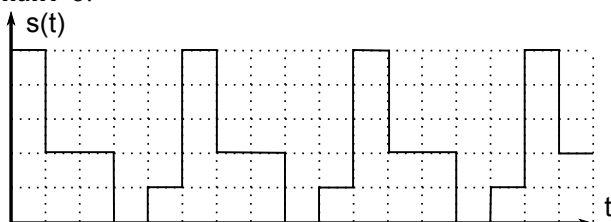
Вариант 6.



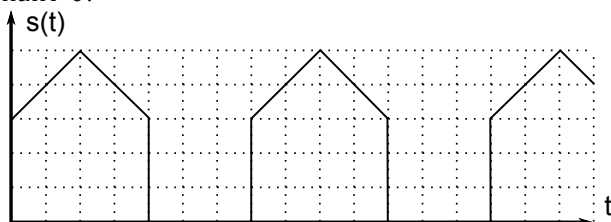
Вариант 7.



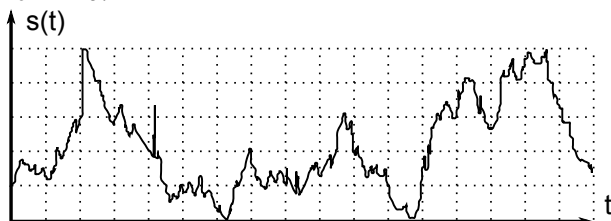
Вариант 8.



Вариант 9.



Вариант 10.



### Пример выполнения работы

В качестве примера реализуем генератор сигнала, изображённого на рис. 11. Этот сигнал представляет собой сумму первой и третьей гармоник с единичными амплитудами

$$f(\varphi) = \sin(\varphi) + \sin(3\varphi) + 1,6.$$

Величина 1,6 подобрана экспериментально, чтобы график сигнала находился полностью над осью абсцисс. Максимальное значение сигнала  $\max f(\varphi) \approx 3,2$ .

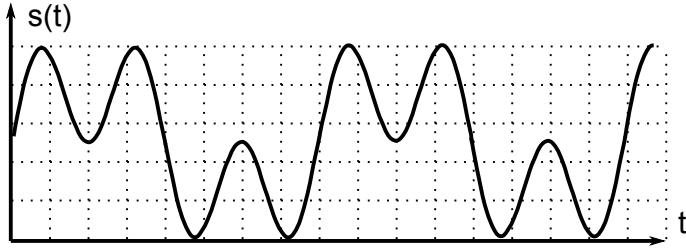


Рис. 11. Сумма первой и третьей гармоник

Период функции  $f(\varphi)$  равен  $\Psi = 2\pi$ . Вычислим отсчёты сигнала на одном периоде и представим их целыми числами из диапазона от 0 до 255:

$$x_i = \text{trunc} \left\{ \frac{255}{3.2} \left[ \sin \left( \frac{2\pi i}{P} \right) + \sin \left( 3 \frac{2\pi i}{P} \right) + 1.6 \right] \right\}.$$

Будем использовать 256 отсчётов на периоде, так что  $p = 8$ ,  $P = 256$ . Один период сигнала как функция номера отсчёта  $i$  показан на рис. 12. Массив отсчетов можно получить в программе Maxima с помощью одной команды:

```
makelist(round(255*(sin(2*%pi*i/256)+
sin(3*2*%pi*i/P)+1.6)/3.2),i,0,255);
```

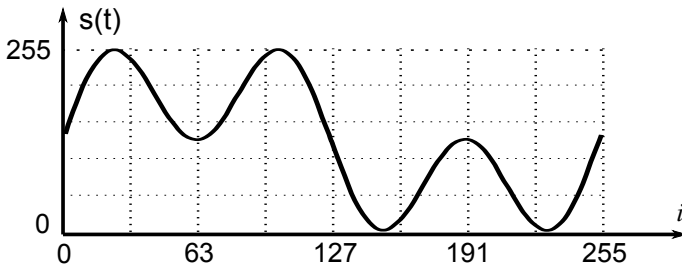


Рис. 12. Один период сигнала



В результате получаем массив из 256 чисел:

128, 135, 143, 151, 158, 166, 173, 180,  
187, 194, 200, 206, 212, 218, 223, 227,  
232, 235, 239, 242, 244, 246, 248, 249,  
250, 250, 250, 249, 248, 247, 245, 243,  
240, 237, 234, 230, 227, 223, 218, 214,  
209, 205, 200, 195, 190, 185, 180, 175,  
171, 166, 162, 157, 153, 149, 146, 142,  
139, 137, 134, 132, 131, 129, 128, 128,  
128, 128, 128, 129, 131, 132, 134, 137,  
139, 142, 146, 149, 153, 157, 162, 166,  
171, 175, 180, 185, 190, 195, 200, 205,  
209, 214, 218, 223, 227, 230, 234, 237,  
240, 243, 245, 247, 248, 249, 250, 250,  
250, 249, 248, 246, 244, 242, 239, 235,  
232, 227, 223, 218, 212, 206, 200, 194,  
187, 180, 173, 166, 158, 151, 143, 135,  
128, 120, 112, 104, 97, 89, 82, 75,  
68, 61, 55, 49, 43, 37, 32, 28,  
23, 20, 16, 13, 11, 9, 7, 6,  
5, 5, 5, 6, 7, 8, 10, 12,  
15, 18, 21, 25, 28, 32, 37, 41,  
46, 50, 55, 60, 65, 70, 75, 80,  
84, 89, 93, 98, 102, 106, 109, 113,  
116, 118, 121, 123, 124, 126, 127, 127,  
128, 127, 127, 126, 124, 123, 121, 118,  
116, 113, 109, 106, 102, 98, 93, 89,  
84, 80, 75, 70, 65, 60, 55, 50,  
46, 41, 37, 32, 28, 25, 21, 18,  
15, 12, 10, 8, 7, 6, 5, 5,  
5, 6, 7, 9, 11, 13, 16, 20,  
23, 28, 32, 37, 43, 49, 55, 61,  
68, 75, 82, 89, 97, 104, 112, 120

Будем далее писать текст программы, сопровождая его пояснениями. Программа начинается включением файла с описанием микроконтроллера `m8def.inc` в текст программы. Этот файл необходимо расположить в папке программы или указать к нему полный путь. Далее присваивается символическое имя `tmp` регистру `r16`.

```
.include "m8def.inc"  
.def tmp = r16
```

Хранить вычисленные ранее отсчёты сигнала будем в памяти программ. Поэтому директивой `.CSEG` подчеркнём, что начался сегмент кода программы. Далее конфигурируем порты ввода-вывода

```
.CSEG  
clr tmp  
out DDRC,tmp ; линии порта C - входы  
ser tmp  
out DDRD,tmp ; линии порта D - выходы  
out DDRB,tmp ; линии порта B - выходы  
ldi tmp, 0b00001111  
out PORTB, tmp ; включили конденсаторы
```

Сами отсчёты будем располагать после всей программы и пометим меткой `signal`. Она представляет собой адрес первого отсчёта сигнала. Доступ к данным, хранящимся в памяти программ, осуществляется с помощью команды `lpm`. При этом используется косвенная адресация через регистровую пару `Z`, в которую входят два регистра `r31:r30`. Команда `lpm` извлекает байт данных из ячейки памяти программ, адрес которой указан в регистровой паре `Z`. При этом следует учитывать особенности адресации памяти программ. Она состоит из шестнадцатиразрядных слов. Поэтому метка `signal` содержит адрес первого отсчёта сигнала, выраженный в словах. Команда же `lpm` использует адресацию каждого байта этой же памяти. Следовательно, для вычисления

адреса первого байта нужно адрес `signal`, выраженный в словах, умножить на 2. В следующем фрагменте программы старший и младший байты адреса `signal` загружаем во вспомогательные регистры `r18` и `r17`. Значение регистровой пары `r18:r17` умножаем на 2 методом сдвига влево на 1 бит. Получаем адрес первого отсчёта сигнала, выраженный в байтах, и загружаем его в регистровую пару `Z`.

```
ldi r18, high(signal)
ldi r17, low(signal)
clc      ; очистка бита C регистра состояния SREG
rol r17  ; циклический сдвиг влево на 1 бит, через
rol r18  ; бит C регистра SREG (умножение на 2)
mov r31, r18      ; загрузка адреса в
mov r30, r17      ; регистровую пару Z
```

В результате микроконтроллер готов к считыванию первого отсчёта сигнала из памяти программ. Будем использовать 24-разрядный аккумулятор фазы. Его старший, средний и младший байты расположим в регистрах `r30:r29:r28` соответственно. Старший байт `r30` аккумулятора фазы определяет номер отсчёта сигнала, который необходимо выводить в ЦАП. Именно поэтому он совпадает с младшим байтом регистровой пары `Z`.

Рассчитаем теперь частотное разрешение. В лабораторном макете используется тактовая частота микроконтроллера  $F_{clk} \approx 1$  МГц. Как будет видно ниже, в основном цикле программы число тактов  $m = 9$ . Следовательно, частота дискретизации  $F \approx 111$  кГц. При 24 разрядах аккумулятора фазы  $N = 16777216$ , а частотное разрешение  $\Delta F = 0.00661611557$  Гц. Пусть необходимо генерировать сигнал с частотой 1 кГц. Разделим требуемую частоту на частотное разрешение, получим значение переменной  $\omega = \text{trunk}\{1000/0,00661611557\} = 151146 = 024E6Ah$ .

Переменную  $\omega$  в программе расположим в трёх регистрах `r26:r25:r24` — старший, средний и младший байты соответственно. Следующий фрагмент присваивает этим байтам вычисленное значение `024E6Ah`.

```
ldi r24,0x6A
ldi r25,0x4E
ldi r26,0x02
```

Далее следует основной цикл программы, в котором к аккумулятору фазы прибавляется величина  $\omega$ , извлекается отсчёт сигнала и выводится в порт. Эти действия занимают 9 тактов микроконтроллера.

```
main:
    ; сложение
    add r28,r24    ; 1 такт
    adc r29,r25    ; 1 такт
    adc r30,r26    ; 1 такт
    ; извлечение
    lpm           ; 3 такта
    ; вывод в порт
    out PORTD,r0  ; 1 такт
    rjmp main     ; 2 такта
```

И, наконец, после всей программы располагается массив отсчётов генерируемого сигнала.

```
signal:
    .db 128,135,143,151,158,166,173,180
    .db 187,194,200,206,212,218,223,227
    .db 232,235,239,242,244,246,248,249
    .db 250,250,250,249,248,247,245,243
    .db 240,237,234,230,227,223,218,214
    .db 209,205,200,195,190,185,180,175
    .db 171,166,162,157,153,149,146,142
    .db 139,137,134,132,131,129,128,128
    .db 128,128,128,129,131,132,134,137
    .db 139,142,146,149,153,157,162,166
    .db 171,175,180,185,190,195,200,205
    .db 209,214,218,223,227,230,234,237
```

.db 240,243,245,247,248,249,250,250  
.db 250,249,248,246,244,242,239,235  
.db 232,227,223,218,212,206,200,194  
.db 187,180,173,166,158,151,143,135  
.db 128,120,112,104,97,89,82,75  
.db 68,61,55,49,43,37,32,28  
.db 23,20,16,13,11,9,7,6  
.db 5,5,5,6,7,8,10,12  
.db 15,18,21,25,28,32,37,41  
.db 46,50,55,60,65,70,75,80  
.db 84,89,93,98,102,106,109,113  
.db 116,118,121,123,124,126,127,127  
.db 128,127,127,126,124,123,121,118  
.db 116,113,109,106,102,98,93,89  
.db 84,80,75,70,65,60,55,50  
.db 46,41,37,32,28,25,21,18  
.db 15,12,10,8,7,6,5,5  
.db 5,6,7,9,11,13,16,20  
.db 23,28,32,37,43,49,55,61  
.db 68,75,82,89,97,104,112,120

Директива DB резервирует необходимое количество байтов в памяти программ. Параметры, передаваемые директиве, — это последовательность выражений, разделённых запятыми. Каждое выражение должно или быть числом в диапазоне 0...255, или давать при вычислении результат в этом же диапазоне.

## ЛИТЕРАТУРА

1. Баранов В.Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы / В.Н. Баранов. — М. : Додэка-XXI, 2004. — 288 с.
2. Белов А.В. Создаем устройства на микроконтроллерах / А.В. Белов — СПб. : Наука и Техника, 2007. — 304 с.
3. Гребнев В.В. Микроконтроллеры семейства AVR фирмы Atmel / В.В. Гребнев. — М. : РадиоСофт, 2002. — 176 с.
4. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы «ATMEL» / А.В. Евстифеев. — М. : Додэка-XXI, 2004. — 560 с.
5. Трамперт В. AVR-RISC микроконтроллеры : пер. с нем. / В. Трамперт. — Киев : МК-Пресс, 2006. — 464 с.