МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
И УПРАВЛЕНИЯ В ТЕХНИЧЕСКИХ СИСТЕМАХ

Кафедра информатики и управления в технических системах

ТЕХНОЛОГИИ И СРЕДСТВА ПРОГРАММИРОВАНИЯ РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

Методические указания
к выполнению лабораторных работ
для студентов направления подготовки
27.03.04 – Управление в технических системах

Севастополь
СевГУ
2020

УДК 681.5
Т38

Р е ц е н з е н т:
В.В. Альчаков - канд. техн. наук, доцент

Составитель: И.Ю. Липко

Т38 Технологии и средства программирования робототехнических систем : методические указания к выполнению лабораторных работ для студентов направления подготовки 27.03.04 – «Управление в технических системах» / Сост. И.Ю. ; Министерство науки и высшего образования РФ, Севастопольский государственный университет. – Севастополь: СевГУ, 2020. – 31 с. – Текст : электронный.

Целью лабораторного практикума является освоение студентами средств программирования и моделирования робототехнических систем, основанных на микропроцессоре Atmega32. Основными рассматриваемыми инструментами являются программы Atmel Studio и Proteus.

Лабораторный практикум предназначен для студентов по направлению подготовки 27.03.04 – «Управление в технических системах».

УДК 681.5

Рассмотрены и рекомендованы к изданию на заседании кафедры «Информатика и управление в технических системах», протокол № 6 от 15 сентября 2020 г.

Содержание

Введение

Лабораторный практикум включает 3 лабораторные работы, охватывающие основные темы, связанные с программированием робототехнических систем, построенных на основе микроконтроллера Atmega32. Каждая лабораторная работа выполняется в течение шести академических часов. В процессе подготовки к выполнению работы студент должен ознакомиться с основными теоретическими положениями, описанием используемых программных инструментов.

Все лабораторные работы выполняются с использованием среды разработки Atmel Studio и пакета программ для автоматизированного проектирования электронных схем Proteus Design Suite. Программы позволяют программировать микроконтроллеры, моделировать аналоговые, цифровые и цифро-аналоговые схемы большой степени сложности. В программах имеются библиотеки, включающие набор типовых электрических и электронных элементов. Параметры элементов можно изменять в широком диапазоне значений. Достаточный набор приборов позволяет производить измерения различных величин, задавать входные воздействия, строить графики. Одним из достоинств программы является вид измерительных прибором, приближенный к реальному. Результаты моделирования можно вывести на печать или импортировать в текстовый или графический редактор.

Отлаженные таким образом программы и схемы собираются с помощью комплекта лабораторного стенда «Микроконтроллер Atmega32» и «Датчики».

1. Лабораторная работа № 1. Порты ввода/вывода

1.1 Цель работы

Изучение принципов программирования портов ввода-вывода микроконтроллера Atmega32 посредством создания программ включения и выключения светодиода с помощью кнопки и моделирования в среде Proteus Design Suite.

1.2 Краткие теоретические сведения

Что такое микроконтроллер. Микроконтроллер (МК) – это электронно-вычислительная машина (ЭВМ). Наиболее известные ЭВМ называют компьютерами (от англ. compute – вычислять, рассчитывать). Конечно, микроконтроллер похож на привычный персональный ЭВМ (ПЭВМ) (в англ. personal computer, PC это персональный компьютер, ПК), но его мощности по сравнению с современными ПК гораздо меньше. Он также обладает ОЗУ (оперативная память) и ПЗУ (энергонезависимая память, FLASH-память) и процессором, содержит внутри себя периферийные устройства: АЦП, компараторы, таймеры, счётчики и др. [1].

В ПЭВМ чаще всего применяется Принстонская архитектура памяти – совместное хранение команд и данных в одной памяти. В микроконтроллерах чаще всего используется Гарвардская архитектура, в которой есть хранилище команд и хранилище данных. Обычно в ПЗУ хранятся команды (программа), а в ОЗУ данные [3].

Другая особенность заключается в том, что в микроконтроллерах используется т.н. сокращенный набор инструкций (команды ассемблера) (англ. Reduced Instruction Set Computer, RISC), в то время как в ПЭВМ сложный набор команд (англ. Complex instruction Set computer, CISC). Каждый из наборов инструкций обладает своими положительными и отрицательными свойствами, требованиями к аппаратной реализации.

На рисунке 1.1 изображены виды корпусов МК Atmega32 – PDIP (слева) и TQFP (посередине) по сравнению с корпусом процессора Intel Core i3. Оба варианты используются в промышленности. Первый чаще встречается в учебных конструкциях или самоделках, поскольку их проще распаять на плате.

Рисунок 1.1 – Корпусы микропроцессоров Atmega32 и Intel Core i3

Порты и регистры микроконтроллера. МК содержит определенное количество выводов (контакты, «ножки», пины). В МК нет консоли и монитор подключить нельзя, как при «обычном» программировании: в*заимодействие МК с внешними устройствами происходит только посредством посыла и приёма сигналов на выводы*.

Каждый контакт может работать в режимах ввода или вывода данных. Под выводом данных подразумевается подача сигнала высокого или низкого логического уровня. Помимо этого, большинство контактов МК могут выполнять и специальные функции периферийных устройств.

Группа контактов логически объединена в порты ввода/вывода. Порт – это программный интерфейс и логическое соединение, соответствующее физическим контактам МК. Работа с контактами не похожа на работу с переменными как при «обычном» программиро-

вании. Например, для считывания и записи значения переменной используется имя этой переменной, а для выполнения тех же операций над контактом используется три регистра.

Регистры это грубо говоря глобальные переменные, информацию из которых мы можем как считать, так и изменить. В регистрах микроконтроллера хранятся “настройки” его состояния и периферии: таймеры-счётчики, порты с пинами, АЦП, шины UART, I2C, SPI и прочие устройства, встроенные в МК. Изменяя значения в регистрах, мы даём МК команду что нужно сделать. Названия (имена) регистров фиксированные, они “заняты”, и создать переменную, совпадающую с ними нельзя. Названия у регистров обычно являются мнемоническими аббревиатурами. Их полное описание находится в документации к микроконтроллеру (англ. datasheet, документация, даташит) [2].

Под каждый порт в адресном пространстве МК зарезервировано по 3 адреса, по которым размещены следующие регистры:

– регистр данных порта (англ. port data register) *PORTxn*,

– регистр направления данных (англ. port Data Direction Register) *DDRxn*,

– регистр ввода порта (англ. Port INput) *PINxn*.

Действительные названия регистров получаются подстановкой названия порта вместо символа «x». Соответственно регистры порта A называются PORTA, DDRA, PINA, порта B – PORTB, DDRB, PINB и т. д. Порты 8-битные. Нумерация идёт с 0 по 7 Символу «n» соответствует номер бита. Например:

PORTA0 – нулевой бит порта A регистра данных,

DDRA3 – третий бит порта A регистра направления данных,

DDRB7 – седьмой бит порта B регистра направления данных,

PINC6 – шестой бит порта C регистра ввода.

Регистр PINxn доступен только для чтения. С его помощью осуществляется доступ к физическим значениям сигналов на выводах порта. Остальные два регистра PORTxn и DDRxn доступны и для чтения, и для записи (табл. 1.1.).

Разряд DDRxn регистра DDRx определяет направление передачи данных через контакт ввода/вывода. Если этот разряд установлен в «1», то n-й вывод порта *x* является выходом, если же сброшен в «0» — входом.

Разряд PORTxn регистра PORTx выполняет двойную функцию. Если порт функционирует как выход (DDRxn = «1»), этот разряд определяет состояние вывода порта. Если разряд установлен в «1», на выводе устанавливается напряжение ВЫСОКОГО уровня. Если разряд сброшен в «0», на выводе устанавливается напряжение НИЗКОГО уровня.

Если же вывод функционирует как вход (DDRxn = «0»), разряд PORTxn определяет состояние внутреннего подтягивающего резистора для данного вывода. При установке разряда PORTxn в «1» подтягивающий резистор подключается между выводом микроконтроллера и проводом питания.

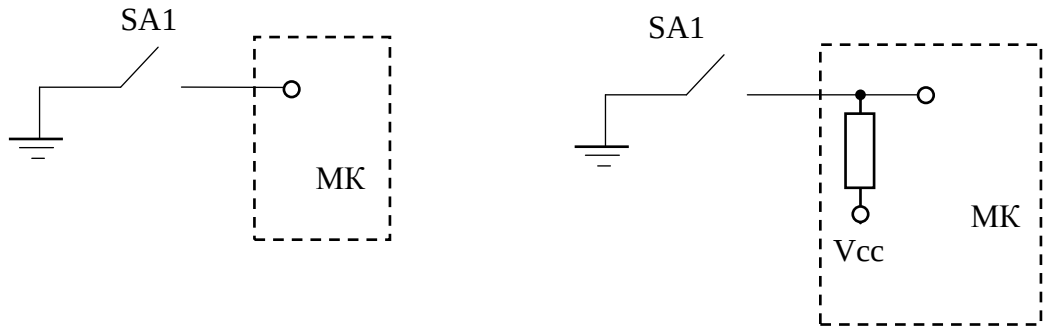
Таблица 1.1 – Настройка режима работы пинов

DDR	**PORT**	**Режим**	**Pullup**	**Комментарий**
0	0	Ввод	Нет	
0	1	Ввод	Да	Включение внутреннего подтягивающего резистора
1	0	Вывод	Нет	Логический «0»
1	1	Вывод	Нет	Логическая «1»

Состояние вывода микроконтроллера (независимо от установок разряда DDRxn) может быть получено путем чтения разряда PINxn регистра PINx. При этом следует помнить, что между действительным изменением сигнала на выводе и изменением разряда PINxn существует задержка. Эта величина задержки может составлять от 0,5 до 1,5 периода системного тактового сигнала. Поэтому после изменения значений портов следует выполнять пустую команду (оператор «;»).

Детальное описание портов и регистров находится в документации [2] в разделах: Register summary, I/O Registers и др.

Подтягивающий резистор. Применение подтягивающих резисторов позволяет снять логическую неопределённость при работе с выключателями. Рассмотрим ситуацию с кнопкой SA1 без подтягивающего резистора (рис. 1.2а). У неё есть два состояния: включено и выключено. Когда она включена, её контакт соединён с землёй, а значит на выводе будет логический «0». Если кнопка выключена, то состояние на выводе МК будет неопределенным: там не подключено ни питание, ни земля; вывод «висит в воздухе».



а) б)

Рисунок 1.2 – Схема без Pullup-резистора (а) и с Pullup-резистором (б)

В случае, когда в схеме есть подтягивающий резистор (рис.1.2б), логика будет следующей. Когда кнопка включена, её контакт соединён с землёй, а значит на выводе будет логический «0». Если кнопка выключена, то состояние на выводе МК будет логическая «1», потому что подключено питание Vcc.

В МК Atmega32 на каждом выводе предусмотрен внутренний подтягивающий резистор, который можно включить установкой соответствующих разрядов регистров PORT и DDR.

Общие сведения о языке C++. Язык C++ (и его предшественник C) называется языком высокого уровня. Основная черта высокоуровневых языков — это абстракция относительно реализации «в железе». Например, использование переменных (именованных областей памяти), вместо регистров и ячеек; введение структур данных и операций над ними легко интерпретируемых человеком, написание которых на ассемблере было бы довольно сложным. Язык C был разработан как замена языкам низкого уровня – ассемблерам, которые интерпретируются «железом», но плохо интерпретируются человеком. Затем язык C++ стал усовершенствованием C путём включения других языковых конструкций и более сложных абстракций относительно «железа» микроконтроллеров и компьютеров.

Минимальная программа для МК выглядит следующим образом:

```
#include <avr/io.h>

int main(void)
{ // начало программы
    while (1) // цикл
    {
    }
}
```

С точки зрения программиста эта программа делает ничего: выполняется бесконечный цикл. По факту МК выполняет следующую программу (дизассемблированный фрагмент):

```
   // начало программы: {
00000036  PUSH R28       Push register on stack
```

```
00000037  PUSH R29        Push register on stack
00000038  IN R28,0x3D     In from I/O location
00000039  IN R29,0x3E     In from I/O location
   // цикл:  while (1)
0000003A  RJMP PC-0x0000      Relative jump
```

МК выполняет «пустые» действия: переходит на текущий адрес (команда RJMP, от англ. Relative JuMP), он не бездействует.

Ряд отличий при работе с МК от «обычного» программирования:

– в МК нет стандартного устройства ввода/вывода, поэтому библиотека iostream и подобные не используются для вывода в консоль;

– необходимо подключение специальной библиотеки, которая предоставит интерфейс для работы с МК;

– МК необходимо настраивать;

– в Arduino IDE т.н. “ардуиновские” функции обладают читаемым названием, но на самом деле также работают с регистрами и портами.

Инструменты для работы. В качестве инструментов разработки используются: среда программирования Atmel Studio 7, среда симулирования Proteus Design Suite 8. Программы рекомендуется запускать с правами администраторов. Тогда в Atmel Studio можно избежать проблем с настройками симулятора, а в Proteus с доступом к электронным компонентам. В настройках оптимизации Atmel Studio рекомендуется задать параметры компилятора и сборщика «без оптимизации кода» (пункт Optimization Level значение «None (-O0)»).

1.3 Практические примеры

Пример 1. Включение диода. В данном примере рассмотрим пример настройки МК для программного включения и выключения светодиода.

Собираем схему как на рис. 1.2 из элементов: микроконтроллер Atmega32, 3 диода (компоненты Led), резистор (компонент Res), кнопка (компонент Button), земля (компонент Ground).

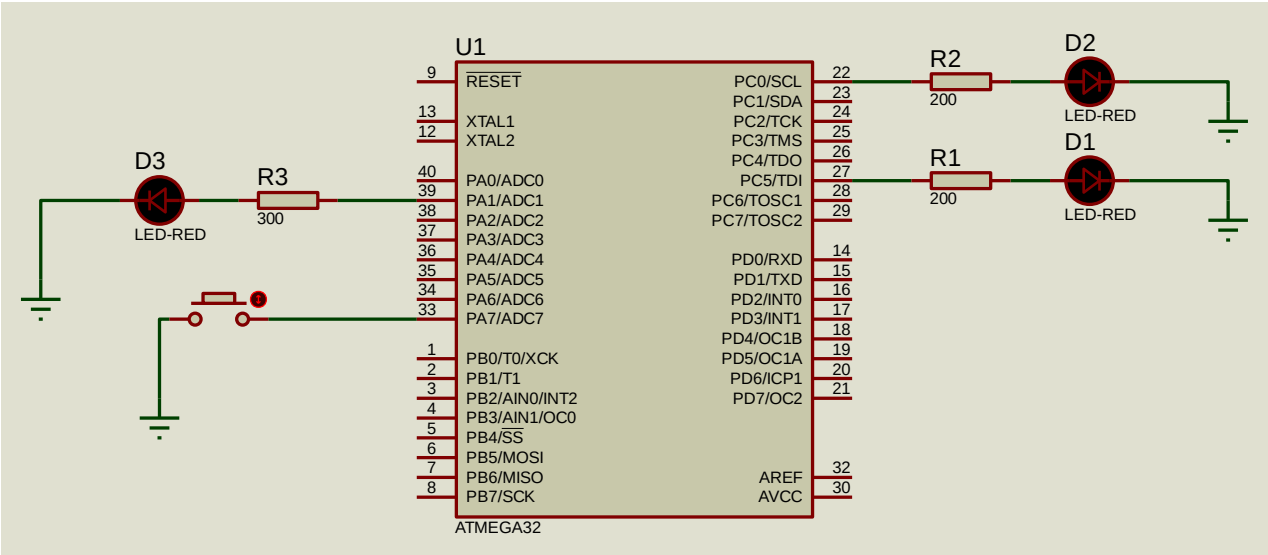


Рисунок 1.2 – Схема для примера с диодами и кнопкой

При включении МК, значение регистров PORTx = 0x00. Для выводов, настроенных как «выход», можно программно управлять его состоянием: если в разряде регистра PORTx записана «1», то на соответствующем ему выводе МК будет высокий потенциал (Vcc), а если «0» – низкий (GND).

В коде ниже назначаем выводу № 5 порта С режим вывода. На его выходе подаём логическую «1».

```
#include <avr/io.h>

int main(void)
{
    DDRC = 0b00100000;  // назначаем направление работы пина
    PORTC = 0b00100000;  // подаём логическую единицу на пин

    while (1)
    {
    }
}
```

После компиляции и настройки МК запускаем симуляцию. Должен засветиться светодиод D1.

Для того, чтобы включить два светодиода D1 и D2 порта C необходимо изменить параметры порта

```
    DDRC = 0b00100001;  // назначаем направление работы 0-го и 5-го пинов
    PORTC = 0b00100001;  // подаём логическую единицу на 0-й и 5-й пин
```

Это будет означать что выводы PORTC0 и PORTC5 будут работать в режиме вывода.

Пример 2. Мигание диодом. В данном примере используются два диода на одном порту. При этом один включён постоянно, а другой мигает. Это реализовано с помощью битовых операций над регистром:

```
    PORTC = 0b00000001;
    PORTC |= 1<<5;
```

Здесь задано начальное значение, а затем с помощью битового сложения (позволяет не изменять значение других разрядов) и сдвига устанавливается значение «1» в пятый разряд.

Затем в цикле изменяется значения разряда мигающего диода. Установка нуля и единицы в 5-й разряд делается так, чтобы не изменить значения других разрядов:

```
      PORTC &= 0b11011111;
      ...
      PORTC |= 0b00100000;
```

Для мигания диодом используется функция *_delay_ms* программной задержки из библиотеки *util/delay.h*. Для правильной работы задержки необходимо задать параметр F_CPU равный частоте МК. Полный код программы:

```
#include <avr/io.h>

#define F_CPU 16000000UL  // 16MHz - частота МК
#include <util/delay.h>

int main(void) {
   DDRC  = 1<<5 | 1<<0;
   PORTC = 0b00000001;
   PORTC |= 1<<5;

   while(1) {
      PORTC &= 0b11011111;
      _delay_ms(100);

      PORTC |= 0b00100000;
      _delay_ms(100);
   }
}
```

Пример 3. Включение диода по нажатию кнопки. В этом примере для включения светодиода задействуется кнопка. Для корректной работы кнопки следует применять подтягивающий резистор:

```
    DDRA = 0b00000000;   // 7й вывод в режиме входа
    PORTA = 0b10000000;   // включаем подтягивающий резистор на 7 вывода
```

Считывание значения с вывода регистра производится следующим образом:

```
      if (!(PINA & 0b10000000))  {
         PORTC |= 0b00000001;
      } else {
         PORTC &= 0b11111110;
      }
```

Конструкция PINA & 0b10000000 называется «маскированием». Благодаря значению 1 в 7-м разряде, в условии учитывается только 7-й разряд регистра PINA.

Полный код программы:

```
#include <avr/io.h>
#define F_CPU 16000000UL //16MHz
#include <util/delay.h>

int main(void)
{
   // инициализация кнопки
   DDRA = 0b00000000;   // 7й вывод в режиме входа
   PORTA = 0b10000000;   // включаем подтягивающий резистор на 7 вывода

   // инициализация диодов
   DDRC = 0b00100001;  // направление работы
   PORTC = 0b00100000;  // 5й вывод лог.1

   while (1)
   {
      if (!(PINA & 0b10000000)) {
         PORTC |= 0b00000001;
      } else {
         PORTC &= 0b11111110;
      }
   }
}
```

1.4 Контрольные вопросы

1. Как называется документация для ATmega32?
2. Поясните, что такое порт, регистр, микроконтроллера.
3. Поясните, что такое PORTA1, DDRB4, PINB4, PORTC, DDRB.
4. Что такое подтягивающий резистор? Поясните его работу.
5. Поясните команды DDRC = 0b00100000; PORTC = 0xFF.
6. Поясните действие операторов &, ! и |.
7. Что такое высокий и низкий логические уровни.
8. Что будет на выводах, если в PORTA записано 01010101?

1.5 Задачи

Решить задачи своего варианта (табл. 1):

1. Написать программы, которые выводят букву своего имени, день рождения на семисегментном диодном элементе. Изменение информации осуществляется по нажатию кнопки.

2. Реализовать бегущий огонек, который доходит до 8 разряда и начинает с 1ого (…0100 0000 -> 1000 0000 -> 0000 0001…)

3. Реализовать бегущий огонек, который доходит до 8 разряда и начинает считать в обратную сторону (…0100 0000 -> 1000 0000 -> 0100 0000 -> 0010 0000…)

4. Реализовать эффект елочки, который доходит до 8 разряда и начинает считать в обратную сторону (…0111 1111 -> 1111 1111-> 01111 1111 -> 0011 1111…)

5. Реализовать эффект елочки, который доходит до 8 разряда и начинает считать с начала (…0111 1111 -> 1111 1111-> 0000 0000 -> 0000 0001…)

6. Реализовать эффект эквалайзера (..0000 0000 –>0001 1000–>0011 1100–>0111 1110–> 1111 1111–>0111 1110–>0011 1100–>0001 1000–>0000 0000.)

7. Реализовать инверсный эффект эквалайзера (см п.6, заменив 0 на 1)

8. Елочка 2 (линейка светодиодов, подключенная к линиям порта, последовательно заполняется огнями и постоянно горит, а звезда — старший бит — моргает).

9. «Бегущий огонёк» в одну сторону бежит по выводам порта B, в другую — порта A.

10. Реализовать работу светофора по нажатию кнопки.

Таблица 1.2 – Варианты заданий

Вариант	**№ задачи**					
1	1	2	4	6	8	10
2	1	3	5	7	9	10
3	1	2	4	6	8	10
4	1	3	5	7	9	10
5	1	2	4	6	8	10
6	1	3	5	7	9	10
7	1	2	4	6	8	10
8	1	3	5	7	9	10
9	1	2	4	6	8	10
10	1	3	5	7	9	10

1.6 Содержание отчёта

Отчёт выполняется в текстовом редакторе Office Word, LibreOffice Writer или подобных.

Структура отчёта:

1. Титульный лист.
2. Введение и условия задания.
3. Результаты решения.
4. результаты исследования.
5. Заключение.

Отчёт должен включать:

- условия решаемых задач, номер варианта;
- скриншоты схем;
- текст программ (не скриншоты);
- текстовые комментарии к результатам исследования и краткие выводы.

2. Лабораторная работа № 2. Основы работы с ШИМ

2.1 Цель работы

Изучение принципов широтно-импульсного модулирования на примере модуля микроконтроллера Atmega32 посредством его программирования и моделирования в среде Proteus Design Suite.

2.2 Краткие теоретические сведения

Широтно-импульсная модуляция (ШИМ, англ. Pulse-Width Modulation, PWM) – это способ управления подачей мощности к нагрузке. Управление заключается в изменении длительности импульса при постоянной частоте следования импульсов. ШИМ

Применение широтно-импульсной модуляции позволяет повысить КПД электрических преобразователей, особенно это касается импульсных преобразователей, составляющих сегодня основу вторичных источников питания различных электронных аппаратов.

В микроконтроллерах ШИМ реализуется таким образом, что выходные импульсы принимают только одно из двух значений (включено или выключено). Выходной сигнал получается благодаря использованию N-битного счетчика.

Две важные характеристики, касающиеся ШИМ: период T, с и длительность импульса τ, с. На практике говорят о скважности $S = \frac{T}{\tau}$ (безразмерная величина) и коэффициенте заполнения (англ. Duty Cycle) $D = \frac{1}{S} = \frac{\tau}{T}$. Коэффициент заполнения говорит о доле высокого логического уровня в периоде. При 25% заполнении наибольшая величина напряжения равна четверти от напряжения, при 50% – половине, и т.д. (рис. 2.1). В результате непрерывного изменения длительности высокого уровня импульса можно получить изменяющийся во времени сигнал (рис. 2.2).

Микроконтроллер позволяет изменять частоту и скважность ШИМ посредством обращения к соответствующим битам регистров, которые управляют таймером/счётчиком.

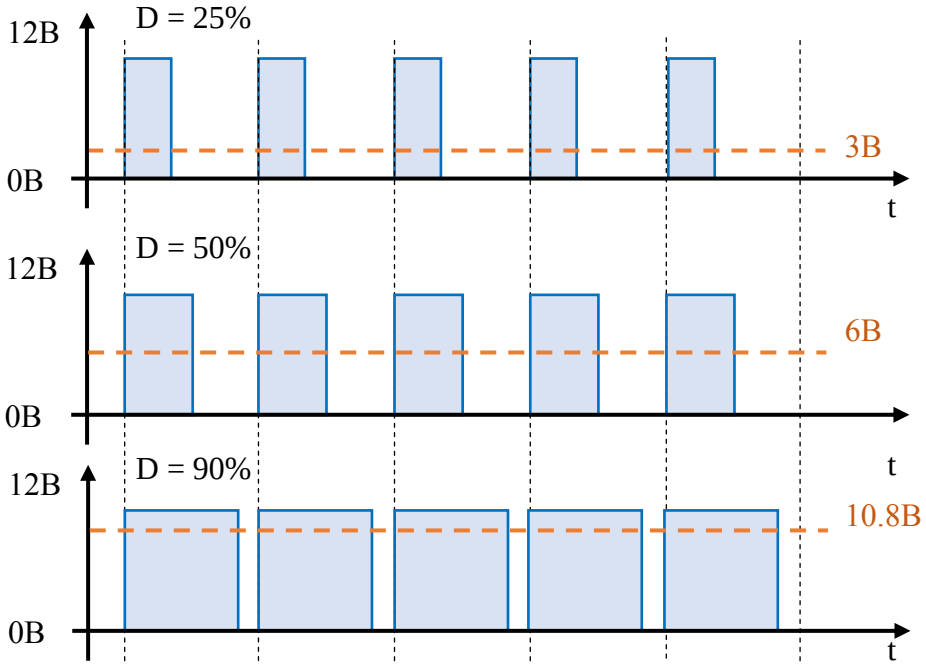


Рисунок 2.1 – Влияние скважности на напряжение выходного сигнала

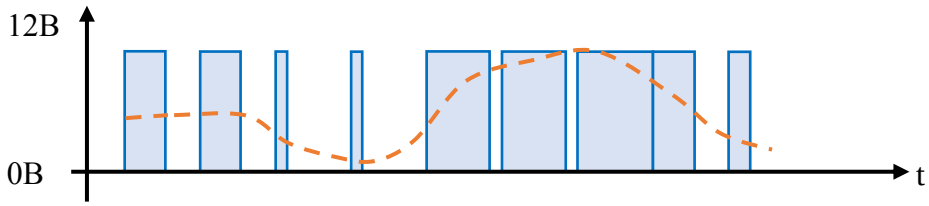


Рисунок 2.2 – Выходной сигнал при изменении скважности

Таймеры/Счётчики. Микроконтроллеры AVR имеют в своем составе таймеры/счетчики (TIMER/COUNTERS) с разрядностью 8 или 16 бит. Таймеры/счетчики можно назвать отдельными устройствами в составе микроконтроллерам, которые могут работать в нескольких режимах. Они могут работать как от внутреннего источника тактовой частоты, так и как счетчики внешних событий.

Их можно использовать для точного формирования временных интервалов, подсчета импульсов на выводах микроконтроллера (режим счётчика), формирования последовательности импульсов (т.е. ШИМ).

Каждый таймер/счетчик использует один или более выводов микроконтроллера. Как правило, эти выводы — линии портов ввода/вывода общего назначения, а функции, реализуемые этими выводами при работе совместно с таймерами/счетчиками, являются их альтернативными функциями.

При использовании альтернативных функций линий портов ввода/вывода необходимо, как правило, самостоятельно сконфигурировать эти выводы в соответствии с их функциональным назначением.

В режиме ШИМ таймер/счетчик может представлять собой широтно-импульсный модулятор и используется для генерирования сигнала с программируемыми частотой и скважностью.

Таймеры/счетчики способны вырабатывать запросы прерываний, переключая процессор на их обслуживание по событиям и освобождая его от необходимости периодического опроса состояния таймеров. Поскольку основное применение микроконтроллеры находят в системах реального времени, таймеры/счетчики являются одним из наиболее важных элементов.

Частота работы таймера/счётчика зависит от тактового сигнала подсистемы ввода/вывод $f_{clkI/O}$ (англ. input/output).

Настройки выводов микроконтроллера для режима Fast PWM. Мы рассмотрим работу с 8-битным таймером/счётчиком T0. Работа с другими таймерами аналогична. В случае работы с 16-битными таймерами или другими моделями котроллеров есть особенности, которые необходимо уточнить по даташиту.

Вывод ШИМ-сигнала осуществляется с вывода OC0 (PB3). Перечень всех используемых для ШИМ регистров показан в таблице 2.1.

Таблица 2.1 – Регистры таймера/счётчика T0

Название регистра	**Назначение**
TCCR0	Регистр управления
TCNT0	Счётный регистр
OCR0	Регистр совпадения
ASSR	Регистр состояния асинхронного режима
TIMSK	Регистр маски прерываний

Изменяя значения битов регистра TCCR0, можно задать различные режимы работы ШИМ (таблица 2.2).

Биты COM01, COM00 определяют поведение вывода OC0 при наступлении события «Совпадение». Влияние содержимого этих битов на состояние вывода зависит от режима работы таймера/счетчика.

Биты CS02:CS00 определяют источник тактового сигнала таймера/счетчика или значения масштабирующего коэффициента.

Таблица 2.2 –Регистр TCCR0

<table>
<tr><th>Бит</th><th>Название</th><th>Описание</th></tr>
<tr><td>7</td><td>FOC0</td><td>Принудительное изменение состояния вывода OC0. В режиме FastPWM и Phase Correct PWM должен быть равен 0.</td></tr>
<tr><td>6,3</td><td>WGM01,
WGM00</td><td>Режим работы таймера/счётчика. (Waveform Generation Mode Bit)
<table>
<tr><th>WGM01</th><th>WGM00</th><th>Режим работы</th></tr>
<tr><td>0</td><td>0</td><td>Нормальный</td></tr>
<tr><td>0</td><td>1</td><td>ШИМ с фазовой коррекцией</td></tr>
<tr><td>1</td><td>0</td><td>CTC</td></tr>
<tr><td>1</td><td>1</td><td>Быстрый ШИМ, Fast PWM</td></tr>
</table></td></tr>
<tr><td>5,4</td><td>COM01,
COM00</td><td>Режим работы блока сравнения. Для FastPWM:
<table>
<tr><th>COM01</th><th>COM00</th><th>Описание</th></tr>
<tr><td>0</td><td>0</td><td>Нормальное состояние порта. OC0 отключён</td></tr>
<tr><td>1</td><td>0</td><td>Очистка OC0 при сравнении. OC0 = 0x00</td></tr>
<tr><td>1</td><td>1</td><td>Установка OC0 при сравнении. OC0 = 0xFF</td></tr>
</table></td></tr>
<tr><td>2…0</td><td>CS02,
CS01,
CS00</td><td>Управление тактовым сигналом.
<table>
<tr><th>CS02</th><th>CS01</th><th>CS00</th><th>Описание</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Нет источника тактирования</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>$f_{clkI/O}$, без предделителя</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>$f_{clkI/O}/8$, от предделителя</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>$f_{clkI/O}/64$, от предделителя</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>$f_{clkI/O}/256$, от предделителя</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>$f_{clkI/O}/1024$, от предделителя</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Внешний источник от вывода T0. Падающий фронт.</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Внешний источник от вывода T0. Нарастающий фронт.</td></tr>
</table></td></tr>
</table>

Нас интересует режим Fast PWM (Быстродействующий ШИМ). Он позволяет генерировать высокочастотный сигнал с широтно-импульсной модуляцией. В связи с высокой частотой генерируемого сигнала данный режим с успехом может использоваться в таких приложениях, как регулирование мощности, выпрямление, цифро-аналоговое преобразование и др.

Частота генерируемого сигнала определяется выражением

$$f_{OC0} = \frac{f_{clkI/O}}{N \cdot 256},$$

где N – коэффициент деления предделителя (англ. prescaler) (см. табл. 2).

Обратите внимание, что значение с вывода OC0 будет видно только если порт настроен как выход (соответствующий бит DDR равен 1).

Цифро-аналоговый преобразователь (ЦАП, англ. Digital-to-analog converter, DAC). ЦАП предназначен для преобразования цифрового кода в аналоговый сигнал. Мы будем использовать свойства RC-цепочки или тот факт, что RC-цепь является фильтром низких частот (отсеиваем низкие частоты, отметая в сторону высокие). По факту RC-цепь преобразует не цифровой код, а ШИМ в аналоговый сигнал.

2.3 Практические примеры

Инициализация ШИМ. Для демонстрации инициализации ШИМ соберём простую схему с контроллером, осциллографом и счётчиком частоты (рис. 2.3).

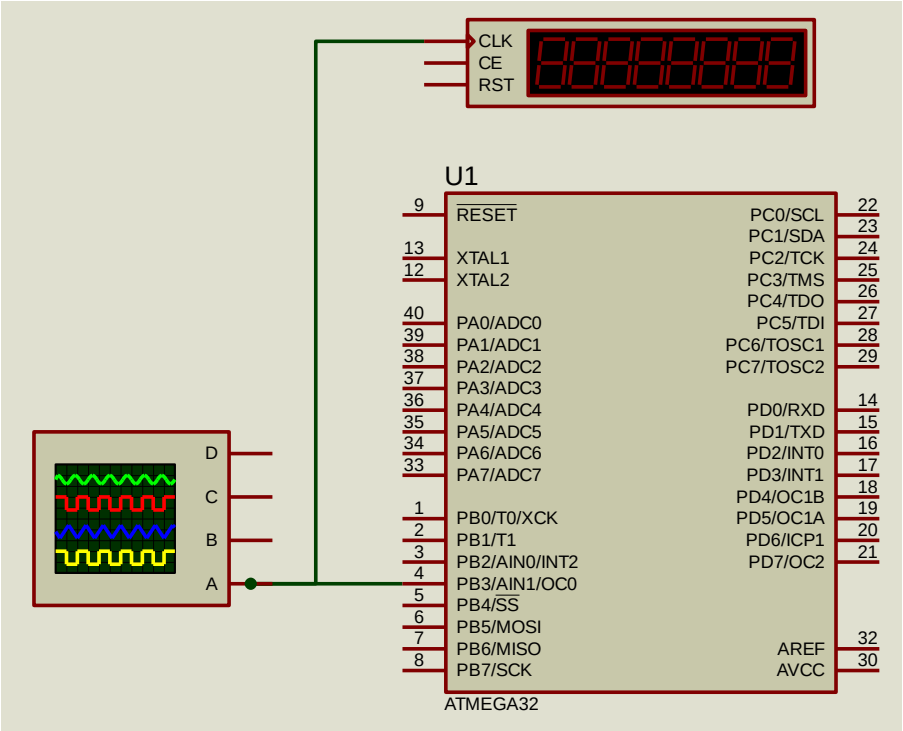


Рисунок 2.3 – Схема микроконтроллера с осциллографом

Посмотрим, как изменяется сигнал при различных настройках.

Следующая программа задаёт режим ШИМ как FastPWM. Настраиваем режимы в регистрах (TCCR0, OCR0, TIMSK, ASSR). Обратите внимание на 6 строку – работа ШИМа без предделителя. Таким образом, мы узнаем его наибольшую скорость.

```
#include <avr/io.h>

int main(void) {
    ASSR = 0x00;  // не асинхронный режим

    TCCR0 = 0b01001000;  // FastPWM
    TCCR0 |= 0b00100000;  // очистка при сравнении
    TCCR0 |= 0b00000001;  // Предделитель

    TCNT0 = 0x00;  // timer counter
    OCR0 = 0x00;  // заполнение импульса - нуль
    TIMSK = 0x00;  // не принимаем никакие прерывания

    DDRB = 0b00001000;  // выход
    PORTB = 0x00;

    OCR0 = 0x0F;  // скважность FF/F = 255/15 = 17 %

    while (1) {

    }
}
```

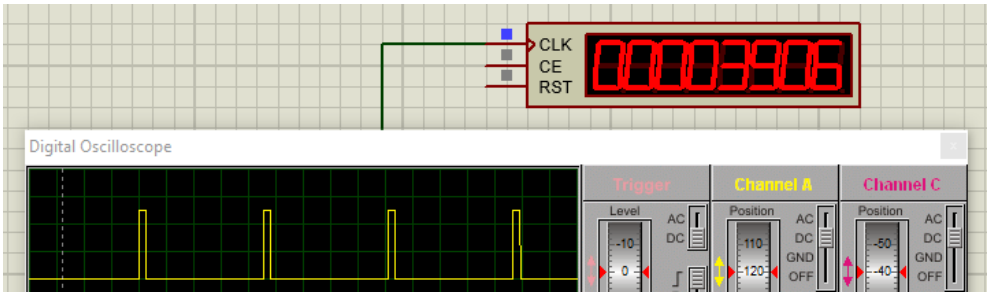
При запуске симулятора можем увидеть значение частотомера равное 3906 Гц (рис. 4а). На самом деле это не совсем верно. Настоящее значение должно быть степенью числа 2. В данном случае ближайшее это 4096 Гц. (Внимание! Здесь могут быть ошибки вычисления частоты и не соответствие реальным значениям. Поэтому здесь я доверяюсь симулятору Proteus).

Обратите внимание, что канал осциллографа настроен для получения значений постоянного тока (DC).

Изменяя значения предделителя (регистр TCCR) увидим, чему равны частоты: 488, 61, 4 Гц. В идеальном случае должно быть соответственно: 512, 64, 4 Гц.

Изменяя значение регистра OCR0, увеличиваем скважность до 50% (рис. 2.4б).

а)



б)

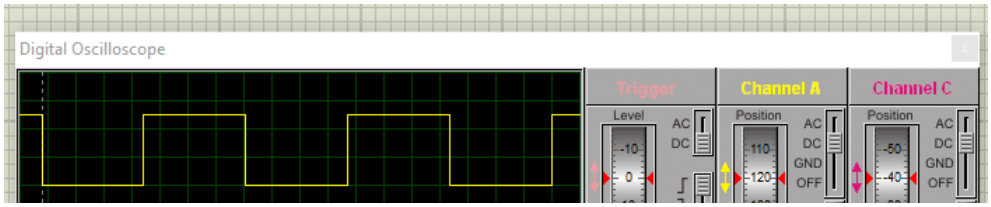
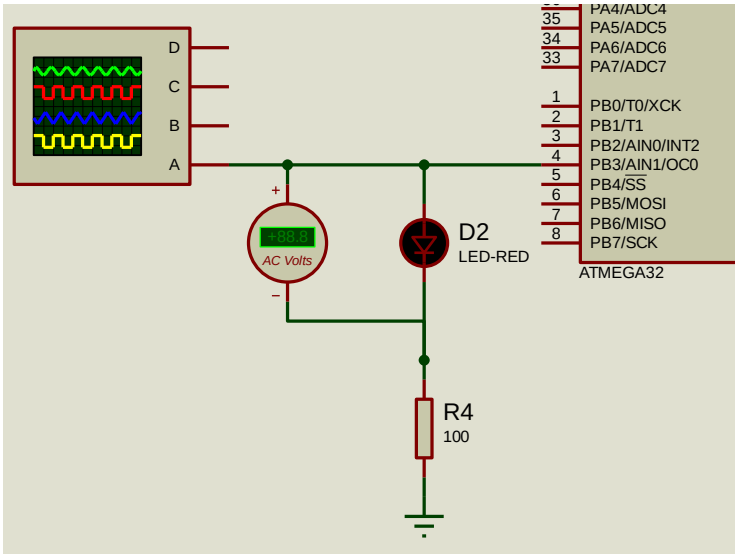


Рисунок 2.4 – Частота ШИМ сигнала (а) и 50% скважность (б)

Плавное свечение светодиода. Одним из показательных примеров использования является плавное зажигание светодиода. Вспомните, что до сих пор мы могли включить светодиод в полную яркость либо выключить. Когда это будет происходить с высокой частотой – глаз будет воспринимать это как изменение яркости свечения светодиода. Ещё это можно увидеть с помощью вольтметра (рис. 2.5): напряжение изменяется.

а)



б)

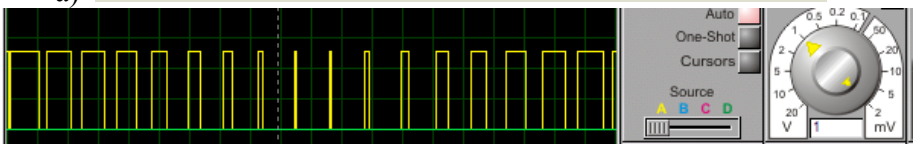


Рисунок 2.5 – Плавное зажигание светодиода с помощью ШИМ

Следующий код показывает, как можно изменять яркость светодиода: плавно зажигать его, а затем плавно гасить.

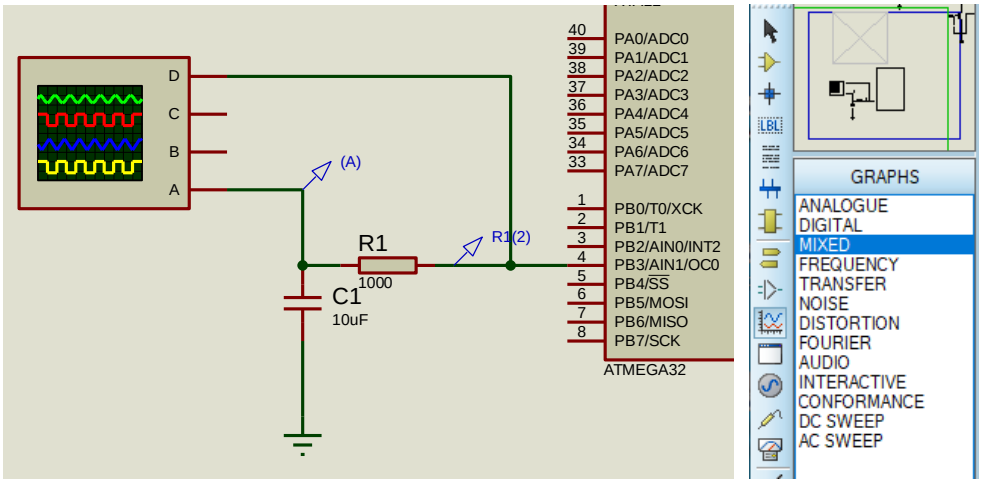
Производим настройку режима ШИМ на FastPWM: делитель на 1024, скважность 0%. Вывод порта на выход.

Переменная _debug_oc0 введена для отладки в Atmel Studio, чтобы наблюдать изменение регистра OCR0 и другого смысла не имеет.

В цикле *while(1)*, увеличиваем скважность до тех пор, пока она не достигнет уровня 100%, а затем уменьшаем до уровня 0%.

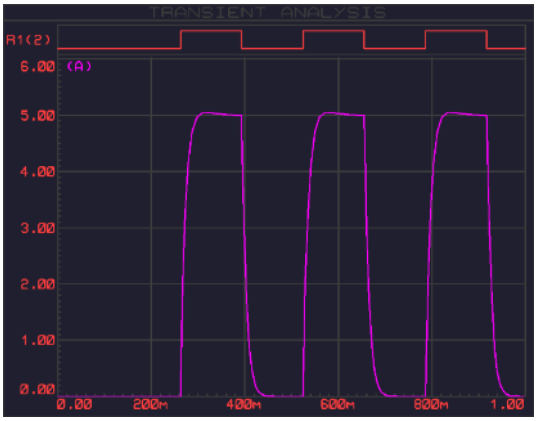
```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>

int main(void)
{
   ASSR = 0x00;  // не асинхронный режим

   TCCR0 = 0b01001000;  // FastPWM
   TCCR0 |= 0b00100000;  // очистка при сравнении
   TCCR0 |= 0b00000101;  // Без предделителя 3906, 8-488, 64-61, 256-16, 1024-4

   TCNT0 = 0x00;  // timer counter
   OCR0 = 0x00;  // заполнение импульса (скважность) - нуль
   TIMSK = 0x00;  // не принимаем никакие прерывания

   DDRB = 0b00001000;  // выход
   PORTB = 0x00;

   // int _debug_oc0 = OCR0;  // для отладки

   while (1)
   {
      while (OCR0 < 0xFF)
      {
         OCR0 += 0x0F;
         //_debug_oc0 = OCR0;  // для отладки
         _delay_ms(10);
      }

      while (OCR0 > 0x00)
      {
         OCR0 -= 0x0F;
         //_debug_oc0 = OCR0;  // для отладки
         _delay_ms(10);
      }
   }
}
```

ЦАП на основе RC-цепи будет собран с использованием резистора 1 КОм и конденсатора на 10 мкФ (рис. 2.6). Все последующие графики будут показаны с помощью инструмента MIXED. Изменяя ёмкость конденсатора, скважность ШИМ-сигнала можно увидеть, как будет изменяться вид графика напряжения (рис. 2.7-2.9).

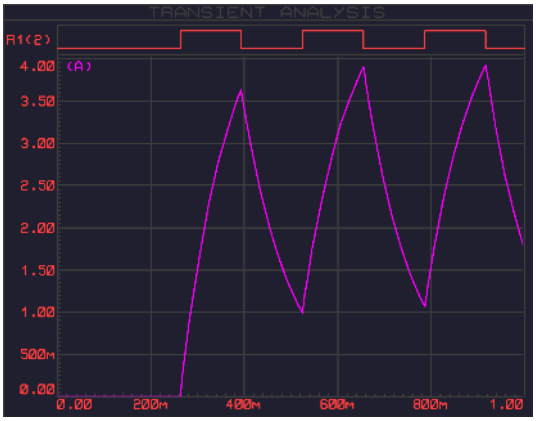


а) б)

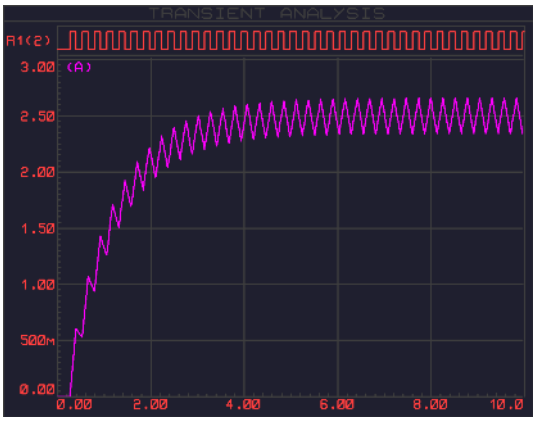
Рисунок 2.6 – ЦАП на RC-цепочке (а) и инструмент для просмотра (б)



а)



б)



в)

Рисунок 2.7 – Влияние ёмкости конденсатора на выходной сигнал:
а) – 10uF, б) – 100uF, в) – 1000uF

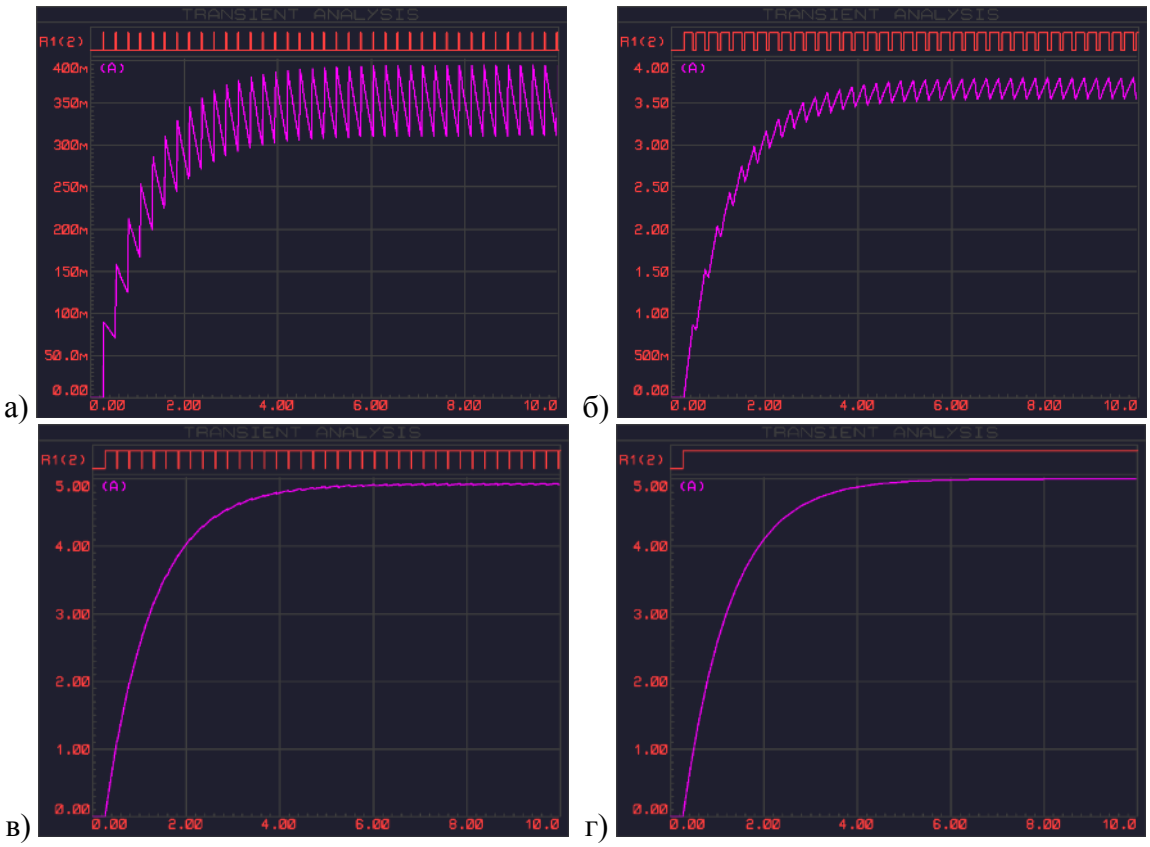


Рисунок 2.8 – Выход ЦАП при разных значениях скважности (OCR0): а –0x11, б – 0xBB, в – 0xFB, г – 0xFF) на выходной сигнал

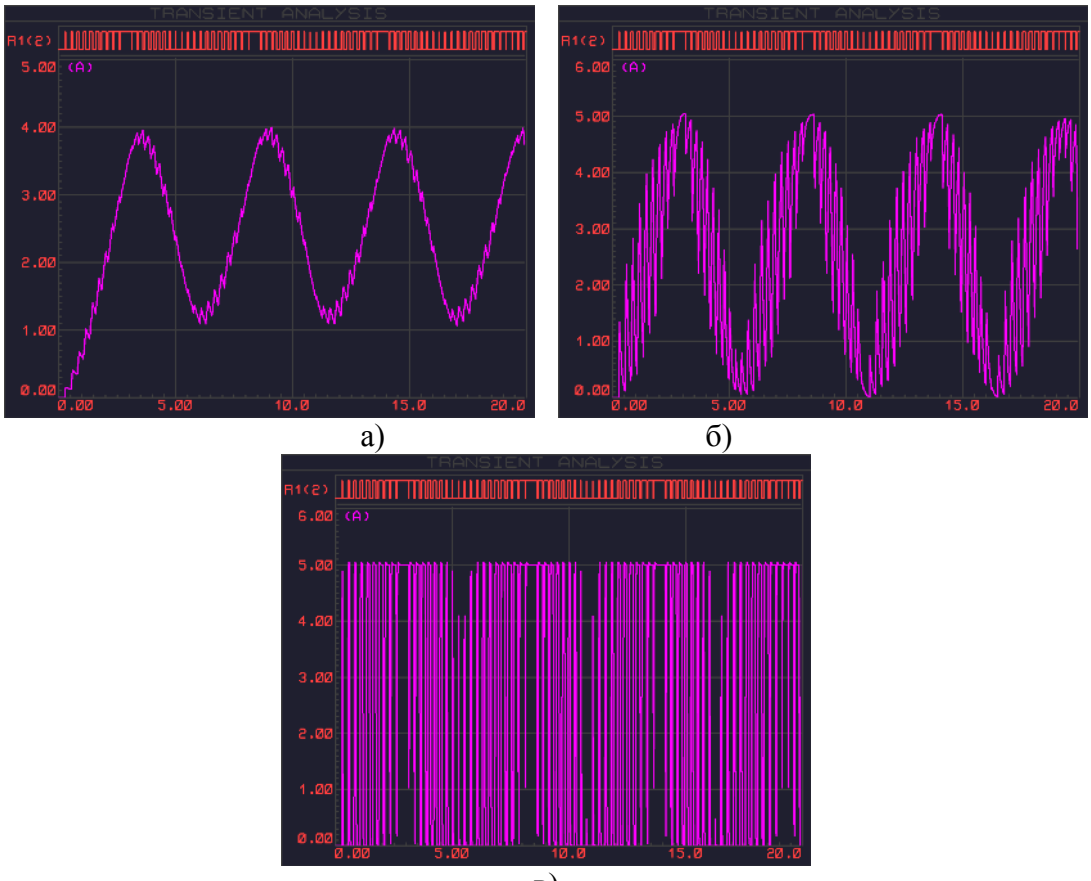


Рисунок 2.9 – Плавное изменение значений скважности ШИМа (а – 1000uF, б – 100uF, в – 10uF)

2.4 Контрольные вопросы

1. Опишите принцип работы ШИМ.
2. Принцип расчёта напряжения с помощью ШИМ.
3. Опишите регистр управления ШИМ в Atmega32.
4. Опишите принцип работы ЦАП на основе RC-цепи.
5. Произвести расчёт частоты генерируемого сигнала.

2.5 Задачи

Решить задачи своего варианта (табл. 3):

1. Задать режим работы ШИМ. Воспроизвести осциллограммы.
2. Плавно повышать и понижать яркость светодиода. Установить необходимую задержку перед повышением яркости. Где следующие значения заданы как % от наибольшего значения:
    a. s_{rise} шаг увеличения яркости,
    b. $s_{decrease}$ шаг уменьшения яркости,
    c. *min* минимальный уровень яркости,
    d. *max* максимальный уровень яркости,
    e. *delay* задержка.
3. Исследовать влияние элементов RC-цепочки на форму и качество выходного сигнала. Изменять значения в сторону увеличения и уменьшения значений элементов ЦАП. Сделать заключение.

Таблица 2.3 – Варианты заданий

	Задача 1		Задача 2					Задача 3
Вариант	Заполнение, %	Предделитель	s_{rise}	$s_{decrease}$	*min*	*max*	*delay, мс*	
1	10, 80	1, 1024	5	40	0	100	500	Для всех вариантов
2	20, 70	8, 1	15	25	30	100	300	
3	30, 60	64, 256	25	5	50	100	500	
4	40, 80	256, 8	5	40	0	75	400	
5	50, 90	1024, 256	15	25	30	75	300	
6	60, 10	8, 1024	25	15	50	100	750	
7	70, 20	64, 8	5	25	0	75	300	
8	80, 30	256, 64	15	40	30	75	500	
9	90, 40	1024, 256	25	25	50	100	500	
10	20, 60	64, 1024	5	15	0	75	300	

2.6 Содержание отчёта

Структура отчёта:

1. Введение и условия задания.
2. Результаты решения задач.
3. Результаты исследования.
4. Заключение.

Отчёт должен включать:

- условия решаемых задач, номер варианта;
- скриншоты схем;
- текст программ (не скриншоты);
- текстовые комментарии к результатам исследования и краткие выводы.

3. Лабораторная работа № 3. Основы работы с АЦП

3.1 Цель работы

Изучение принципов работы аналого-цифрового преобразователя микроконтроллера Atmega32 посредством его программирования и моделирования в среде Proteus Design Suite.

3.2 Краткие теоретические сведения

Аналого-цифровой преобразователь (АЦП, англ. Analog-to-digital converter, ADC) – это устройство, преобразующее входной аналоговый сигнал в цифровой код. Как правило, в качестве входного сигнала используется напряжение [1]. АЦП является отдельным устройством внутри микроконтроллера (МК).

Аналого-цифровое преобразование (АЦ-преобразование) используется везде, где требуется принимать аналоговый сигнал и обрабатывать его в цифровой форме:

- АЦП является составной частью цифрового вольтметра и мультиметра.
- Звуковые карты компьютеров преобразуют аналоговый сигнал микрофонов.
- Специальные видео-АЦП используются в компьютерных ТВ-тюнерах, платах видеовхода, видеокамерах для оцифровки видеосигнала. Микрофонные и линейные аудиовходы компьютеров подключены к аудио-АЦП.
- Очень быстрые АЦП необходимы в цифровых осциллографах.
- Весы используют АЦП, преобразующие сигнал непосредственно от тензометрического датчика (сигма-дельта-АЦП).
- Резистивные и полупроводниковые датчики (например, LM35, LM135, TMP37) дают информацию о внешней среде путём изменения своих свойств. Выходной вывод возвращает напряжение, которое необходимо преобразовать. Другие датчики могут иметь встроенный АЦП (например, датчики с встроенным АЦП DS18B20, LM75, MSP9808).

Идея АЦ-преобразования. Предположим, что есть АЦП с разрядностью 1, а на его вход может быть подан сигнал в диапазоне от 0 до 5 В. Такой АЦП может распознать только 2^1 уровня напряжения: на выходе устанавливается 0, если на входе от 0 до 2.5 В; и 1, если на входе от 2.5 до 5 В:

$$A = A(V_{in}) = \begin{cases} 0_2, & \text{если } 0 \le V_{in} < 2.5; \\ 1_2, & \text{если } 2.5 < V_{in} \le 5. \end{cases}$$

Если АЦП с разрядностью 2, то возможно распознать 2^2 уровня входного сигнала:

$$A = A(V_{in}) = \begin{cases} 0_2, & \text{если } 0 \le V_{in} < 1.25; \\ 1, & \text{если } 1.25 \le V_{in} < 2.5; \\ 10_2, & \text{если } 2.5 \le V_{in} < 3.75; \\ 11_2, & \text{если } 3.75 \le V_{in} < 5. \end{cases}$$

Количество уровней (количество чисел в диапазоне), которое может быть на выходе АЦП равно 2^N, где N – разрядность АЦП. В двоичных АЦП вместо разрядности говорят о битности АЦП [4,5].

На рис. 3.1 изображены уровни напряжений для АЦП разрядности 1, 2, 3.

В приведённых выше примерах диапазон входной величины наблюдается от 0 до 5 В. В общем случае этот диапазон от 0 до опорного напряжения V_{REF}. Опорное напряжение определяет границу диапазона, с которым будет работать АЦП. В ATmega32 есть внутренний источник опорного напряжения, равный 2.56 В, но можно подключить и внешний источник через вывод AREF или использовать напряжение питания АЦП через вывод AVCC. (Обычно AVCC подключают к выводу питания МК).

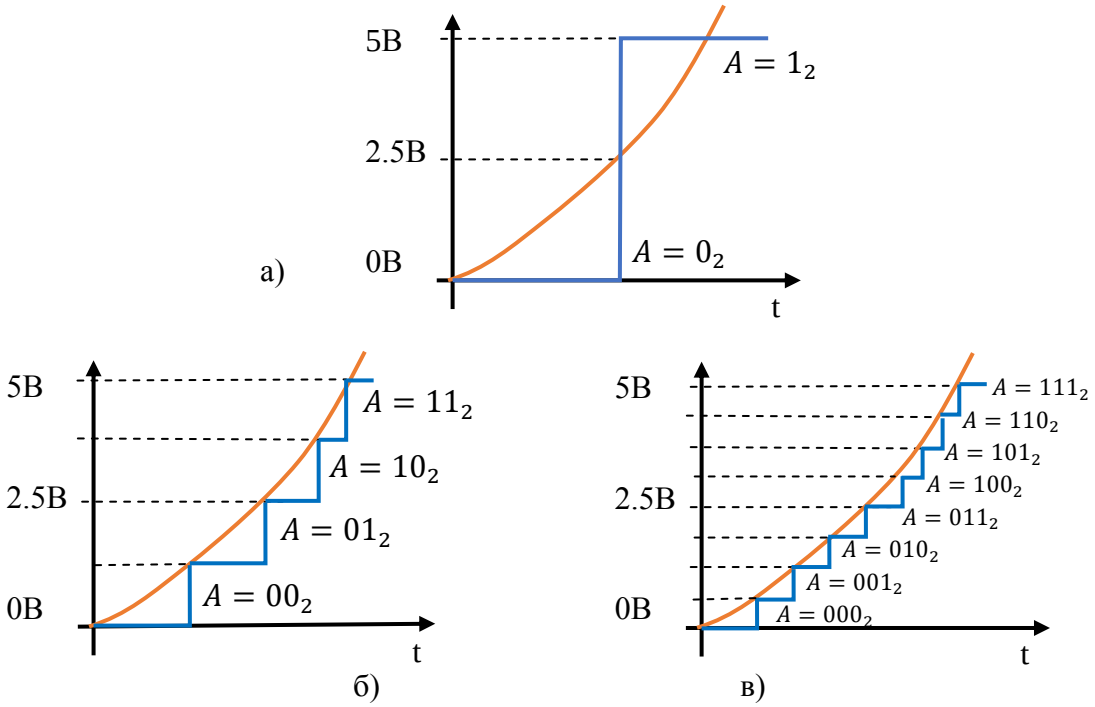


Рисунок 3.1 – Входное напряжение и уровни напряжений
АЦП разрядности 1 (а), 2 (б), 3 (в)

Формула расчета точности АЦП:

$$C = \frac{V_{REF}}{2^N},$$

где C – точность АЦП в вольтах; V_{REF}— величина опорного напряжения.

Для примеров выше точность были соответственно: 5/2 = 2.5 В, 5/4 = 1.25 В, 5/8 = 0.625 В.

Этот нюанс может стать причиной ошибки измерения с помощью АПЦ. Для изменения точности АЦП необходимо изменять разрядность АЦП или опорное напряжение.

Как происходит процесс получения конкретного значения на выходе АЦП? Существует несколько основных типов АЦП: параллельного преобразования, последовательного приближения, интегрирующие АЦП, сигма-дельта АЦП. В ATmega32 *применятся АЦП последовательного преобразования.* Этот тип преобразования считается хорошим в скорости и точности. Рассмотрим пример получения значения этим методом.

АЦП содержит компаратор (для сравнения аналоговых значений), вспомогательный ЦАП (для формирования аналогово сигнала) и регистр последовательного приближения (для хранения результата). Преобразование аналогового сигнала в цифровой сигнал производится за N шагов, где N — разрядность АЦП. На каждом шаге определяется по одному биту искомого цифрового значения, *начиная от старшего значащего разряда* и заканчивая младшим значащим разрядом (англ. Least Significant Bit, LSB) [2]. Последовательность шагов по определению очередного бита:

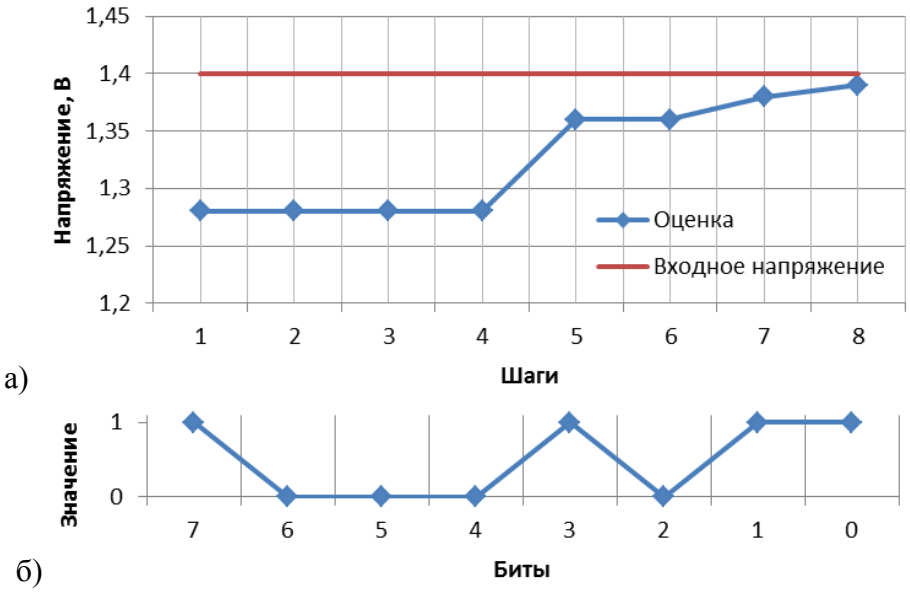
1. Все биты на выходе равны 0.

2. На вспомогательном ЦАП выставляется аналоговое значение, образованное из битов, определённых на предыдущих шагах; бит, который должен быть определён на этом шаге, выставляется в 1.

3. Полученное на вспомогательном ЦАП значение сравнивается с входным аналоговым значением. Если значение входного сигнала больше, чем на вспомогательном ЦАП, то определяемый бит получает значение 1, в противном случае 0.

4. Повторить шаги 2,3,4 пока не дойдём до младшего разряда.

На рис. 3.2 показан пример реализации этого алгоритма для опорного напряжения $V_{REF} = 2.56$ В, входного сигнала $V_{IN} = 1.4$ В. Значение старшего бита соответствует $\frac{1}{2}V_{REF}$, следующий бит $\frac{1}{4}V_{REF}$, последующий $\frac{1}{8}V_{REF}$ и т.д. В результате выполнения описанного алгоритма на выходе АЦП будет сформировано значение, равное $10001011_2 = 8B_{16} = 139_{10}$. Этому значению будет соответствовать напряжение, равное $V_{ADC} = 1.39$ В. Обратите внимание, что это значение меньше V_{IN}.



а)

б)

Рисунок 3.2 – Измерение напряжения в АЦП последовательного приближения
а) оценка входного напряжение, б) значение битов

В ATmega32 установлен 10-битный АЦП последовательного приближения с 8 входными каналами. АЦП преобразовывает аналоговое входное напряжение в 10-битное число путём последовательного приближения. Наименьшее значение представлено землёй (GND), а наибольшее значение V_{ADC_MAX}:

$$V_{ADC\_MAX} = V_{REF} - L,$$

где V_{REF}— величина опорного напряжения, $L = 1$ LSB – младший значащий бит.

В качестве источника опорного напряжения для АЦП может использоваться как напряжение питания микроконтроллера, так и внутренний либо внешний источник опорного напряжения.

Модуль АЦП ATmega32 может работать в двух режимах:

− режим одиночного преобразования, когда запуск каждого преобразования инициируется пользователем;

− режим непрерывного преобразования, когда запуск преобразований выполняется непрерывно через определенные интервалы времени.

Включение и настройка АЦП. Для управления АЦП ATmega32 используются управляющие регистры ADCSRA, ADMUX и SFIOR (табл. 3.1). Подробное описание регистров показано в Приложении А.

Таблица 3.1 – Управляющие регистры АЦП

Название регистра	**Назначение**
ADCSRA	Регистр А управления и состояния. ADC Control and Status Register A
ADMUX	Регистр управления мультиплексором. ADC Multiplexer Selection Register
SFIOR	Регистр специальных функций. Special Function IO Register
ADC	Регистр данных (содержит регистры ADCH и ADCL)

АЦП включается путём установки бита ADEN регистра ADCSRA равным 1, выключается при установке в 0 (табл. 3.2). Изменения не будут приниматься, пока этот флаг не будет установлен в 1. Если АЦП будет выключен во время цикла преобразования, то преобразование завершено не будет (в регистре данных АЦП останется результат предыдущего преобразования).

Таблица 3.2 – Регистр ADCSRA

<table>
<tr><th>Бит</th><th>Название</th><th>Описание</th></tr>
<tr><td>7</td><td>ADEN</td><td>ADC Enable. Разрешить работу АЦП (1 – включено, 0 – выключено)</td></tr>
<tr><td>6</td><td>ADSC</td><td>ADC Start Conversion. Запуск преобразования (1 – начать преобразование)</td></tr>
<tr><td>5</td><td>ADATE</td><td>ADC Auto Trigger Enable. Выбор режима работы АЦП</td></tr>
<tr><td>4</td><td>ADIF</td><td>ADC Interrupt Flag. Флаг прерывания от компаратора</td></tr>
<tr><td>3</td><td>ADIE</td><td>ADC Interrupt Enable. Разрешение прерывания от компаратора</td></tr>
<tr><td>2:0</td><td>ADPS2:0</td><td>ADC Prescaler Select Bits. Выбор частоты преобразования
<table>
<tr><th>ADPS2</th><th>ADPS1</th><th>ADPS0</th><th>Коэффициент предделителя (Division Factor)</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>2</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>4</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>8</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>16</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>32</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>64</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>128</td></tr>
</table>
</td></tr>
</table>

Для работы АЦП необходимо установить его тактовую частоту. Источником тактирования АЦП является тактовый генератор МК. Оптимальная частота преобразования лежит в диапазоне 50-200 кГц. Для уменьшения частоты тактирования в АЦП имеется предделитель. Коэффициент деления предделителя и соответственно длительность преобразования определяются состоянием битов ADPS2...ADPS0 регистра ADCSRA. Например, для получения частоты АЦП в 125 кГц, при тактовой частоте МК 16 МГц необходимо выбирать предделитель $\frac{16\ \text{Мгц}}{125\text{кГц}} = 128$, что соответствует значениям битов ADPS2...ADPS0 равными 111.

Запуск АЦП возможен по команде пользователя или по прерыванию от некоторых периферийных устройств, имеющихся в составе микроконтроллера. Для выбора режима работы используется бит ADATE регистра ADCSRA и биты ADTDS2:0 регистра SFIOR (табл. 3.3). Если бит ADATE равен 0, то АЦП работает в режиме одиночного преобразования.

Таблица 3.3 – Регистр SFIOR

<table>
<tr><th>Бит</th><th>Название</th><th>Описание</th></tr>
<tr><td>7:5</td><td>ADTS2:0</td><td>Источник для запуска преобразования. ADC Auto Trigger Source
<table>
<tr><th>ADTS2</th><th>ADTS1</th><th>ADTS0</th><th>Источник стартового сигнала</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Режим непрерывного преобразования</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Прерывание от аналогово компаратора</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Внешнее прерывание INT0</td></tr>
<tr><td colspan="4">Другие режимы</td></tr>
</table>
</td></tr>
</table>

Коэффициент деления предделителя, используемый для формирования тактовой частоты модуля АЦП, и соответственно длительность преобразования определяются состоянием битов ADPS2...ADPS0 регистра ADCSRA.

Биты ADIF и ADIE регистра ADCSRA предназначены для настройки прерывания: результат будет возвращён по окончанию работы АЦП. Используя прерывания тратить время на ожидание результатов не надо будет. Этот пункт не рассматриваем, поскольку прерывание в данной работе не используем.

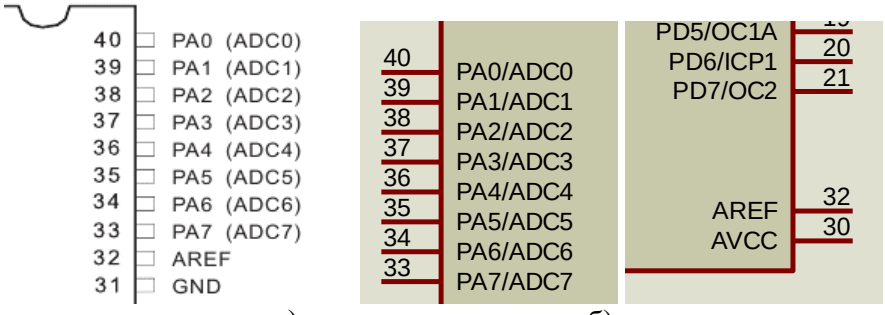
Биты REFS1:REFS0 регистра ADMUX определяют источник опорного напряжения. Мы работаем с внутренним источником V_{cc} = 2.56 В, поэтому они должны быть равны 1 (табл. 3.4).

Таблица 3.4 – Регистр ADMUX

<table>
<tr><th>Бит</th><th>Название</th><th>Описание</th></tr>
<tr><td>7,6</td><td>REFS1:REFS0</td><td>Выбор источника опорного напряжения. Reference Selection Bits.
<table>
<tr><td>REFS1</td><td>REFS0</td><td>Выбор опорного напряжения</td></tr>
<tr><td>0</td><td>0</td><td>AREF, Внутренний источник отключён</td></tr>
<tr><td>0</td><td>1</td><td>AVCC с внешним конденсатором на AREF</td></tr>
<tr><td>1</td><td>0</td><td>Зарезервировано</td></tr>
<tr><td>1</td><td>1</td><td>Внутренний источник 2.56 В опорного напряжения с внешним конденсатором на AREF</td></tr>
</table></td></tr>
<tr><td>5</td><td>ADLAR</td><td>Выравнивание результата преобразования. ADC Left Adjust Result</td></tr>
<tr><td>4:0</td><td>MUX4:MUX0</td><td>Выбор входного канала.
<table>
<tr><td>MUX3:0</td><td>Вывод</td></tr>
<tr><td>0000</td><td>ADC 0</td></tr>
<tr><td>0001</td><td>ADC 1</td></tr>
<tr><td>0010</td><td>ADC 2</td></tr>
<tr><td>0011</td><td>ADC 3</td></tr>
<tr><td>0100</td><td>ADC 4</td></tr>
<tr><td>0101</td><td>ADC 5</td></tr>
<tr><td>0110</td><td>ADC 6</td></tr>
<tr><td>0111</td><td>ADC 7</td></tr>
</table></td></tr>
</table>

Выводы микроконтроллера, подключенные к входу АЦП, определяются состоянием битов MUX4:0 регистра ADMUX. Эти выводы также совмещены с выводами порта А. Расположение пинов показано на рис. 3.3.

Выбор канала АЦП должен предшествовать старту преобразования. Переключение каналов лучше всего осуществлять после завершения преобразования.



а) б)

Рисунок 3.3 – Выводы, подключённые к АЦП в а) Datasheet, б) Proteus

Запуск каждого преобразования в режиме одиночного преобразования, а также запуск первого преобразования в режиме непрерывного преобразования осуществляется установкой в 1 бита ADSC регистра ADCSRA. Преобразование начинается по первому нарастающему фронту тактового сигнала после установки бита ADSC. После завершения преобразования, бит ADSC аппаратно сбрасывается в 0 (в режиме одиночного преобразования), и результат сохраняется в регистре данных АЦП.

Если АЦП работает в режиме непрерывного преобразования, то новый цикл начнется сразу же после записи результата. В режиме одиночного преобразования новое преобразование может быть запущено сразу же после сброса бита ADSC (до сохранения результата текущего преобразования). Однако в действительности цикл преобразования начнется не ранее чем через один такт после окончания текущего преобразования.

Получение результата. После завершения преобразования результат сохраняется в регистре данных АЦП, доступном только для чтения. Поскольку АЦП 10-битный – результатом является 10-битное число. Физически оно размещено в двух регистрах ADCH и ADCL, которые представляют собой верхний и нижний байты одного числа (табл. 3.5).

Первые 6 битов ADCH ничего не содержат. Последовательность регистров данных можно изменить установкой бита ADLAR в регистре ADMUX.

Таблица 3.5 – Регистры данных ADCH и ADCL

Биты	15	14	13	12	11	10	9	8
ADCH	–	–	–	–	–	–	ADC9	ADC8
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
Биты	7	6	5	4	3	2	1	0

Для получения данных сначала необходимо прочитать регистр ADCL, а затем ADCH. Это требование связано с тем, что после обращения к регистру ADCL процессор блокирует доступ к регистрам данных со стороны АЦП до тех пор, пока не будет прочитан регистр ADCH. Благодаря этому можно быть уверенным, что при чтении регистров в них будут находиться составляющие одного и того же результата. Соответственно, если очередное преобразование завершится до обращения к регистру ADCH, результат преобразования будет потерян.

Для упрощения работы можно считывать результат преобразования из регистра ADC, который находится по тем же адресам что и ADCH и ADCL, но даёт доступ ко всем 16 битам. При включении микроконтроллера в регистре данных АЦП содержится значение $0000.

Для каналов с несимметричным входом результат преобразования определяется выражением

$$A = 2^N \cdot \frac{V_{IN}}{V_{REF}}$$

где A – значение, определяемое АЦП, V_{IN} – значение входного напряжения, V_{REF} – величина опорного напряжения, N – разрядность АЦП.

Если входное напряжение на канале выше V_{REF}, то на выходе АЦП будет код, близкий к 0x3FF [2, гл. ADC Voltage Reference.].

Особенности.

Длительность цикла преобразования составляет 13 тактов при использовании несимметричного входа; выборка и запоминание входного сигнала осуществляется в течение первых 1.5 и 2.5 тактов соответственно. Через 13 тактов преобразование завершается.

При первом запуске после включения АЦП для выполнения преобразования потребуется 25 тактов, т. е. на 12 тактов больше, чем обычно. В течение этих 12 тактов выполняется «холостое» преобразование, во время которого производится инициализация АЦП.

Наибольшая точность преобразования достигается, если тактовая частота модуля АЦП находится в диапазоне 50...200 кГц. Соответственно, коэффициент деления преддели-

теля рекомендуется выбирать таким, чтобы тактовая частота модуля АЦП находилась в указанном диапазоне. Если же точности преобразования меньше 10 битов достаточно, можно использовать более высокую частоту, увеличивая тем самым частоту выборки.

Состояние битов MUX4...MUX0 можно изменить в любой момент, однако если это будет сделано во время цикла преобразования, то смена канала произойдет только после завершения преобразования. Благодаря этому в режиме непрерывного преобразования можно легко осуществлять последовательное преобразование сигналов нескольких каналов.

В результате АЦ-преобразования возникают ошибки, например ошибки квантования: из-за округления (до определённого разряда) или усечения (отбрасывания младших разрядов) сигнала.

Подробное описание электрических характеристик АЦП можно найти в [2, гл. ADC Characteristics].

Что не было рассмотрено в данной работе, но, будет важным на практике:

1. На практике встречаются датчики и схемы, отличающиеся по уровням напряжений от микроконтроллера. Поэтому всегда необходимо проверять опорное напряжение и диапазон возможных напряжений на канале, по которому происходит АЦ-преобразование.
2. Ошибки измерений АЦП. Подробнее смотреть в [2, Analog Noise Cancelling Techniques, Offset Compensation, ADC Accuracy Definitions].
3. Не рассмотрен дифференциальный режим преобразования АЦП, который может быть полезен в некоторых приложениях.
4. Не рассмотрен режим АЦ-преобразования по прерыванию. Данный режим позволяет не ждать обработки данных от АЦП, а тратить такты МК на другие задачи.

3.3 Практические примеры

Пример с переменным резистором. Есть переменный резистор и светодиод (рис. 3.4). Необходимо сделать индикацию: если резистор накручен более чем на 50%, то светодиод включается, иначе гаснет.

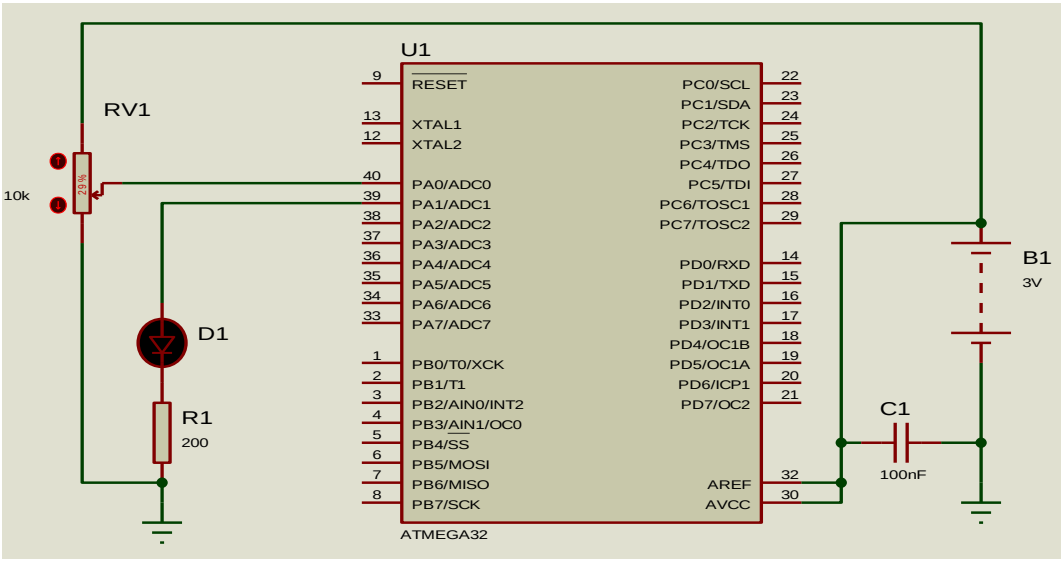


Рисунок 3.4 – Схема простой индикации сопротивления

Для решения потребуется следующие компоненты:

- Переменный резистор POT-HG (интерактивный),
- МК Atmega32,
- конденсатор CAP цепи питания АЦП,
- батарейка BATTERY для питания АЦП и входного канала,
- светодиод LED-GREEN,

– резистор RES токоограничивающий для светодиода.

Мы используем внутренний источник напряжения, поэтому в цепи питания необходимо включить конденсатор [2, гл. ADC].

Настройка АЦП. Для улучшения читаемости все действия вынесены в отдельные функции. Функция *adc_init* выполняет настройку АЦП и калибровочный запуск. Изначально регистры ADCSRA и ADMUX обнуляем. Включаем АЦП. Заполняя биты ADPS2:ADPS0 единицами, устанавливаем предделитель на 128, что обозначает работу АЦП на частоте 125кГц. АЦП будет работать в режиме одиночного преобразования при установке флага ADSC, потому что бит ADATE равен 0. Используем внутренний источник опорного напряжения.

Выполняем калибровочный запуск АЦП. Пока оно выполняется – ожидаем.

```
void adc_init(){
   ADCSRA = 0;
   ADMUX = 0;

   ADCSRA |= 1 << ADEN;
   ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
   // ADATE == 0 - одиночное преобразование
   // без прерываний
   ADMUX |= (1 << REFS1) | (1 << REFS0);   // внутренний ист.напряж.
   // вывод PA0 на АЦП

   // калибровочный запуск
   ADCSRA |= (1<<ADSC);
   while(ADCSRA & (1<<ADSC));  // ожидание результата
}
```

Идентификаторы ADEN, ADPS2, REFS1, ADSC и другие определены как числа, поэтому при выполнении кода

```
ADCSRA |= 1 << ADEN;
```

мы выполняем сдвиг 1 влево на соответствующее количество знаков.

Основная программа выполняет вызов функций и настройку портов для включения светодиода, основной цикл снятия результатов и включения светодиода.

```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>

// объявление и реализация функций
// ...

int main(void)
{
   adc_init();

   DDRA = 0b00000110;
   PORTA = 0b00000000;

   int adc_val = 0;

    while(1) {
     adc_val = adc_read(0);
```

```
      if (adc_val > 512)
         PORTA = 0b00000010;
      else
         PORTA = 0b00000000;
   }
}
```

Получение результата преобразования выполняет функция *adc_read*. Параметр *ch* определяет номер канала (вывод порта), по которому проводится преобразование. Поскольку в регистре ADMUX только первые 5 бит обозначают каналы, то остальные необходимо маскировать.

После установки битов канала запускаем преобразование и ждём. Результат возвращаем в переменную *adc_val*. Если результат преобразования больше 512 (напряжение больше половины, а значит и резистор более чем на половину сдвинут).

```
uint16_t adc_read(uint16_t ch){
   ADMUX = (ADMUX & 0b00011111) | ch;
   ADCSRA |= (1<<ADSC);
   while(ADCSRA & (1<<ADSC));
   return ADC;
}
```

Пример с датчиком температуры LM35. Принципиально работа с аналоговым датчиком температуры не отличается от рассмотренного примера выше. Данные считываются также, результат можно вывести таким же образом через диоды (а лучше через 7-сегментный индикатор). Единственная особенность заключается в настройке соответствия температуры значениям на выходе АЦП. Эта настройка осуществляется путём изменения пороговых значений для *adc_val*.

Принципиальная схема с подключением датчика LM35 показана на рис. 3.5.

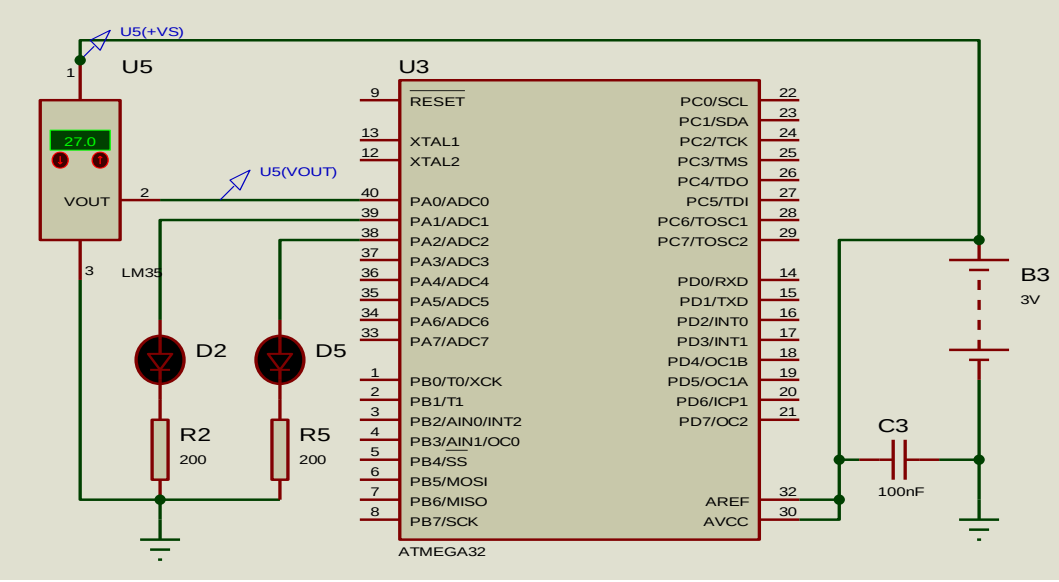


Рисунок 3.5 – Схема для считывания температуры

3.4 Контрольные вопросы

1. Какой метод аналого-цифрового преобразования сигнала применяется в ATmega32? Назовите другие типы АЦП.
2. Укажите диапазон тактовой частоты, рекомендуемый для работы АЦП ATmega32.

3. Назовите возможные режимы работы АЦП.
4. Прокомментировать результат выполнения команды ADCSRA|=0x40;.
5. Назовите способы изменения точности работы АЦП.

3.5 Задачи

Решить задачи с учётом варианта (табл. 3.6):

1. Расчёт значений преобразования по заданным опорному и входному напряжениям.
2. Настроить АЦП с параметрами.
3. Реализовать индикацию по уровням с помощью 3-4 диодов. Индикация может быть выполнена подобно эффекту эквалайзера или включению одного диода соответственно каждому уровню.
4. Продемонстрировать работу с датчиком температуры LM35. Собрать схему, запрограммировать индикацию. Провести калибровку.

Таблица 3.6 – Варианты заданий

Вариант	Задача 1			Задача 2		Задача 3	Задача 4
	Опорное напряжение, В	Входные напряжения, В	Разрядность АЦП	Частота АЦП, кГц	Номер канала (вывода)		
1	5	1, 2, 3, 5	6	125	0	Для всех вариантов	Для всех вариантов
2	5	0.5, 2.5, 4, 5	8	250	1		
3	5	1.5, 3.5, 4.5, 5	10	500	2		
4	3	0.5, 2, 2.5, 3	4	125	3		
5	3	1, 2, 2.5, 3	6	250	4		
6	3	1, 2, 2.5, 3	8	500	5		
7	12	2, 4, 11, 12	10	125	6		
8	12	1, 3, 5, 12	8	250	7		
9	12	4, 7, 9, 12	6	500	0		
10	12	2, 6, 10, 12	12	125	1		

3.6 Содержание отчёта

Отчёт выполняется в текстовом редакторе Office Word, LibreOffice Writer или подобных.

Структура отчёта:

1. Введение и условия задания.
2. Результаты решения.
3. Заключение.

Отчёт должен включать:

- условия решаемых задач, номер варианта;
- скриншоты схем;
- текст программ (не скриншоты);
- ручные вычисления (набрано печатаемым образом или написано от руки);
- описание схемы и программы для работы с датчиком LM35.

Библиографический список

1. Ефстифеев А.В. Микроконтроллеры AVR семейства Mega. Руководство пользователя / А.В. Ефстифеев. – Москва: Додэка-XXI, 2007. – 592 с.: ил.
2. AVR ATmega32 8-bit Microcontroller with 32 Kbytes In-System Programmable Flash. [Электронный ресурс]. – URL: http://ww1.microchip.com/downloads/en/DeviceDoc/doc2503.pdf. – 2020. – 356 с.
3. Козырев В.Г. Архитектура компьютерных систем: учеб. Пособие / В.Г. Козырев. – Севастополь: СевГУ, 2017. – 250 с.
4. Что такое АЦП [Электронный ресурс]. – URL: http://www.av-assembler.ru/mc/what-is-adc.php.
5. Аналого-цифровые преобразования — АЦП / О. Евсегнеев. – [Электронный ресурс]. URL: https://robotclass.ru/tutorials/arduino_adc/
6. Using the Simulator in Atmel Studio. [Электронный ресурс]. – URL: https://www.youtube.com/watch?v=9QlDSNeuAdY.
7. Microcontroller Firmware Programming Upload with ATMEL Studio. – URL: http://www.pic-control.com/microcontroller-firmware-programming-upload-with-atmel-studio/ [Дата обращения 17.02.2020].
8. Архив для Программирование AVR / коллектив авторов. [Электронный ресурс]. – URL: http://narodstream.ru/rub_avr/page/13/.
9. Программирование на AVR / коллектив авторов. [Электронный ресурс]. – URL: https://radioparty.ru/programming/avr/c.
10. Программирование AVR. Уроки / О. Евсегнеев. – [Электронный ресурс]. – URL: https://robotclass.ru/tutorials.

ТЕХНОЛОГИИ И СРЕДСТВА ПРОГРАММИРОВАНИЯ РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

Методические указания
к выполнению лабораторных работ

Составитель: **Липко** Иван Юрьевич

В авторской редакции

Изд. № 306/2020. Объем 2 п.л.
РИИЦМ ФГАОУ ВО «Севастопольский государственный университет»