МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ
«ЛУГАНСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Д.А. Капустин
Р.Н. Сентяй
С.Н. Гвоздюкова

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ

Учебное пособие
для студентов очной и заочной форм обучения
по направлению подготовки
44.03.01 «Педагогическое образование. Информатика»

КНИТА
Луганск
2021

УДК 004.4(075.8)
ББК 32.973.26-018.2я73
К20

Рецензенты:

Клиев А.А. – доцент кафедры компьютерных систем и сетей Государственного образовательного учреждения высшего образования Луганской Народной Республики «Луганский государственный университет имени Владимира Даля», кандидат технических наук, доцент;

Гавриленко П.Н. – доцент кафедры информационных образовательных технологий и систем Государственного образовательного учреждения высшего образования Луганской Народной Республики «Луганский государственный педагогический университет», кандидат технических наук, доцент;

Шмыров В.В. – доцент кафедры информационных технологий и систем Государственного образовательного учреждения высшего образования Луганской Народной Республики «Луганский государственный педагогический университет», кандидат физико-математических наук.

Капустин Д.А., Сентяй Р.Н., Гвоздюкова С.Н.

К20 **Программное обеспечение ЭВМ** : учебное пособие / Д.А. Капустин, Р.Н. Сентяй, С.Н. Гвоздюкова ; ГОУ ВО ЛНР «ЛГПУ». – Луганск : Книта, 2021. – 156 с.

В учебном пособии представлен материал, который позволит освоить структуру микроконтроллера, способы, методы подключения и программирования микроконтроллера. Учебное издание снабжено теоретическими сведениями, практическими заданиями и контрольными вопросами с целью приобретения практических навыков работы с микроконтроллерами семейства Atmel.

Предлагаемое пособие по выполнению лабораторных работ предназначено для оказания теоретической и практической помощи студентам направления подготовки 44.03.01 «Педагогическое образование» профиль подготовки «Информатика» в процессе подготовки к лабораторным работам по дисциплине: «Программное обеспечение ЭВМ».

УДК 004.4(075.8)
ББК 32.973.26-018.2я73

Рекомендовано Учебно-методическим советом ГОУ ВО ЛНР «ЛГПУ» в качестве учебного пособия для студентов, обучающихся по направлению подготовки 4403.01 «Педагогическое образование» профиль подготовки «Информатика» (протокол № 9 от 25.05.2021)

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Добавление интеллектуальных функций в современную радиоэлектронную аппаратуру базируется на использовании микропроцессоров (МП), программируемых логических интегральных схем (ПЛИС), приборов типа «система на кристалле» (System-On-Chip – SOC) и других современных цифровых устройств. Подобная интеграция позволяет автоматизировать процессы измерения, управления, контроля, регулирования и обработки информации, а также обеспечить такие свойства приборных комплексов, как многофункциональность, модифицируемость, адаптивность, обучаемость и ряд других.

Для интеграции в различные устройства применяют микроконтроллеры.

Микроконтроллер (МК, англ. Micro Controller Unit, MCU) – микросхема, которая сочетает на одном кристалле функции процессора и периферийных устройств, содержит ОЗУ или ПЗУ. По сути, это однокристальный компьютер, способный выполнять простые задачи.

Владение основами цифровой и микропроцессорной техники является неотъемлемым элементом компетенций специалистов в области радиоэлектроники и информационных технологий.

Учебная дисциплина «Программное обеспечение ЭВМ» относится к разделу базовой части блока направления подготовки 44.03.01 «Педагогическое образование» профиль подготовки «Информатика».

Предлагаемый цикл лабораторных работ предназначен для изучения принципов построения и функционирования электронно-вычислительных средств (ЭВС), построенных на базе микроконтроллеров семейства AVR с использованием программного пакета моделирования Arduimo IDE.

При выполнении лабораторных работ студенты должны практически освоить основы организации микроконтроллеров семейства AVR, которые рассматриваются в лекционном материале дисциплины и теоретических сведениях к лабораторным работам.

По каждой лабораторной работе оформляется отчет, который предоставляется преподавателю для защиты не позднее следующего лабораторного занятия. К защите лабораторной работы студент должен подготовить ответу на контрольные вопросы.

Лабораторная работа № 1.
Широтно-импульсная модуляция в микроконтроллерах Atmega

В этой лабораторной узнаем о широтно-импульсной модуляции, о реализации этого способа управления в микроконтроллерах Atmega328, о режимах и функциях работы с ШИМ в среде програмирования Arduino IDE. В итоге, мы напишем программу для микроконтроллера, которая будет управлять уровнем аналогового сигнала на выходе микросхемы LM317T.

Цель работы: приобрести практические навыки по управлению уровнем аналогового сигнала.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программы, указанному в примере.
4. Выполнить задание для самостоятельной работы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.

Теоретические сведения

Широтно-импульсная модуляция. Широтно-импульсная модуляция (ШИМ) это способ управления мощностью на нагрузке с помощью изменения скважности импульсов при постоянной амплитуде и частоте импульсов.

Можно выделить две основные области применения широтно-импульсной модуляции:

– во вторичных источниках питания, различных регуляторах

мощности, регуляторах яркости источников света, скорости вращения коллекторных двигателей и т.п. В этих случаях применение ШИМ позволяет значительно увеличить КПД системы и упростить ее реализацию;

– для получения аналогового сигнала с помощью цифрового выхода микроконтроллера. Своеобразный цифро-аналоговый преобразователь (ЦАП). Очень простой в реализации, требует минимума внешних компонентов. Часто достаточно одной RC цепочки.

Принцип регулирования с помощью ШИМ – изменение ширины импульсов при постоянной амплитуде и частоте сигнала.

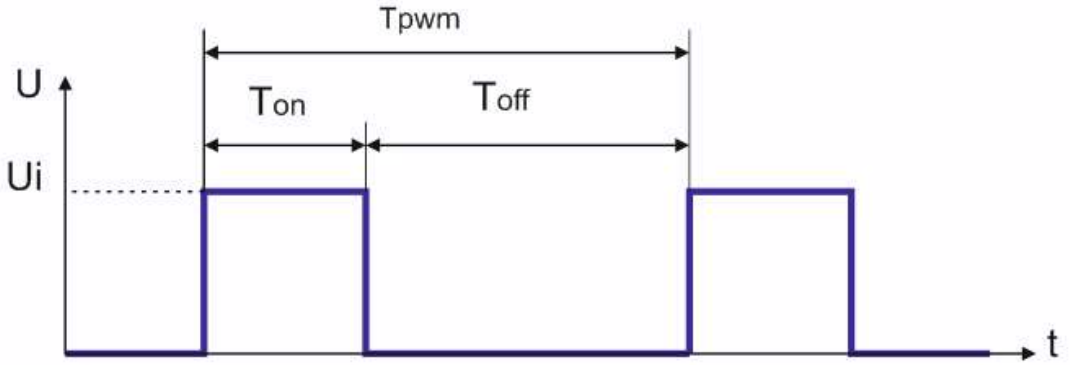


Рис. 1.1. Основные параметры ШИМ сигнала

На диаграмме (рис. 1.1) можно увидеть основные параметры ШИМ сигнала:

– Ui – амплитуда импульсов;

– Ton – время активного (включенного) состояния сигнала;

– Toff – время отключенного состояния сигнала;

– Tpwm – время периода ШИМ.

Даже интуитивно понятно, что мощность на нагрузке пропорциональна соотношению времени включенного и отключенного состояния сигнала.

Это соотношение определяет коэффициент заполнения ШИМ: Kw = Ton / Tpwm.

Он показывает, какую часть периода сигнал находится во включенном состоянии. Может меняться:

– от 0 – сигнал всегда выключен;

– до 1 – сигнал все время находится во включенном состоянии.

Чаще используют процентный коэффициент заполнения. В этом случае он находится в пределах от 0 до 100% (рис. 1.2).

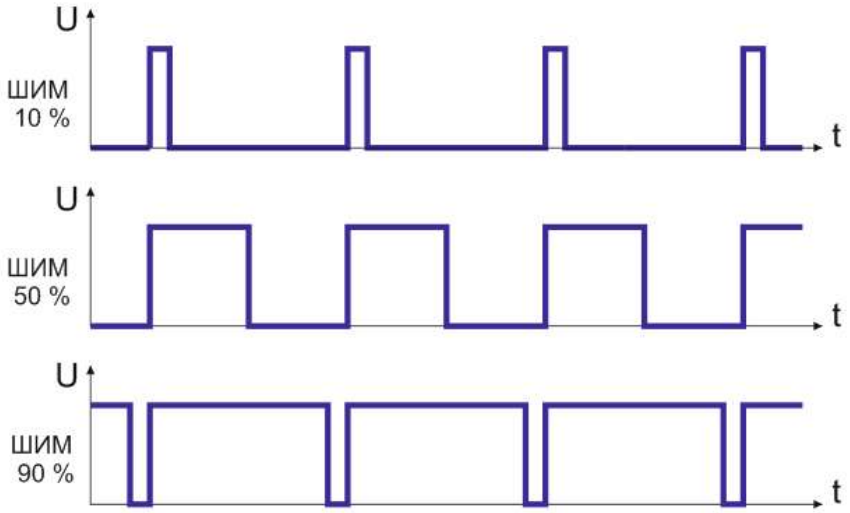


Рис. 1.2. ШИМ сигнал с различным коэффициентом заполнения

Значение электрической мощности на нагрузке строго пропорционально коэффициенту заполнения. Когда говорят, что ШИМ равен, например, 20%, то имеют в виду именно коэффициент заполнения.

Если сигнал ШИМ пропустить через фильтр низких частот (ФНЧ), то на выходе фильтра мы получим аналоговый сигнал, напряжение которого пропорционально коэффициенту заполнения ШИМ: U = Kw * Ui.

В качестве ФНЧ можно использовать простейшую RC цепочку (рис. 1.3).

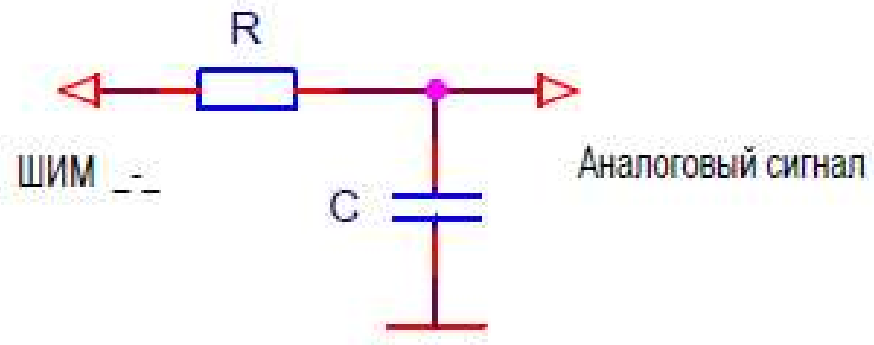


Рис. 1.3. RC цепочка

Из-за неидеальной характеристики такого фильтра частота среза должна быть минимум на порядок меньше частоты ШИМ. Для простого RC фильтра частота среза вычисляется по формуле:

$F = 1 / (2 \pi R C)$.

При повышении частоты среза ФНЧ на выходе фильтра увеличиваются пульсации с частотой ШИМ. При уменьшении частоты среза фильтра снижается время реакции выходного аналогового сигнала на изменения ширины импульсов.

Из этого вытекает главный недостаток широтно-импульсной модуляции. Метод способен синтезировать только достаточно медленные аналоговые сигналы или требует применения фильтров низких частот с высокой добротностью, сложных в реализации.

В связи с этим рекомендуют:

– в случае, когда к быстродействию аналогового сигнала жестких требований нет выбирать заведомо заниженную частоту среза фильтра;

– если необходимо оптимизировать быстродействие аналогового преобразователя, то лучше промоделировать схему.

Даже простейшие моделирующие программы вычисляют уровень пульсаций достаточно точно. Вот результаты моделирования на SwCAD для ШИМ частотой 500 Гц и RC фильтрами с частотами среза 500 Гц, 50 Гц и 5 Гц. Зеленым цветом показана диаграмма ШИМ, синим – напряжение на выходе RC фильтра (рис. 1.4 – 1.6).

Точность преобразования широтно-импульсных модуляторов определяется погрешностью амплитуды импульсов (т.е. стабильностью питания микроконтроллера) и значением падения напряжения на ключах цифровых выходов микроконтроллера. Как правило, точность ШИМ микроконтроллеров невысока. Добиться высокой точности ШИМ преобразования можно с помощью дополнительной схемы с аналоговыми ключами и источником опорного напряжения.

К недостаткам использования широтно-импульсных модуляторов в качестве ЦАП также следует отнести высокое выходное сопротивление. Оно определяется сопротивлением резистора RC фильтра и не может быть низким из-за малой нагрузочной способности выходов микроконтроллера.

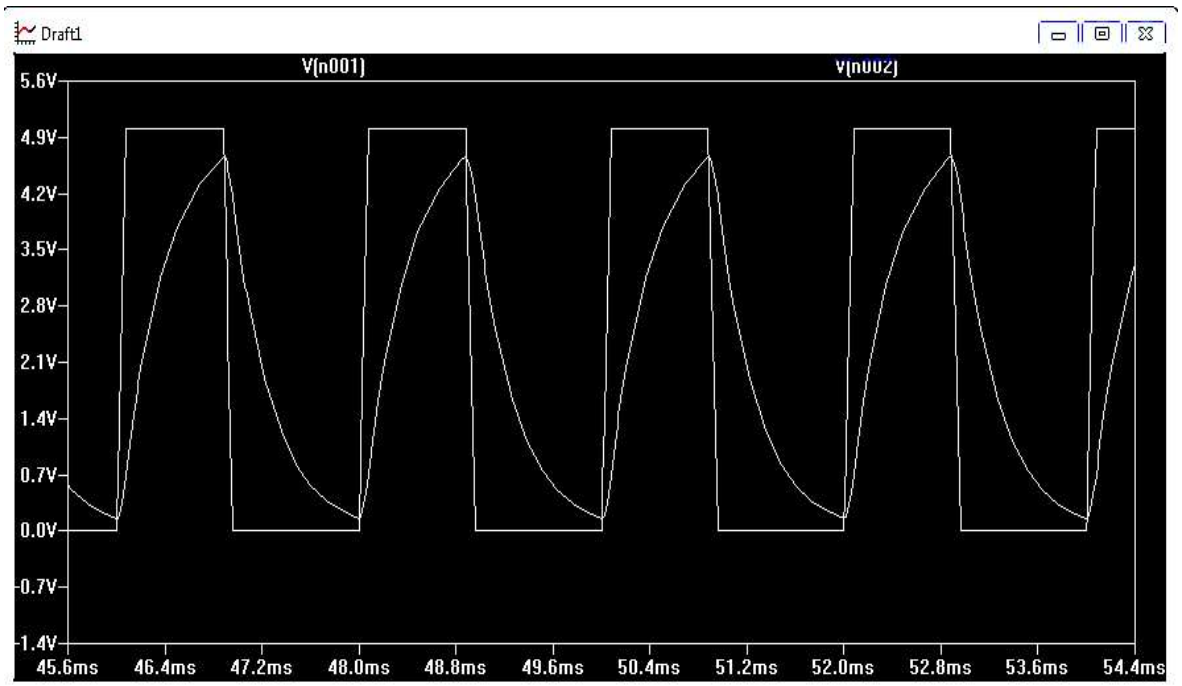


Рис. 1.4. Диаграмма ШИМ сигнала для частоты среза RC цепи 500 Гц (R=10 кОм, C=32 нФ).

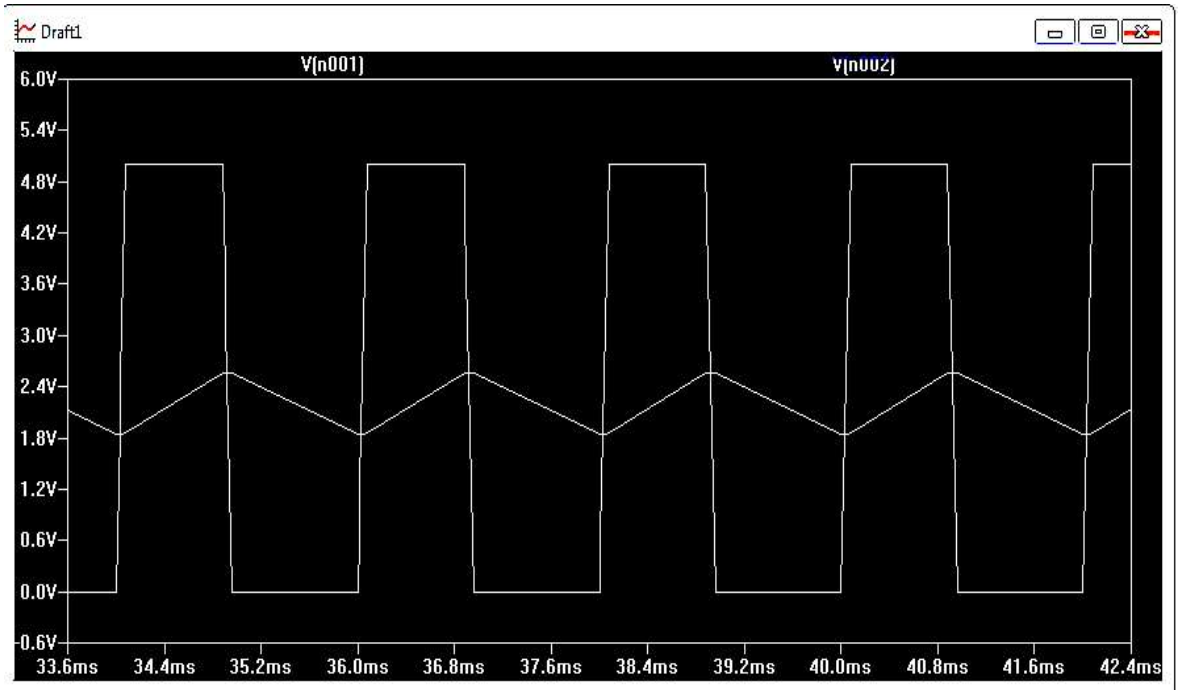


Рис. 1.5. Диаграмма ШИМ сигнала для частоты среза RC цепи 50 Гц (R=10 кОм, C=320 нФ)

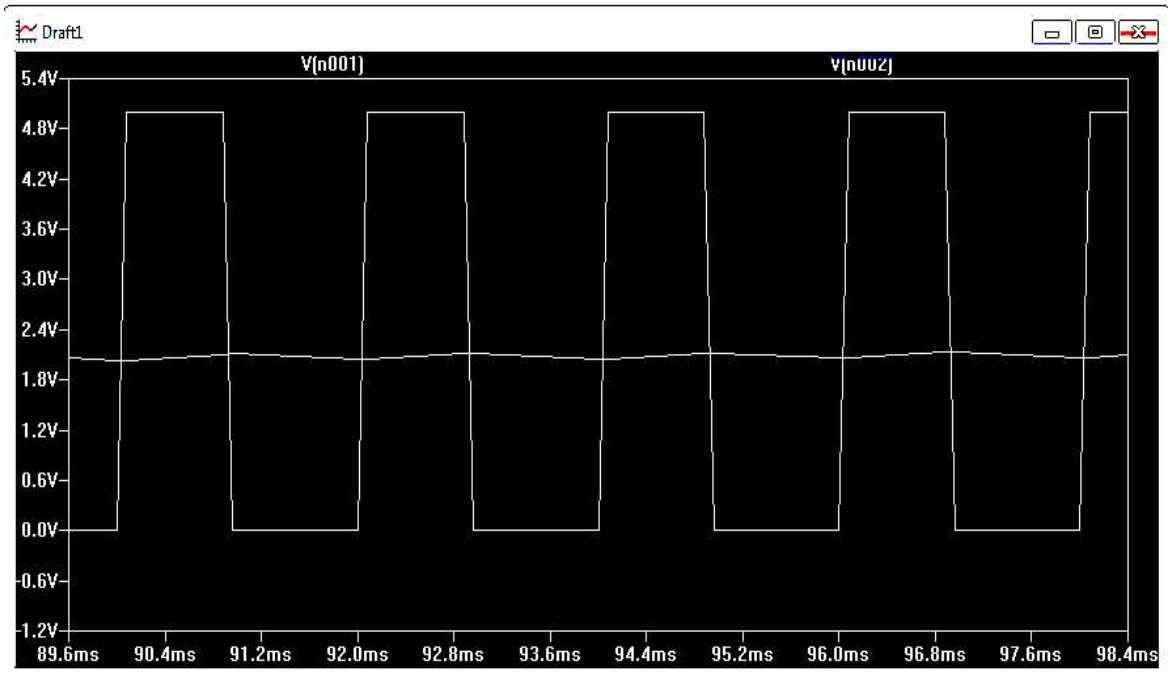


Рис. 1.6. Диаграмма ШИМ сигнала для частоты среза RC цепи 5 Гц (R=10 кОм, C=3,2 мкФ)

Широтно-импульсные модуляторы в среде Arduino IDE. Платы Arduino на базе микроконтроллеров ATmega168/328 имеют 6 аппаратных широтно-импульсных модуляторов. Сигналы ШИМ могут быть сгенерированы на выводах 3, 5, 6, 9, 10, 11.

Управление аппаратными ШИМ осуществляется с помощью системной функции analogWrite().

Void analogWrite (pin, val). Функция переводит вывод в режим ШИМ и задает для него коэффициент заполнения. Перед использованием analogWrite() функцию pinMode() для установки вывода в режим «выход» вызывать необязательно.

Аргументы:

– pin – номер вывода для генерации ШИМ сигнала;

– val – коэффициент заполнения ШИМ. Без дополнительных установок диапазон val от 0 до 255 и соответствует коэффициенту заполнения от 0 до 100 %. Т.е. разрядность системных ШИМ в Arduino 8 разрядов.

analogWrite (9, 25); // на выводе 9 ШИМ = 10%

Частота ШИМ Arduino UNO 488,28 Гц.

Для генерации ШИМ используются все три таймера Arduino UNO (табл. 1.1).

Таблица 1.1. Сопоставление таймеров микроконтроллера Atmega328 с выводами платы Arduino UNO

Таймер	Используется для генерации ШИМ на выводах
Таймер 0	выводы 5 и 6
Таймер 1	выводы 9 и 10
Таймер 2	выводы 3 и 11

Если таймер используется для других целей, например, для прерывания, то параметры ШИМ соответствующих выводов могут не соответствовать указанным выше.

Поэтому, при использовании библиотек MsTimer2, TimerOne или им подобных некоторые выводы в качестве ШИМ сигналов использовать нельзя.

Увеличение частоты и разрядности ШИМ Arduino. Система Arduino устанавливает на всех выводах ШИМ параметры:

– частота 488,28 Гц;

– разрешение 8 разрядов (0…255).

Очень низкая частота. Для большинства приложений совершенно не допустимая.

В разработке контроллера элемента Пельтье, начатой в предыдущем уроке, частота ШИМ должна быть не менее 30-50 кГц. В интернете достаточно много предложений по увеличению частоты ШИМ во всех описываются методы увеличения частоты до 31 кГц. В принципе приемлемый вариант, но можно добиться и большего.

В таблице 1.2 сведены все возможные варианты ШИМ для таймера 1 в таблицу.

Строчки из правого столбца для выбранного варианта необходимо написать в функции setup().

Следующий пример генерирует на выводе 9 ШИМ с частотой 62,5 кГц и коэффициентом заполнения примерно 10 %.

Пример:

```
void setup () {
// ШИМ 8 разрядов, 62,5 кГц
TCCR1A = TCCR1A & 0xE0 | 1;
TCCR1B = TCCR1B & 0xE0 | 0x09;
}
void loop () {
analogWrite (9, 25); //на выводе 9 ШИМ=10%
}
```

Таблица 1.2. Варианты параметров ШИМ на выводах 9 и 10 Arduino UNO (таймер 1)

Разрешение	Частота ШИМ	Команды установки режима
8 бит	62 500 Гц	TCCR1A = TCCR1A & 0xe0 | 1; TCCR1B = TCCR1B & 0xe0 | 0x09;
	7 812,5 Гц	TCCR1A = TCCR1A & 0xe0 | 1; TCCR1B = TCCR1B & 0xe0 | 0x0a;
	976,56 Гц	TCCR1A = TCCR1A & 0xe0 | 1; TCCR1B = TCCR1B & 0xe0 | 0x0b;
	244,14 Гц	TCCR1A = TCCR1A & 0xe0 | 1; TCCR1B = TCCR1B & 0xe0 | 0x0c;
	61,04 Гц	TCCR1A = TCCR1A & 0xe0 | 1; TCCR1B = TCCR1B & 0xe0 | 0x0d;
9 бит	31 250 Гц	TCCR1A = TCCR1A & 0xe0 | 2; TCCR1B = TCCR1B & 0xe0 | 0x09;
	3 906,25 Гц	TCCR1A = TCCR1A & 0xe0 | 2; TCCR1B = TCCR1B & 0xe0 | 0x0a;
	488,28 Гц	TCCR1A = TCCR1A & 0xe0 | 2; TCCR1B = TCCR1B & 0xe0 | 0x0b;
	122,07 Гц	TCCR1A = TCCR1A & 0xe0 | 2; TCCR1B = TCCR1B & 0xe0 | 0x0c;
	30,52 Гц	TCCR1A = TCCR1A & 0xe0 | 2; TCCR1B = TCCR1B & 0xe0 | 0x0d;
10 бит	1 5625 Гц	TCCR1A = TCCR1A & 0xe0 | 3; TCCR1B = TCCR1B & 0xe0 | 0x09;
	1 953,13 Гц	TCCR1A = TCCR1A & 0xe0 | 3; TCCR1B = TCCR1B & 0xe0 | 0x0a;
	244,14 Гц	TCCR1A = TCCR1A & 0xe0 | 3; TCCR1B = TCCR1B & 0xe0 | 0x0b;
	61,04 Гц	TCCR1A = TCCR1A & 0xe0 | 3; TCCR1B = TCCR1B & 0xe0 | 0x0c;
	15,26 Гц	TCCR1A = TCCR1A & 0xe0 | 3; TCCR1B = TCCR1B & 0xe0 | 0x0d;

Это максимально возможная частота ШИМ Arduino для большинства плат (с частотой генератора 16 мГц).

Управление микросхемой LM317T с помощью ШИМ сигнала. Популярный регулируемый линейный стабилизатор напряжения LM317T фирмы National Semiconductor обеспечивает выходное напряжение 1.25 … 37 В при токе до 1.5 А. Обычно для установки напряжения используют потенциометр. На рис. 1.7 показано, как заменить потенциометр аналоговым напряжением, формируемым из

сигнала ШИМ (ШИМ – широтно-импульсная модуляция). Параметры ШИМ могут устанавливаться внешним микроконтроллером или иной цифровой схемой. Если использовать микроконтроллер еще и для динамического мониторинга выхода LM317T, получится замкнутая система регулирования.

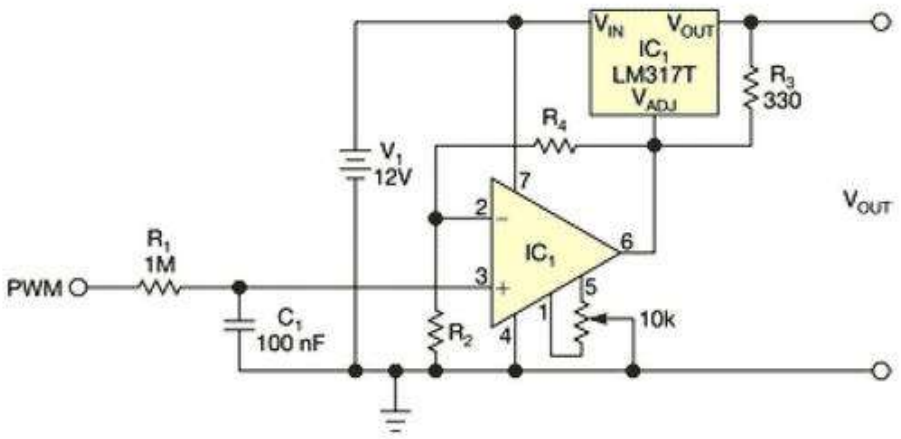


Рис. 1.7. В этой схеме потенциометр заменен аналоговым напряжением, получаемым из сигнала ШИМ

С помощью RC фильтра низких частот и операционного усилителя (ОУ) можно преобразовать сигнал ШИМ в постоянное напряжение и регулировать этим напряжением выходной уровень микросхемы LM317T. Изменение скважности импульсов ШИМ дает возможность генерировать аналоговое напряжение на выходе фильтра в диапазоне от 0 до 5 В. ОУ усиливает это напряжение для получения требуемого диапазона.

Если усилить выходное напряжение вдвое, на вывод регулировки LM317T будет подаваться напряжение 0 … 10 В, при этом на выходе схемы напряжение будет меняться от 1.25 до 11.25 В, в соответствии с уравнением: $V_{OUT} = V_{ADJ} + 1.25$ В.

Меняя значения сопротивлений резисторов R_4 и R_2, можно получить требуемое усиление. Если вам захочется уменьшить напряжение смещения, воспользуйтесь ОУ, имеющим выводы подстройки нуля, например, LM741. Выбор значений емкости C_1 и сопротивления R_1 фильтра определяется частотой сигнала ШИМ. Номиналы, показанные на схеме, рассчитаны для частоты 1 кГц.

Схему можно усовершенствовать, заменив RC фильтр

активным фильтром и заведя сигнал обратной связи с выхода схемы на микроконтроллер для динамического регулирования напряжения.

Задание для самостоятельной работы

1. Запрограммируйте 9-й вывод отладочной платы Arduino UNO. со всеми вариантами параметров ШИМ из табл. 1.2 (скважность 10%, 50%, 90%)
2. При помощи осциллографа сравнить полученные на выходе сигналы с устанавливаемыми, сделать выводы.

Контрольные вопросы

1. Дайте определение широтно-импульсной модуляции.
2. В каких областях применяется ШИМ?
3. Опишите основные параметры ШИМ сигнала.
4. Назовите основные недостатки ШИМ.
5. С помощью какой системной функции осуществляется управление аппаратным ШИМ?

Лабораторная работа № 2.
Изучение принципов работы двигателя постоянного тока и принципа программного управления для систем автоматического управления

В этой лабораторной разберем популярную, широко распространенную миросхему L293D. Изучим принципы работы двигателя постоянного тока и принцип программного управления драйвером L293D. Програмное управление будет осуществлятся микроконтроллером Atmega328 в среде програмирования Arduino IDE.

В итоге, мы напишем программу для микроконтроллера, которая будет управлять двигателями постоянного тока, сможем менять направление вращения и скорость вращения валов двигателей.

Цель работы: приобрести практические навыки по управлению двигателя постоянного тока.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программы, указанному в примере.
4. Выполнить задание для самостоятельной работы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.

Теоретические сведения

Драйвер двигателей L293D. Микросхема включает в себя

сразу два драйвера для управления слаботочными моторами.

Принцип работы каждого из драйверов, входящих в состав микросхемы, идентичен, поэтому рассмотрим принцип работы одного из них (рис. 2.1).

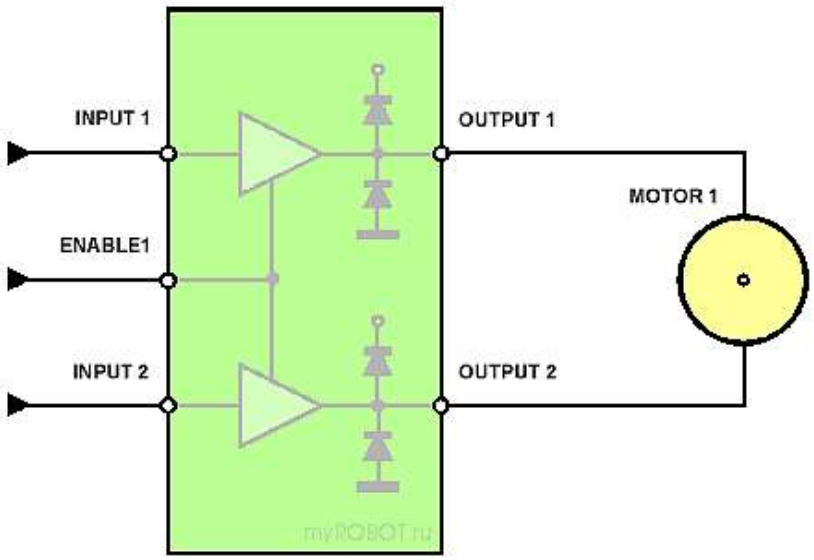


Рис. 2.1. Внутреннее устройство драйвера L293D

К выходам OUTPUT1 и OUTPUT2 подключим электромотор MOTOR1.

На вход ENABLE1, включающий драйвер, подадим сигнал (соединим с положительным полюсом источника питания +5V). Если при этом на входы INPUT1 и INPUT2 не подаются сигналы, то мотор вращаться не будет.

Если вход INPUT1 соединить с положительным полюсом источника питания, а вход INPUT2 – с отрицательным, то мотор начнет вращаться.

Теперь попробуем соединить вход INPUT1 с отрицательным полюсом источника питания, а вход INPUT2 – с положительным. Мотор начнет вращаться в другую сторону.

Попробуем подать сигналы одного уровня сразу на оба управляющих входа INPUT1 и INPUT2 (соединить оба входа с положительным полюсом источника питания или с отрицательным) – мотор вращаться не будет.

Если мы уберем сигнал с входа ENABLE1, то при любых вариантах наличия сигналов на входах INPUT1 и INPUT2 мотор вращаться не будет.

Представить лучше принцип работы драйвера двигателя можно, рассмотрев следующую таблицу 2.1.

Таблица 2.1. Таблица истинности для драйвера L293D

ENABLE1	INPUT1	INPUT2	OUTPUT1	OUTPUT2
1	0	0	0	0
1	1	0	1	0
1	0	1	0	1
1	1	1	1	1

Теперь рассмотрим назначение выводов микросхемы L293D (рис. 2.2).

– Входы ENABLE1 и ENABLE2 отвечают за включение каждого из драйверов, входящих в состав микросхемы.

– Входы INPUT1 и INPUT2 управляют двигателем, подключенным к выходам OUTPUT1 и OUTPUT2.

– Входы INPUT3 и INPUT4 управляют двигателем, подключенным к выходам OUTPUT3 и OUTPUT4.

– Контакт Vs соединяют с положительным полюсом источника электропитания двигателей или просто с положительным полюсом питания, если питание схемы и двигателей единое. Проще говоря, этот контакт отвечает за питание электродвигателей.

– Контакт Vss соединяют с положительным полюсом источника питания. Этот контакт обеспечивает питание самой микросхемы.

– Четыре контакта GND соединяют с «землей» (общим проводом или отрицательным полюсом источника питания). Кроме того, с помощью этих контактов обычно обеспечивают теплоотвод от микросхемы, поэтому их лучше всего распаивать на достаточно широкую контактную площадку.

Несомненным плюсом данной микросхемы является раздельное питание логической части микросхемы, напряжение питания которой лежит в приделах 4.5-5 вольт (VSS), и силовой части питания двигателей (VS).

Используя данную микросхему, мы можем управлять двигателями с довольно широким диапазоном питающего напряжения от 4.5 до 36 вольт, но при этом, L293D может выдать всего лишь 600mA продолжительного тока нагрузки на каждый канал. Пиковый (максимальный) ток может кратковременно подскочить до 1.2A.

Так же из положительных сторон данной микросхемы следует отметить её не привередливость к напряжению входных сигналов, подаваемых на выводы INPUT.

Логический «0» распознается микросхемой, когда входное напряжение <1.5Вольт.

Логическая «1» появляется при входном напряжении, лежащем в пределах от 2.3 до 7Вольт.

Диапазон рабочих температур от -40°C до +150°C

Скорость переключения до 5 kHz

Вывод ENABLE1 это главная фигура в управлении левым каналом, без лога единицы на его выводе (или ШИМ, об этом чуть позже) ничего работать не будет, вне зависимости от того, что творится на выводах INPUT1 и INPUT2.

Выводы INPUT1 и INPUT2 задают направление вращения мотора. Их можно сравнить с рулем машины, тем более что в данном случае сравнение подходит идеально, ведь мы не можем повернуть руль сразу в две стороны, а необходимо выбирать одну из двух. Из вышесказанного следует, что для поворота нам надо подать логическую единицу на вывод INPUT1, а на INPUT2 подать логический ноль. Для смены направления поменять местами INPUT1 «0», INPUT2 «1».

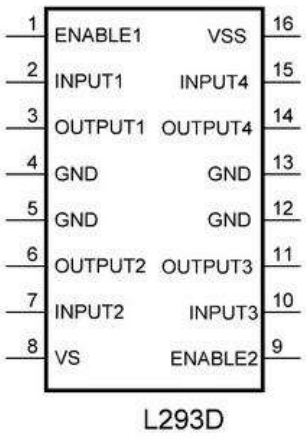


Рис. 2.2. Назначение выводов микросхемы L293D

При подаче одинаковых логических сигналов на входы IN1 и IN2, мотор вращаться не будет, следовательно, вращение можно остановить либо подачей логического нуля на вывод ENABLE1, при любой конфигурации IN1 и IN2, либо одинаковыми логическими сигналами на IN1 и IN2, не изменяя конфигурации вывода EN1 (данный вариант мы и рассмотрим ниже).

Контакты GND соединяются с отрицательным полюсом источника питания (земля).

Оставшиеся выводы OUTPUT1 и OUTPUT2 служат непосредственно для подключения мотора. Правый канал работает абсолютно идентично.

Рассмотрим самый простой вариант подключения L293D рис. 2.2. Скорость вращения при таком варианте двигателя нерегулируемая, вывод EN1 подключен напрямую к +5V. Питание мотора также берется от стабилизатора, установленного на Arduino. Питать таким образом от USB можно только очень слабые нагрузки (в данном случае потребляемый ток моторчика 100mA, и нет никаких внешних воздействий способных повысить потребляемый ток), и то такое подключение крайне нежелательно (рис. 2.3).

Заставим моторчик вращаться «вправо» 4 секунды, остановиться на 0.5 секунды, вращаться «влево» 4 секунды, остановка 5 секунд и снова цикл повторяется.

Пример:

```
int IN1 = 7; // Input1 подключён к выводу 7
int IN2 = 8; // Input2 подключён к выводу 7

void setup ()
{
pinMode (IN2, OUTPUT); // Input2 настроен на выход
pinMode (IN1, OUTPUT); // Input1 настроен на выход
}
void loop ()
{
digitalWrite (IN2, HIGH); // Вращать "вправо" 4 секунды
digitalWrite (IN1, LOW);
delay (4000);
digitalWrite (IN2, LOW); // Остановить на 0.5 секунды
delay (500);
digitalWrite (IN1, HIGH); // Вращать "влево" 4 секунды
delay (4000);
digitalWrite (IN1, LOW); // Остановить на 5 секунд
delay (5000);
}
```

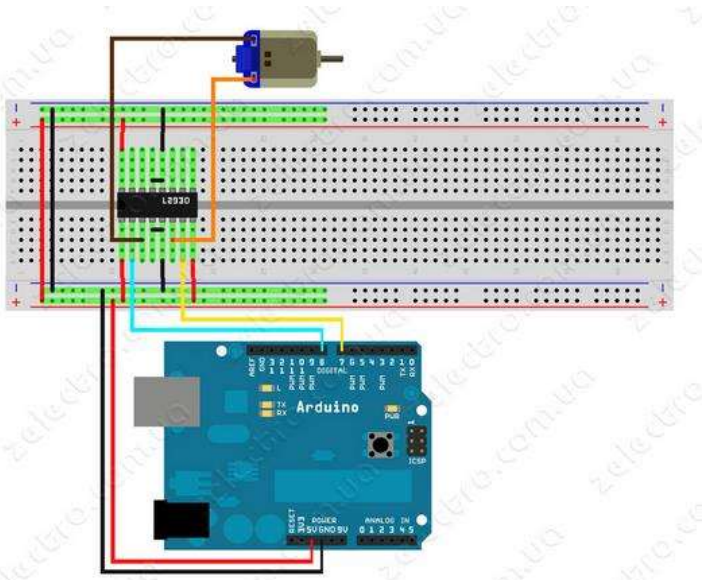


Рис. 2.3. Схема подключения L293D для одного двигателя

Теперь усложним нашу схему. При использовании мотора с рабочим током выше 100мА настоятельно рекомендуем использовать внешний источник питания силовой части. Также, в следующем примере, подсоединим вывод EN1 к ШИМ (PWM) выводу Arduino под номером 6.

В данном примере мы подсоединили вывод EN1 к выводу ШИМ. Задействуем возможность драйвера управлять скоростью, меняя скважность посылаемого ШИМ сигнала (рис. 2.4). Значения скважности задаются функцией ***analogWrite* (*pin, число*).**

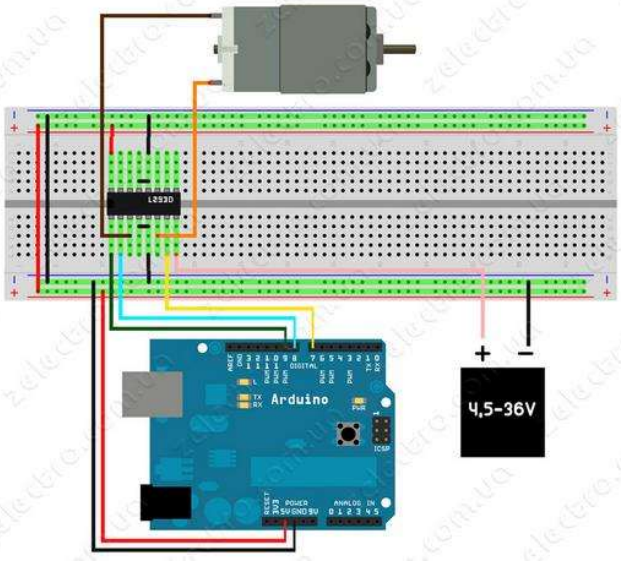


Рис. 2.4. Схема подключения 2-х двигателей с регулировкой скорости вращения

Где число изменяется от 0 до 255, прямо пропорционально скважности сигнала. Для наглядности, были подобраны четыре значения, при которых двигатель стартует с низких оборотов, набирает средние, выходит на максимальные и останавливается.

Пример:

```
int IN1 = 7; // Input1 подключён к выводу 7
int IN2 = 8; // Input2 подключён к выводу 7
int EN1 = 6; // En1 подключён к выводу 6
void setup ()
{
pinMode (IN2, OUTPUT); // Input2 настроен на выход
 pinMode (IN1, OUTPUT); // Input1 настроен на выход
 pinMode (EN1, OUTPUT); // En1 настроен на выход
 }
 void loop ()
 {
 digitalWrite (IN2, HIGH);
 digitalWrite (IN1, LOW);
 analogWrite (EN1, 63); // Скорость вращения 25%
 delay (2000); //в течении двух секунд
 analogWrite (EN1, 127); // Скорость вращения 50%
 delay (2000); //в течении двух секунд
 analogWrite (EN1, 255); // Скорость вращения 100%
 delay (2000); //в течении двух секунд
 analogWrite (EN1, 0);
 // Принудительная остановка двигателя
 delay (5000); //на пять секунд
 }
```

Подключим два двигателя с регулировкой скорости (рис. 2.5).

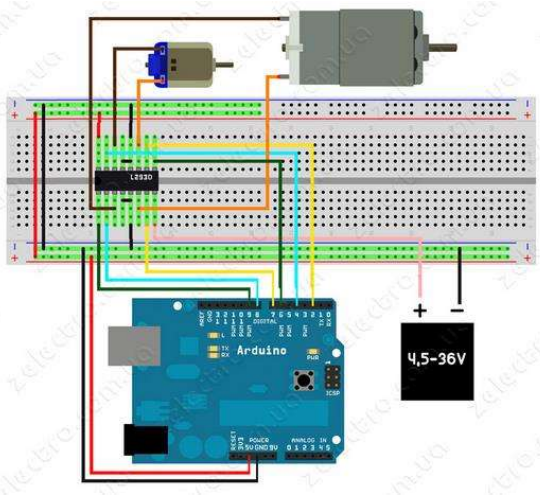


Рис. 2.5. Схема подключения 2-х двигателей с регулировкой скорости вращения

Пример:

```
int IN1 = 7; // Input1 подключён к выводу 7
int IN2 = 8; // Input2 подключён к выводу 8
int IN3 = 10; // Input1 подключён к выводу 10
int IN4 = 13; // Input2 подключён к выводу 13
int EN1 = 6; // En1 подключён к выводу 6
int EN2 = 9; // En2 подключён к выводу 9
int i;
void setup ()
{
pinMode (IN1, OUTPUT); // Input1 настроен на выход
pinMode (IN2, OUTPUT); // Input2 настроен на выход
pinMode (IN3, OUTPUT); // Input3 настроен на выход
pinMode (IN4, OUTPUT); // Input4 настроен на выход
pinMode (EN1, OUTPUT); // En1 настроен на выход
pinMode (EN2, OUTPUT); // En2 настроен на выход
}
void loop ()
{
digitalWrite (IN2, HIGH);
digitalWrite (IN1, LOW);
digitalWrite (IN4, HIGH);
digitalWrite (IN3, LOW);
for (i = 50; i <= 180; ++i) {//Плавное увеличение
analogWrite (EN1, i); // скорости вращения двух
analogWrite (EN2, i); //двигателей
delay (30);
}
analogWrite (EN1, 0); // Остановка двух двигателей
analogWrite (EN2, 0); // на 0,5 секунды
delay (500);
digitalWrite (IN1, HIGH);
digitalWrite (IN2, LOW);
digitalWrite (IN3, HIGH);
digitalWrite (IN4, LOW);
for (i = 50; i <= 180; ++i) {// Плавное увеличение
analogWrite (EN1, i); //скорости вращения двух
analogWrite (EN2, i); //двигателей в другую сторону
delay (30);
}
analogWrite (EN1, 0); // Остановка двух двигателей
analogWrite (EN2, 0);
delay (8000); // на 8 секунд
}
```

Задание для самостоятельной работы

1. Запрограммируйте отладочную плату Arduino UNO вместе с платой (к лабораторной работе №2), чтобы при нажатии левой кнопки (вывод A1) первый двигатель вращался влево, а при нажатии правой кнопки (вывод A2) второй двигатель вращался вправо. При нажатии на кнопки должен загораться светодиод (вывод A0).
2. Запрограммируйте отладочную плату Arduino UNO вместе с платой (к лабораторной работе №2), чтобы при нажатии левой кнопки (вывод A1) оба двигателя вращался влево, а при нажатии правой кнопки (вывод A2) оба двигателя вращался вправо. При нажатии на кнопки должен загораться светодиод (вывод A0).

Контрольные вопросы

1. Назовите принципы работы двигателя постоянного тока.
2. Что включает в себя принцип программного управления драйвером L293D?
3. Отчего зависит направление вращения вала двигателя?
4. Как можно изменить скорость вращения валов двигателя?
5. Назовите с помощью какой команды можно включать светодиод на модельной плате?

Лабораторная работа № 3.
Изучение принципов работы шагового двигателя и принципа программного управления для систем автоматического управления

В этой лабораторной разберем популярную, широко распространенную миросхему L293N. Изучим принципы работы шагового двигателя и принцип программного управления драйвером L293D. Програмное управление будет осуществлятся микроконтроллером Atmega328 в среде програмирования Arduino IDE.

В итоге, мы напишем программу для микроконтроллера, которая будет управлять шаговым двигателем, сможем менять направление вращения и скорость вращения валов двигателей.

Цель работы: приобрести практические навыки по управлению шаговыми двигателями.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программы указанному в примере.
4. Выполнить задание для самостоятельной работы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.

Теоретические сведения

Драйвер двигателей L293N. С помощью драйвера L298N подключим к плате Arduino биполярный шаговый двигатель. Для

управления будем использовать программы из предыдущих уроков для униполярных двигателей.

Микросхема включает в себя сразу два драйвера для управления слаботочными моторами.

Принцип работы каждого из драйверов, входящих в состав микросхемы, идентичен, принципу работы драйверов микросхемы L298D, рассмотренный в предыдущей лабораторной работе.

Использование шаговых двигателей в биполярном режиме дает:

– повышение крутящего момента примерно на 40% по сравнению с униполярным двигателем;

– позволяет применять двигатели с любой конфигурацией фазных обмоток.

Недостаток биполярного режима – более сложный драйвер.

Драйвер биполярного шагового двигателя.

У биполярного шагового двигателя две обмотки, по одной для каждой фазы (рис. 3.1).

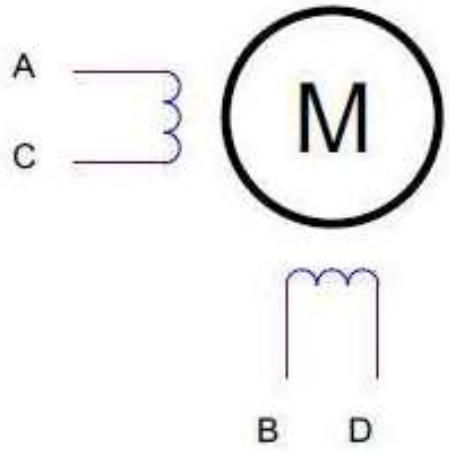


Рис. 3.1. Изображение шагового двигателя на схемах.

Если для управления униполярным двигателем достаточно 4 ключей, замыкающих выводы на землю, то биполярный привод требует более сложной коммутации обмоток. Необходимо каждую обмотку:

– подключать к источнику питания в прямой полярности;

– отключать;

– подключать к источнику в противоположной полярности.

Такую коммутацию может обеспечить мостовая схема с четырьмя ключами.

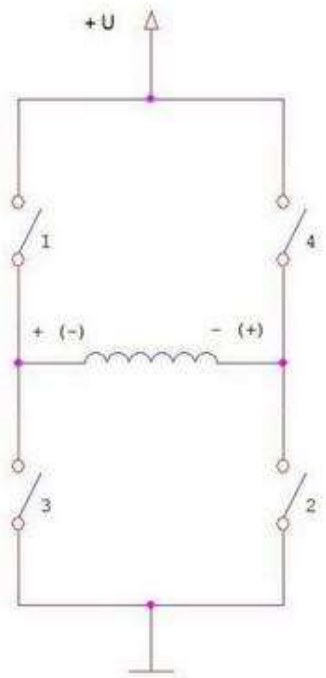


Рис. 3.2. Мостовая схема включения обмотки шагового двигателя с четырьмя ключами

При замыкании ключей 1 и 2 (рис. 3.2) на обмотку подается напряжение питания в прямом направлении. Замыкание ключей 3 и 4 подключает источник питания в обратной полярности.

Драйвер биполярного шагового двигателя намного сложнее, чем драйвер униполярного привода:

– требуется 4 ключа на обмотку, т.е. 8 ключей на двигатель;

– необходимы сложные схемы управления верхними ключами (ключи 1, 4) от логических сигналов микроконтроллера, «привязанных» к земле;

– существуют проблема сквозных токов. Они возникают при одновременном включении транзисторов из одного плеча (ключей 1, 3 или 2, 4). Это может привести к замыканию источника питания и выгоранию ключей;

– сквозные токи могут появляться из-за неодинакового быстродействия верхних и нижних ключей. Например, верхний ключ уже открылся, а нижний не успел закрыться.

Поэтому реализовать схему мощного драйвера биполярного шагового двигателя с использованием дискретных элементов достаточно сложно. Гораздо практичнее, удобнее, дешевле использовать интегральный драйвер.

Драйвер биполярного шагового двигателя L298N. Микросхема L298, наверное, самый распространенный биполярный драйвер.

Это полный мостовой драйвер, позволяющий управлять биполярными нагрузками с током до 2 А и максимальным напряжением 46 В. Драйвер разработан для управления компонентами с индуктивными нагрузками, такими как электромагниты, реле, шаговые двигатели. Сигналы управления имеют TTL совместимые уровни. Два входа разрешения дают возможность отключать нагрузку независимо от входных сигналов микросхемы. Предусмотрена возможность подключения внешних датчиков тока для защиты и контроля тока каждого моста. Питание логической схемы и нагрузки L298N разделены. Это позволяет подавать на нагрузку напряжение другой величины, чем питание микросхемы. Микросхема имеет защиту от перегрева на уровне + 70 °С (рис. 3.3).

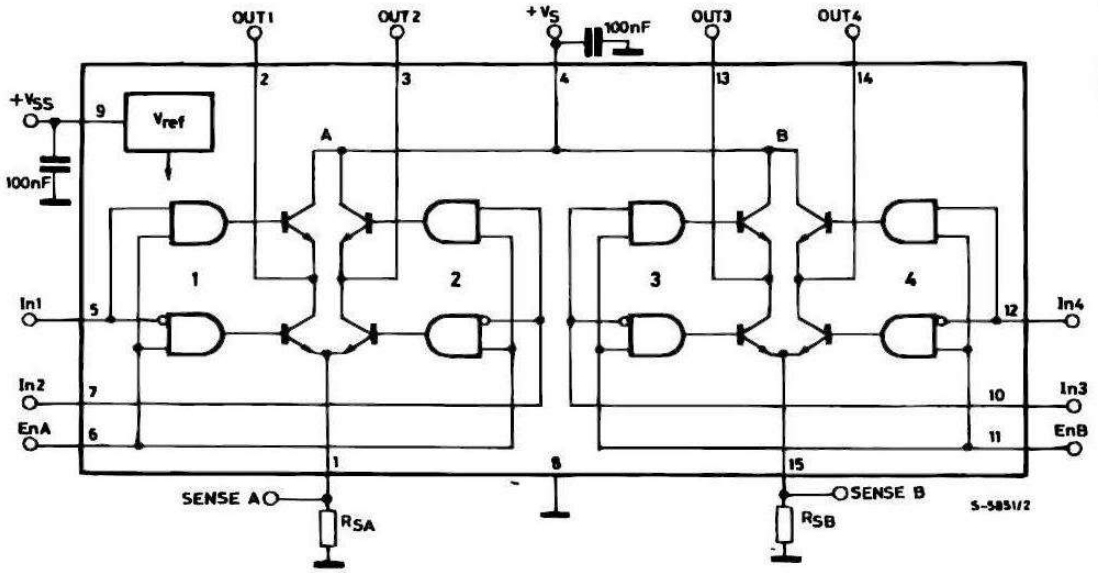


Рис. 3.3. Структурная схема L298N

Микросхема выполнена в 15-ти выводном корпусе с возможностью крепления радиатора охлаждения (таблица 3.1).

Таблица 3.1. Предельно допустимые параметры L298N

Обозначение	Параметр	Значение
Vs	Напряжение питания	50 В
Vss	Напряжение питания логики	7 В
Vi, Ven	Напряжение логических входов	-0,3...7 В
Io	Выходной ток (для каждого канала) не повторяющийся импульс 100 мкс импульсы (80% включен, 20% выключен, включен 10 мс) при постоянном токе	 3 А 2,5 А 2 А
Vsens	Напряжение датчиков тока	-1...2,3 В
Ptot	Мощность рассеивания (температура корпуса 75°C)	25 Вт
Top	Рабочая температура кристалла	-25...130°C
Tstg	Температура хранения	-40...150°C

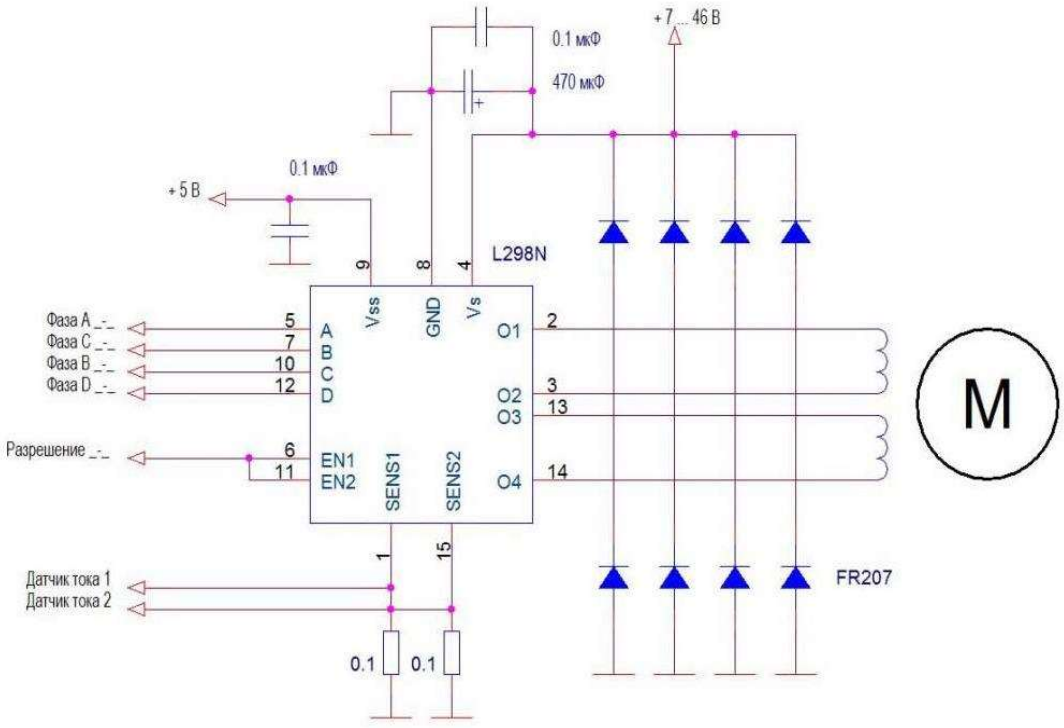


Рис. 3.4. Схема подключения шагового двигателя к микроконтроллеру с помощью драйвера L298N

Диаграмма работы этой схемы в полно шаговом режиме выглядит так (рис. 3.5).

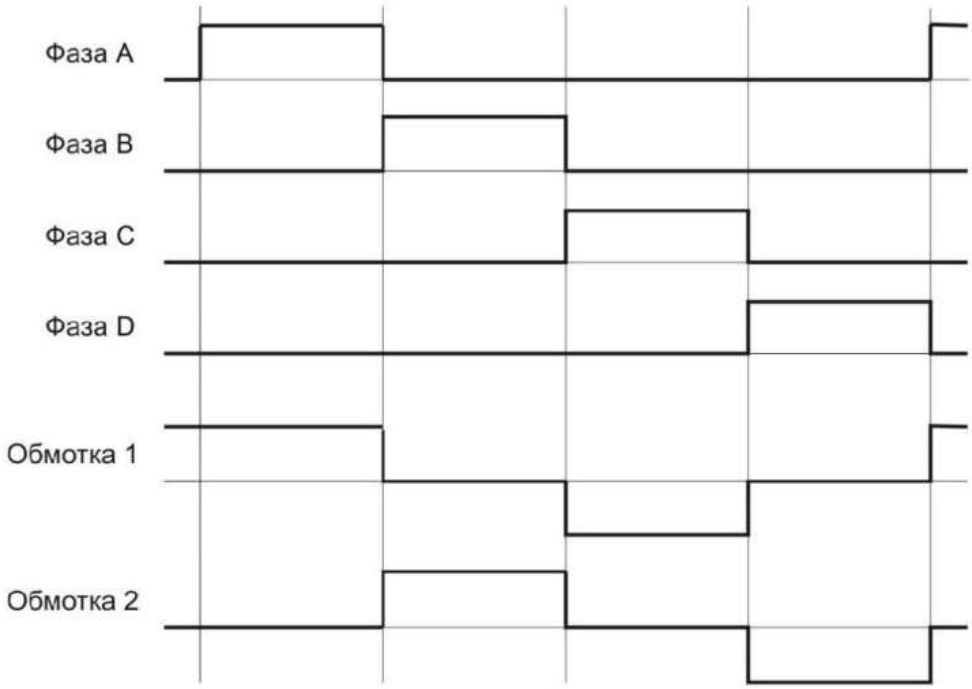


Рис. 3.5. Диаграмма работы схемы L298N в полношаговом режиме

Диоды на рис. 3.4 защищают ключи от выбросов при коммутации обмоток. Через них происходит разряд энергии, запасенной в индуктивности обмоток.

Управления полумостами происходит от входных сигналов IN1, …, IN4. Уровни сигналов 0 / 5 В. При низком уровне выход подключается к земле, при высоком – к источнику питания двигателя (+7…46 V).

Предельно-допустимый ток фазы 2 А. Защиты по току в модуле нет. Но реализация токовой защиты значительно усложняет схему, а короткое замыкание обмоток двигателя событие маловероятное. Я с таким не встречался. К тому же механическое блокирование вала шагового привода не вызывает перегрузки по току.

Подключение к контроллеру униполярных шаговых двигателей. В униполярном режиме могут работать двигатели с конфигурациями обмоток 5, 6 и 8 проводов (рис. 3.6).

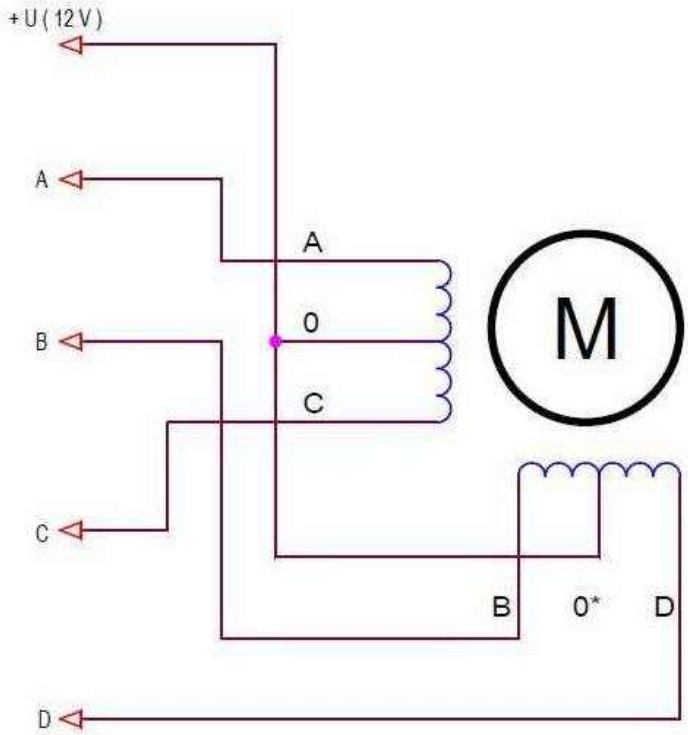


Рис. 3.6. Схема подключения униполярного шагового двигателя с 5 и 6 проводами (выводами).

При подключении униполярного шагового двигателя с 8 выводами используют то же самое подключение, как и для предыдущего варианта, только все соединения обмоток происходят вне двигателя.

Выбор напряжение для шагового двигателя.

По закону Ома через сопротивление обмотки и допустимый ток фазы.

U = I фазы * R обмотки

Сопротивление обмотки постоянному току можно измерить, а ток надо искать в справочных данных.

Следует подчеркнуть, что речь идет о простых драйверах, которые не обеспечивают сложную форму тока и напряжения. Такие режимы используются на больших скоростях вращения.

В униполярных двигателях с 5 и 6 выводами, средний вывод можно определить, измерив, сопротивление обмоток. Между фазами сопротивление будет в два раза больше, чем между средним выводом и фазой. Средние выводы подключаются к плюсу источника питания.

Дальше любой из фазных выводов можно назначить фазой А. Останется 8 вариантов коммутаций выводов. Можно их перебрать. Если учесть, что обмотка фазы В имеет другой средний провод, то вариантов становится еще меньше. Попутка обмоток фаз не ведет к выходу из строя драйвера или двигателя. Двигатель дребезжит и не крутится.

Только надо помнить, что к такому же эффекту приводит слишком высокая скорость вращения (выход из синхронизации). Т.е. надо скорость вращения установить заведомо низкую.

Пример:

```
//Объявление глобальных переменных
int Enable1Pin = 6; //выводы для подключения
int IN1Pin = 7; //микросхемы L239D и управления
int IN2Pin = 8; //обмоткой 1 ШД
int Enable2Pin = 9; //выводы для подключения
int IN3Pin = 10; //микросхемы L239D и управления
int IN4Pin = 13; //обмоткой 2 ШД
int ButtonPin = 15; //вывод для подключения кнопки запуска
boolean ButtonState; //состояние кнопки запуска
void setup () {
//Перевод выводов платы в режим работы «выход»:
pinMode (Enable1Pin, OUTPUT);
pinMode (Enable2Pin, OUTPUT);
pinMode (IN1Pin, OUTPUT);
pinMode (IN2Pin, OUTPUT);
pinMode (IN3Pin, OUTPUT);
pinMode (IN4Pin, OUTPUT);
digitalWrite (Enable1Pin, LOW); //отключаем обмотки ШД
digitalWrite (Enable2Pin, LOW);
}
void loop () {
ButtonState=digitalRead(ButtonPin);//получение состояния кнопки запуска
if (ButtonState) {//если нажата кнопка запуска
GoSteps (100, 5000, 0); //прямой ход
delay (1000); //задержка простоя
GoSteps (100, 10000, 1); //обратный ход
}
}
```

Продолжение примера

```
void GoSteps (int nSteps, int nSemiStepDelay, boolean dir) {
// Функция выполняет nSteps шагов с задержкой nSemiStepDelay между
// полушагами в направлении dir
// (dir = 0 - прямой ход, dir = 1 - обратныйход)
int i = nSteps; //счетчик сделанных шагов
digitalWrite (Enable1Pin, HIGH); //подключаем обе обмотки ШД
digitalWrite (Enable2Pin, HIGH);
while (i > 0) {//начало цикла (в теле цикла - 2 шага ШД)
if(i>0) {// Step 1/2
digitalWrite (IN1Pin, dir xor HIGH);
digitalWrite (IN2Pin, dir xor LOW);
digitalWrite (IN3Pin, HIGH);
digitalWrite (IN4Pin, LOW);
delayMicroseconds(nSemiStepDelay);// задержка в положении Step 1/2
i=i-1; //декремент счетчика шагов
}
if(i>0) {// Step 1
digitalWrite (IN1Pin, dir xor LOW);
digitalWrite (IN2Pin, dir xor HIGH);
digitalWrite (IN3Pin, HIGH);
digitalWrite (IN4Pin, LOW);
delayMicroseconds(nSemiStepDelay); // задержка в положении Step 1
i=i-1; //декремент счетчика шагов
}
if (i > 0) {// Step 3/2
digitalWrite (IN1Pin, dir xor LOW);
digitalWrite (IN2Pin, dir xor HIGH);
digitalWrite (IN3Pin, LOW);
digitalWrite (IN4Pin, HIGH);
delayMicroseconds(nSemiStepDelay);//задержка в положении Step 3/2
i=i-1; //декремент счетчика шагов
}
if (i > 0) {// Step 2
digitalWrite (IN1Pin, dir xor HIGH);
digitalWrite (IN2Pin, dir xor LOW);
digitalWrite (IN3Pin, LOW);
digitalWrite (IN4Pin, HIGH);
delayMicroseconds(nSemiStepDelay);//задержка в положении Step 2
i=i-1; //декремент счетчика шагов
}
}
digitalWrite (Enable1Pin, LOW); //отключаем обе обмотки
digitalWrite (Enable2Pin, LOW); //чтобы не шел ток и не грелась МС
}
```

Задание для самостоятельной работы

1. Запрограммируйте отладочную плату Arduino UNO вместе с платой (к лабораторной работе №3), чтобы при нажатии левой кнопки (вывод A1) шаговый двигатель вращался влево, а при нажатии правой кнопки (вывод A2) шаговый двигатель вращался вправо. При нажатии на кнопки должен загораться светодиод (вывод A0).
2. Определить максимальную скорость вращения шагового двигателя на плате (к лабораторной работе №3), максимальное количество шагов и максимальное перемещение в мм.

Контрольные вопросы

1. Опишите основные характеристики драйвера двигателей L293N.
2. Что такое биполярный шаговый двигатель?
3. Что дает использование шаговых двигателей в биполярном режиме?
4. Какое предельно допустимое значение параметр напряжения датчиков тока для L298N?
5. Как происходит подключение униполярного шагового двигателя с 8 выводами?

Лабораторная работа № 4.
Изучение принципов работы шагового двигателя и принципа программного управления для систем автоматического управления

В этой лабораторной разберем популярную, широко распространенную миросхему L6208, а также драйверы шаговых двигателей A4988 и DRV8825. Изучим принципы работы шагового двигателя и принцип программного управления драйвером L6208. Програмное управление будет осуществлятся микроконтроллером Atmega328 в среде програмирования Arduino IDE.

В итоге, мы напишем программу для микроконтроллера, которая будет управлять шаговым двигателем, сможем менять направление вращения и скорость вращения валов двигателей.

Цель работы: приобрести практические навыки по управлению шаговыми двигателями.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программы указанному в примере.
4. Выполнить задание для самостоятельной работы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.

Теоретические сведения

Микросхемы – драйвера шаговых двигателей такие, как A4988, отличаются от обычных H-мостов, или по-другому драйверов коллекторных двигателей таких как L298, возможностью

автоматической стабилизации тока, а так же автоматическим формированием управляющих сигналов на обмотки шагового двигателя для вращения или удержания ротора. Если сравнить блок-схемы микросхем этих драйверов, вам станет понятно, что они так же похожи как боевой самолет и кукурузник. Н-мост работает просто как усилитель тока и напряжения, в то время как драйвер шагового двигателя берёт на себя множество логических операций и формирует на свои внутренние Н-мосты близкие к идеалу сигналы широтно-импульсной модуляции с обратной связью по току через обмотки двигателя.

Драйвер двигателей A4988. Характеристики представлены в таблице 4.1.

Таблица 4.1. Характеристики Драйвер двигателей A4988

Характеристики:	
модель:	A4988;
напряжения питания:	от 8 до 35 В;
возможность установки шага:	от 1 до 1/16 от максимального шага;
напряжение логики:	3-5.5 В;
защита от перегрева:	да;
максимальный ток на фазу:	1 А без радиатора, 2 А с радиатором;
расстояние между рядами ножек:	12 мм;
размер платы:	20 х 15 мм;
габариты драйвера:	20 х 15 х 10 мм;
габариты радиатора:	9 х 5 х 9 мм;
вес с радиатором:	3 г;
вес без радиатора:	2 г.

Плата создана на базе микросхемы A4988 компании Allegro – драйвера биполярного шагового двигателя (рис. 4.1). Особенностями A4988 являются регулируемый ток, защита от перегрузки и перегрева, драйвер также имеет пять вариантов микрошага (вплоть до 1/16-шага). Он работает от напряжения 8 - 35 В и может обеспечить ток до 1 А на фазу без радиатора и дополнительного охлаждения (дополнительное охлаждение необходимо при подаче тока в 2 А на каждую обмотку).

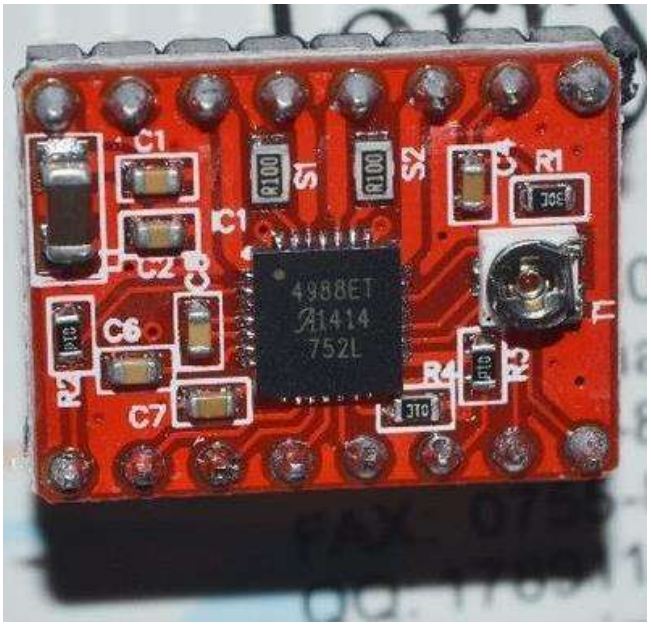


Рис. 4.1. Внешний вид платы драйвера шагового двигателя A4988

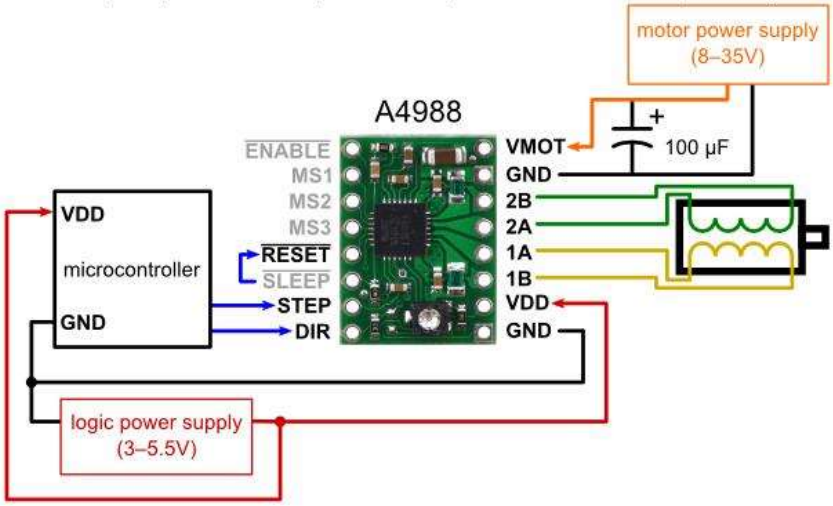


Рис. 4.2. Схема подключения драйвера шагового двигателя A4988

Описание:

Драйвер создан на базе микросхемы управления шаговым двигателем компании Allegro A4988, изготовленной по ДМОП-технологии с регулятором и защитой по току, поэтому мы настоятельно рекомендуем, перед использованием этого продукта, ознакомиться со спецификацией A4988 (рис. 4.2). Этот драйвер позволит управлять биполярным шаговым двигателем с выходным током до 2 А на обмотку (для получения дополнительной информации смотрите раздел о рассеивании мощности). Ниже приведены ключевые особенности драйвера:

– простой интерфейс управления шагом и направлением вращения электродвигателя;

– пять различных разрешений перемещения: полный шаг, 1/2-шага, 1/4-шага, 1/8-шага, 1/16-шага;

– регулируемый контроль тока с помощью потенциометра, позволит установить максимальный выходной ток. Это даст вам возможность использовать напряжение выше допустимого диапазона для достижения более высокой угловой скорости шага двигателя;

– интеллектуальное управление автоматически выбирает режим регулировки затухания тока (медленный и быстрый режимы);

– защитное отключение при перегреве и перегрузке по току, а также блокировка питания при пониженном напряжении;

– защита от короткого замыкания на землю, защита от замыкания в нагрузке.

Обратите внимание, что Pololu производит несколько драйверов шаговых двигателей, которые могут быть использованы в качестве альтернативы этого модуля. У драйвера шагового двигателя Pololu A4988 Black Edition производительность на 20% выше, и за исключением тепловых характеристик, Black Edition, и данная (зеленая) плата являются взаимозаменяемыми. Есть также большая версия драйвера Pololu на A4988, которая имеет защиту от обратной мощности на главном входе питания, а также встроенной 5 В и 3,3 В стабилизаторы напряжения, которые устраняют необходимость в покупке отдельного питания для логики и двигателей. Платы Pololu на DRV8825 предлагают на около 50% более высокую производительность в более широком диапазоне напряжений и с несколькими дополнительными функциями, в то время как платы на DRV8834 работают с двигателями с напряжением питания от 2,5 В; любую из этих плат можно использовать в качестве альтернативы этого драйвера во многих приложениях.

Использование:

Соединение с источником питания:

Для работы с драйвером необходимо питание логического уровня (3 - 5,5 В), подаваемое на выводы VDD и GND, а также питание двигателя (8 - 35 В) на выводы VMOT и GND. Чтобы

обеспечить необходимый потребляемый ток (при пиковых до 4 А), необходимо поставить конденсаторы для гальванической развязки как можно ближе к плате.

Внимание: В плате используются керамические конденсаторы с низким эквивалентным последовательным сопротивлением, что делает её уязвимой для индуктивно-ёмкостных скачков напряжения, особенно если питающие провода длиннее нескольких сантиметров. В некоторых случаях, эти скачки могут превысить максимально допустимое значение (35 В для A4988) и повредить плату. Одним из способов защиты платы от подобных скачков является установка большого (не меньше 47 мкФ) электролитического конденсатора между выводом питания (VMOT) и землёй близко к плате.

Соединение двигателя:

При правильном подключении, через Pololu A4988 можно управлять четырёх-, шести- и восьми- проводными шаговыми двигателями.

Внимание: Соединение или разъединение шагового двигателя при включённом драйвере может привести к поломке двигателя.

Размер шага (и микрошага):

У шаговых двигателей обычно установлена конкретная величина (например, 1,8° или 200 шагов на оборот), при которой достигается полный оборот в 360°. Микрошаговый драйвер, такой как A4988 позволяет увеличить разрешение за счёт возможности управления промежуточными шагами. Это достигается путём возбуждения обмоток средней величины тока. Например, управление мотором в режиме четверти шага даст двигателю с величиной 200-шагов-за-оборот уже 800 микрошагов при использовании разных уровней тока.

Разрешение (размер шага) задаётся комбинациями переключателей на входах (MS1, MS2, и MS3). С их помощью можно выбрать пять различных шагов, в соответствии с таблице 4.2. На входы MS1 и MS3 переключателя установлены 100 кОм подтягивающие на землю резисторы, а на MS2 - 50 кОм, и, если оставить их не подключёнными, двигатель будет работать в полношаговом режиме. Для правильной работы в режиме микрошага необходим слабый ток (см. ниже), который обеспечивается ограничителями по току. В противном случае, промежуточные уровни будут некорректно восприниматься, и двигатель будет пропускать микрошаги.

Таблица 4.2. Установка разрешения микрошага для драйвера A4988

MS1	MS2	MS3	Разрешение микрошага
Низкий	Низкий	Низкий	Полный шаг
Высокий	Низкий	Низкий	1/2 шага
Низкий	Высокий	Низкий	1/4 шага
Высокий	Высокий	Низкий	1/8 шага
Высокий	Высокий	Высокий	1/16 шага

Входы управления:

Каждый импульс на входе STEP соответствует одному микрошагу двигателя, направление вращения которого зависит от сигнала на выводе DIR. Обратите внимание, что выводы STEP и DIR не подтянуты к какому-либо конкретному внутреннему напряжению, поэтому вы не должны оставлять эти выводы плавающими при создании приложений. Если вы просто хотите вращать двигатель в одном направлении, вы можете соединить DIR непосредственно с VCC или GND. Чип имеет три различных входа для управления состоянием питания: RST, SLP и EN. Дополнительные сведения об этих состояниях см. в техническом описании. Обратите внимание, что вывод RST плавает; если вы его не используете, вы можете подключить его к соседнему контакту SLP на печатной плате, чтобы подать на него высокий уровень и включить плату.

Для достижения высокой скорости шага, питания двигателя, как правило, гораздо выше, чем это было бы допустимо без активного ограничения тока. Например, типичный шаговый двигатель может иметь максимальный ток 1 А с 5 Ом; сопротивлением обмотки, отсюда максимально допустимое питание двигателя равно 5 В (U=I*R). Использование же такого двигателя с питанием 12 В позволит повысить скорость шага. Однако, чтобы предотвратить повреждение двигателя, необходимо ограничить ток до уровня ниже 1 А.

Pololu A4988 поддерживает активное ограничение тока, которое можно установить подстроечным потенциометром на плате. Один из способов установить предельный ток - подключить драйвер в полношаговый режим и измерять ток, протекающий через одну обмотку двигателя без синхронизации по входу STEP. Измеренный ток будет равен 0,7 части предельного тока (так как обе обмотки всегда ограничиваются примерно на 70% от текущей настройки

предельного тока в полношаговом режиме). Учтите, что при изменении логического напряжения Vdd, на другое значение, изменит предельный ток, поскольку напряжение на выводе «ref» является функцией Vdd.

Еще один способ установить предельный ток – измерить напряжение на выводе «ref» и вычислить полученное ограничение тока (резисторы SENSE равны 0,05 Ом). Напряжение вывода доступно через металлизированное сквозное отверстие (в кружке на шёлкографии печатной платы). Ограничение тока относится к опорному напряжению следующим образом:

Current Limit = VREF × 2,5.

Например: опорное напряжение равно 0,3 В, предельный ток 0,75 А. Как упоминалось выше, в режиме полного шага, ток через катушки ограничен 70% от текущего предела, поэтому, чтобы получить полный шаг тока катушки в 1 А, текущий предел должен быть 1 А / 0,7 = 1,4 А, что соответствует VREF 1,4 А / 2,5 = 0,56 В. Смотрите спецификацию A4988 для получения дополнительных сведений.

Примечание: Ток обмотки может сильно отличаться от тока источника питания, поэтому не следует измерять ток на источнике питания, чтобы установить ограничение тока. Подходящим местом для измерения тока является одна из обмоток вашего шагового двигателя.

Рекомендации по рассеиванию мощности:

Максимально допустимый ток, подаваемый на обмотку, у микросхемы A4988 равен 2 А. Фактический ток, который можно подать на плату, зависит от качества охлаждения микросхемы. Плата разработана с учётом отвода тепла от микросхемы, но при токе выше 1 А на обмотку необходим теплоотвод или другое дополнительное охлаждение.

Эта плата может нагреться так, что можно получить ожог, задолго до того, как перегреется сама микросхема. Будьте осторожны при обращении с платой и со всеми подключёнными к ней устройствами.

Обратите внимание, что ток, измеренный на источнике питания, как правило, не соответствует величине тока на обмотке. Так как напряжение, подаваемое на драйвер, может быть значительно выше напряжения на обмотке, то, соответственно, измеряемый ток

на источнике питания может быть немного ниже, чем ток на обмотке (драйвер и обмотка в основном работают в качестве переключаемого источника с пошаговым понижением питания). Кроме того, если напряжение питания намного выше необходимого двигателю уровня для достижения требуемого тока, то скважность будет очень низкой, что также приводит к существенным различиям между средним и RMS током (среднеквадратичное значение переменного тока).

Драйвер шагового двигателя DRV8825. Плата создана на базе микросхемы компании TI (Texas Instruments Inc.) DRV8825 – биполярном шаговом драйвере двигателя. Расположение выводов и интерфейс модуля почти совпадает с драйвером шагового двигателя Pololu на микросхеме A4988, поэтому DRV8825 может стать высокопроизводительной заменой этой платы во многих приложениях (рис. 4.3). Особенностями DRV8825 являются регулируемый ток, защита от перегрузки и перегрева, драйвер также имеет шесть вариантов микрошага (вплоть до 1/32-шага). Он работает от напряжения 8,2 - 45 В и может обеспечить ток до 1,5 А на фазу без радиатора и дополнительного охлаждения (дополнительное охлаждение необходимо при подаче тока в 2,2 А на каждую обмотку).

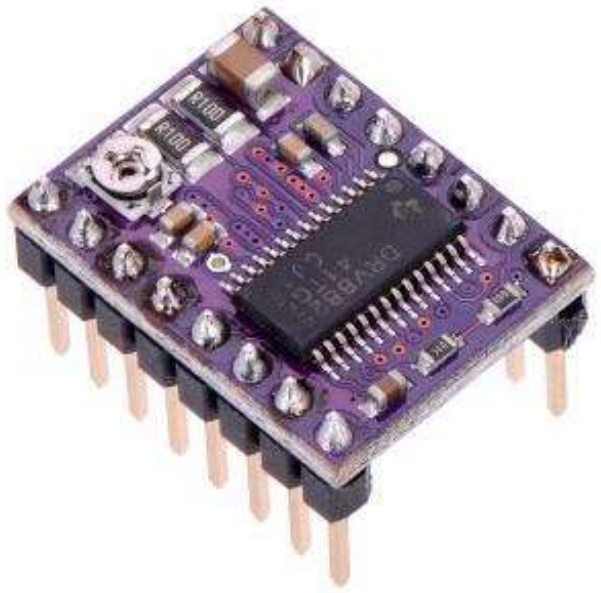


Рис. 4.3. Внешний вид платы драйвера шагового двигателя DRV8825

Описание:

Драйвер создан на базе микросхемы управления шаговым

двигателем компании TI DRV8825; поэтому настоятельно рекомендуем, перед использованием этого продукта, ознакомиться со спецификацией DRV8825. Этот драйвер позволит управлять биполярным шаговым двигателем с выходным током в 1,2 А на обмотку (для получения дополнительной информации смотрите раздел о рассеивании мощности). Ниже приведены ключевые особенности драйвера:

– простой интерфейс управления шагом и направлением вращения электродвигателя;

– шесть различных дискретных перемещений: полный шаг, пол шага, 1/4-шага, 1/8-шага, 1/16-шага и 1/32-шага;

– регулируемый контроль тока с помощью потенциометра, позволит установить максимальный выходной ток. Это даст вам возможность использовать напряжение выше допустимого диапазона для достижения более высокой угловой скорости шага двигателя;

– интеллектуальное управление автоматически выбирает режим регулировки затухания тока (медленный и смешанный режимы);

– максимальное напряжение питания 45 В;

– встроенный стабилизатор напряжения (не требуется дополнительного питания для внешних логических схем);

– может напрямую взаимодействовать с 3,3 В и 5 В системами;

– защитное отключение при перегреве и перегрузке по току, а также блокировка питания при пониженном напряжении;

– защита от короткого замыкания на землю, защита от замыкания в нагрузке;

– четырехслойная печатная плата с медным слоем толщиной 0,07 мм для лучшего рассеивания тепла;

– контакт заземления доступен для пайки в нижней части печатной платы, внизу микросхемы;

– размер модуля, расположение выводов и интерфейс во многом соответствуют драйверу шагового двигателя Pololu на A4988 (смотрите внизу страницы для получения дополнительной информации).

Pololu также выпускает драйвер шагового двигателя на DRV8824, который можно использовать в качестве прямого аналога платы на DRV8825 при работе со слаботочными шаговыми

двигателями. При работе с драйвером на DRV8824 без дополнительного радиатора, можно подавать ток до 0,75 А на обмотку и до 1,2 А при правильном охлаждении, но у DRV8824 более мощные токочувствительные резисторы, благодаря которым, выполнение микрошага лучше, чем у драйвера на микросхеме DRV8825 при низком токе.

Отличить драйверы можно по маркировкам на микросхемах DRV8824 и DRV8825; если у вас есть несколько разных драйверов, то вы можете их пометить (для этого предусмотрен пустой квадрат на шёлкографии).

Данное изделие поставляется со всеми компонентами поверхностного монтажа, включая интегральную схему драйвера DRV8825 – размещение компонентов показано на фотографии продукта.

Некоторыми однополярными шаговыми двигателями (например, с шестью или восемью выводами) можно управлять с помощью этого драйвера как биполярными. Драйвер нельзя использовать для управления униполярными двигателями с пятью выводами (рис. 4.4).

Использование:

Соединение с источником питания:

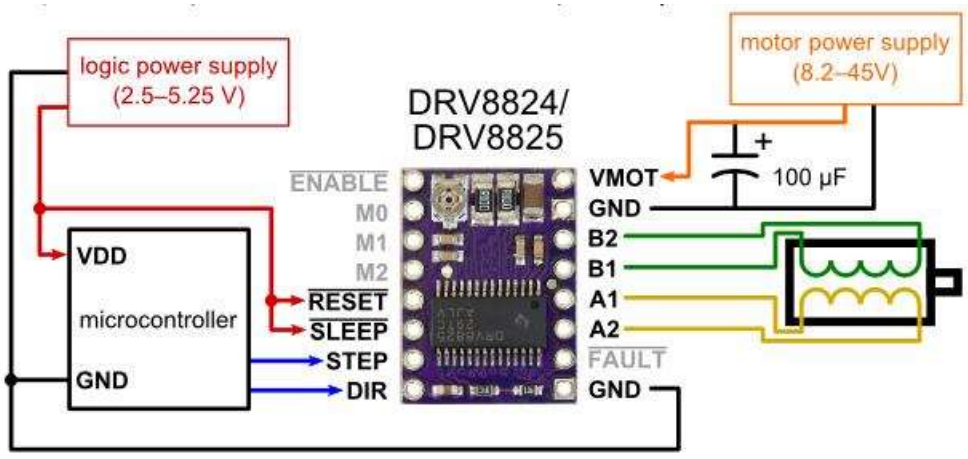


Рис. 4.4. Схема подключения драйвера шагового двигателя DRV8825

Драйверу необходим источник питания двигателя в 8,2 - 45 В, соединённый через выводы VMOT (питание двигателя) и GND (заземление). Питанию необходимы развязывающие конденсаторы, их следует разместить поближе к плате, а также удостовериться, что

они способны обеспечивать необходимый для двигателей ток.

Внимание: В плате используются керамические конденсаторы с низким эквивалентным последовательным сопротивлением, что делает её уязвимой для индуктивно-ёмкостных скачков напряжения, особенно если питающие провода длиннее нескольких сантиметров. В некоторых случаях, даже при напряжении питания двигателя всего в 12 В, эти скачки могут превысить максимально допустимое значение (45 В для DRV8825) и повредить плату. Одним из способов защиты платы от подобных скачков является установка большого (не меньше 47 мкФ) электролитического конденсатора между выводом питания (VMOT) и землёй близко к плате.

Соединение двигателя:

При правильном подключении, через Pololu DRV8825 можно управлять четырёх-, шести- и восьми- проводными шаговыми двигателями.

Внимание: Соединение или разъединение шагового двигателя при включённом драйвере может привести к поломке двигателя.

Размер шага (и микрошага):

У шаговых двигателей обычно установлена конкретная величина (например, 1,8° или 200 шагов на оборот), при которой достигается полный оборот в 360°. Микрошаговый драйвер, такой как DRV8825 позволяет увеличить разрешение за счёт возможности управления промежуточными шагами. Это достигается путём возбуждения обмоток средней величины тока. Например, управление мотором в режиме четверти шага даст двигателю с величиной 200-шагов-за-оборот уже 800 микрошагов при использовании разных уровней тока.

Разрешение (размер шага) задаётся режимами входов переключателей (MODE0, MODE1, и MODE2). С их помощью можно выбрать шесть различных шагов, в соответствии с таблицей 4.3 ниже. Все три входа переключателя имеют 100 кОм подтягивающие на землю резисторы, поэтому если оставить их не подключёнными, двигатель будет работать в полношаговом режиме. Для правильной работы в режиме микрошага необходим слабый ток (см. ниже), который обеспечивается ограничителями по току. В противном случае, промежуточные уровни будут некорректно восприниматься, и двигатель будет пропускать микрошаги.

Таблица 4.3. Установка разрешения микрошага для драйвера DRV8825

MS1	MS2	MS3	Разрешение микрошага
Низкий	Низкий	Низкий	Полный шаг
Высокий	Низкий	Низкий	1/2 шага
Низкий	Высокий	Низкий	1/4 шага
Высокий	Высокий	Низкий	1/8 шага
Низкий	Низкий	Высокий	1/16 шага
Высокий	Низкий	Высокий	1/32 шага
Низкий	Высокий	Высокий	1/32 шага
Высокий	Высокий	Высокий	1/32 шага

Входы управления:

Каждый импульс на входе STEP соответствует одному микрошагу двигателя, направление вращения которого зависит от сигнала на выводе DIR. По умолчанию, оба этих импульса подтянуты к низкому логическому уровню 100 кОм подтягивающими на землю резисторами. Если вам необходимо вращение только в одном направлении, вывод DIR можно оставить отключённым.

У микросхемы есть три разных входа для контроля состояния питания: RESET, SLEEP, и ENBL. Для получения более подробной информации смотрите документацию микросхемы. Обратите внимание, что вход SLEEP подтянут через 1 мОм резистор на землю, так же, как и разъёмы RESET и ENBL через 100 кОм. Эти настройки предотвратят срабатывание драйвера; для включения драйвера оба этих входа должны иметь высокий логический уровень (можно подать «высокий» уровень напряжения – между 2,2 и 5,25 В – или динамически управлять через цифровые выводы микроконтроллера). По умолчанию, вывод ENBL запускает драйвер, поэтому его можно оставить не подключённым.

Особенностью DRV8825 является низкий уровень выхода FAULT, при котором полевые транзисторы H-моста отключаются в результате перегрузок по току или перегрева. Выводы SLEEP и

FAULT соединены на плате через 10 кОм резистор, работающий как подтягивающий к питанию вывод FAULT, при внешнем высоком уровне вывода SLEEP. Поэтому для вывода FAULT не требуется внешнего подтягивания к питанию. Отметим, что на плате имеется 1,5 кОм защитный резистор, соединённый последовательно с выводом FAULT. Это позволяет безопасно соединять плату непосредственно с источником питания логики. Это может пригодиться при использовании платы с устройствами, разработанными для работы с совместимым по выводам драйвером на A4988. В подобных устройствах 10 кОм резистор между выводами SLEEP и FAULT работает в качестве подтягивающего к питанию для SLEEP, что делает плату на DRV8825 непосредственной заменой для платы на A4988 (На плате с A4988 есть внешний подтягивающий к питанию резистор на выводе SLEEP). Чтобы не было проблем от подтягивания вывода SLEEP, добавленный вами внешний резистор не должен превышать номинала в 4,7 кОм.

Ограничение тока:

Для достижения высокой скорости шага, питания двигателя, как правило, гораздо выше, чем это было бы допустимо без активного ограничения тока. Например, типичный шаговый двигатель может иметь максимальный ток 1 А с 5 Ом; сопротивлением обмотки, отсюда максимально допустимое питание двигателя равно 5 В. Использование же такого двигателя с питанием 12 В позволит повысить скорость шага. Однако, чтобы предотвратить повреждение двигателя, необходимо ограничить ток до уровня ниже 1 А.

DRV8825 поддерживает активное ограничение тока, которое можно установить подстроечным потенциометром на плате. Пользователи, как правило, предпочитают устанавливать предельный ток на уровне или ниже параметров вашего шагового двигателя. Один из способов установить предельный ток – подключить драйвер в полношаговый режим и измерять ток, протекающий через одну обмотку двигателя без синхронизации по входу STEP. Измеренный ток будет равен 0,7 части предельного тока (так как обе обмотки всегда ограничиваются примерно на 70% от текущей настройки предельного тока в полношаговом режиме).

Еще один способ установить предельный ток – измерить напряжение на выводе «ref» и вычислить полученное ограничение

тока (резисторы SENSE равны 0,100 Ом). Напряжение вывода доступно через металлизированное сквозное отверстие (в кружке на шёлкографии печатной платы). Ограничение тока относится к опорному напряжению следующим образом:

Current Limit = VREF × 2

Например: если у вас шаговый двигатель рассчитан на 1 А, то вы можете получить такое значение тока (1А), установив опорное напряжение в 0,5 В.

Примечание: Ток обмотки может сильно отличаться от тока источника питания, поэтому не следует измерять ток на источнике питания, чтобы установить ограничение тока. Подходящим местом для измерения тока является одна из обмоток вашего шагового двигателя.

Рекомендации по рассеиванию мощности:

Максимально допустимый ток, подаваемый на обмотку, у микросхемы DRV8825 равен 2,5 А, но токочувствительные резисторы ограничивают его на уровне 2,2 А. Фактический ток, который можно подать на плату, зависит от качества охлаждения микросхемы. Плата разработана с учётом отвода тепла от микросхемы, но при токе выше 1,5 А на обмотку необходим теплоотвод или другое дополнительное охлаждение.

Эта плата может нагреться так, что можно получить ожог, задолго до того, как перегреется сама микросхема. Будьте осторожны при обращении с платой и со всеми подключёнными к ней устройствами.

Обратите внимание, что ток, измеренный на источнике питания, как правило, не соответствует величине тока на обмотке. Так как напряжение, подаваемое на драйвер, может быть значительно выше напряжения на обмотке, то, соответственно, измеряемый ток на источнике питания может быть немного ниже, чем ток на обмотке (драйвер и обмотка в основном работают в качестве переключаемого источника с пошаговым понижением питания). Кроме того, если напряжение питания намного выше необходимого двигателю уровня для достижения требуемого тока, то скважность будет очень низкой, что также приводит к существенным различиям между средним и RMS током (среднеквадратичное значение переменного тока). Кроме того, обратите внимание, что ток обмотки является функцией ограничения тока, но она не обязательно равна текущему порогу. Действующее значение тока обмотки меняется с каждым

микрошагом. Для получения дополнительной информации смотрите описание DRV8825.

Использование:

Номинал резисторов R2 и R3 – 0,100 Ом на плате с DRV8825.

Основные отличия между микросхемами DRV8825 и A4988:

Драйвер на микросхеме DRV8825 создавался в качестве прямого аналога драйвера на A4988. Он может использоваться для замены A4988 во множестве приложений. Это доступно благодаря одинаковому размеру плат, размещению выводов и общему интерфейсу управления. Тем не менее, есть несколько отличий, которые необходимо отметить:

– вывод для подачи логического напряжения на A4988 используется как выход FAULT в DRV8825, поскольку DRV8825 не требуется логическое питание (у A4988 нет выхода FAULT). Обратите внимание, что прямое соединение выхода FAULT с логическим питанием является безопасным (между микросхемой и выводом установлен 1,5 кОм резистор), поэтому модуль DRV8825 может использоваться в устройствах, предназначенных для A4988, в котором напряжение логики подаётся через этот вывод;

– вывод SLEEP на DRV8825 не подтянут к питанию по умолчанию как на A4988, но он соединён на плате с выводом FAULT через 10k резистор. Таким образом, у устройства, предназначенного для работы на A4988, в котором напряжение логики подаётся выводом FAULT, вывод SLEEP будет надёжно подтянут через 10k резистор к питанию;

– различное расположение потенциометра, регулирующего предельный ток;

– разное отношение предельного тока к опорному напряжению;

– драйвер на DRV8825 поддерживает 1/32-шаг в микрошаговом режиме; у A4988 минимум 1/16-шага;

– вход выбора 1/16-шагового режима на A4988 соответствует 1/32-шаговому на DRV8825. В остальном таблица переключения режимов у драйверов совпадает;

– у драйверов различные требования к минимальному размеру синхронизирующих импульсов вывода STEP. У DRV8825 шаг импульсов высокого и низкого уровня должен быть не меньше 1,9 мкс; при использовании A4988 шаг можно сократить до 1 мкс;

– DRV8825 поддерживает более высокое напряжение питания

A4988 (45 В против 35 В). Это означает, что драйвер DRV8825 безопаснее при работе с высоким напряжением и менее восприимчив к индуктивно-ёмкостным скачкам напряжения;

– без дополнительного охлаждения, драйвер DRV8825 выдерживает ток в 1,5 А на обмотку против 1 А у A4988, при тестировании в полношаговом режиме (1,2 А на обмотку для A4988 Black Edition);

– у DRV8825 используется различное наименование выводов, но функционально они соответствуют выводам A4988, поэтому одинаковое соединение драйверов приведёт к одинаковому движению моторов. На обеих платах первая часть маркировки обозначает номер обмотки (на DRV8825 обмотки обозначены как «A» и «B», а у A4988 – «1» и «2»);

– обратите внимание, что плата DRV8825 фиолетового цвета – это может быть важно для светочувствительных устройств.

Таким образом, драйвер на микросхеме DRV8825 схож с A4988, схема подключения к микроконтроллеру с минимальным количеством проводников является общей для обеих плат.

Драйвер двигателей L6208N. Принцип управления драйвером двигателя L6208N аналогичен управлению драйверами DRV8825 и A4988 (табл. 4.4). На отладочной плате к лабораторной работе №4 собрана схема, представленная на рис. 4.5 - 4.6.

Таблица 4.4. Общие характеристики драйвера двигателя L6208N

Назначение	Контроллеры DC двигателей
Напряжение питания	8…52 В
Максимальный выходной ток	2,8 А
Рабочая температура	-40°…150° C

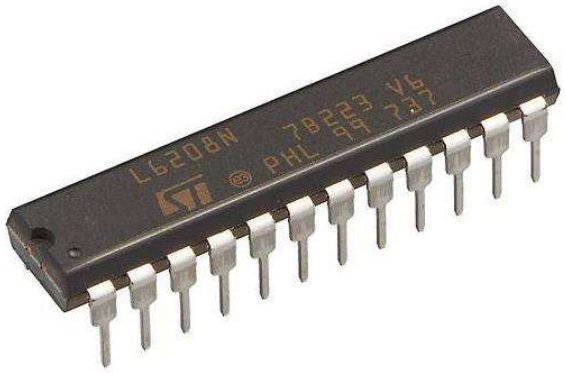


Рис. 4.5. Внешний вид платы драйвера шагового двигателя L6208N

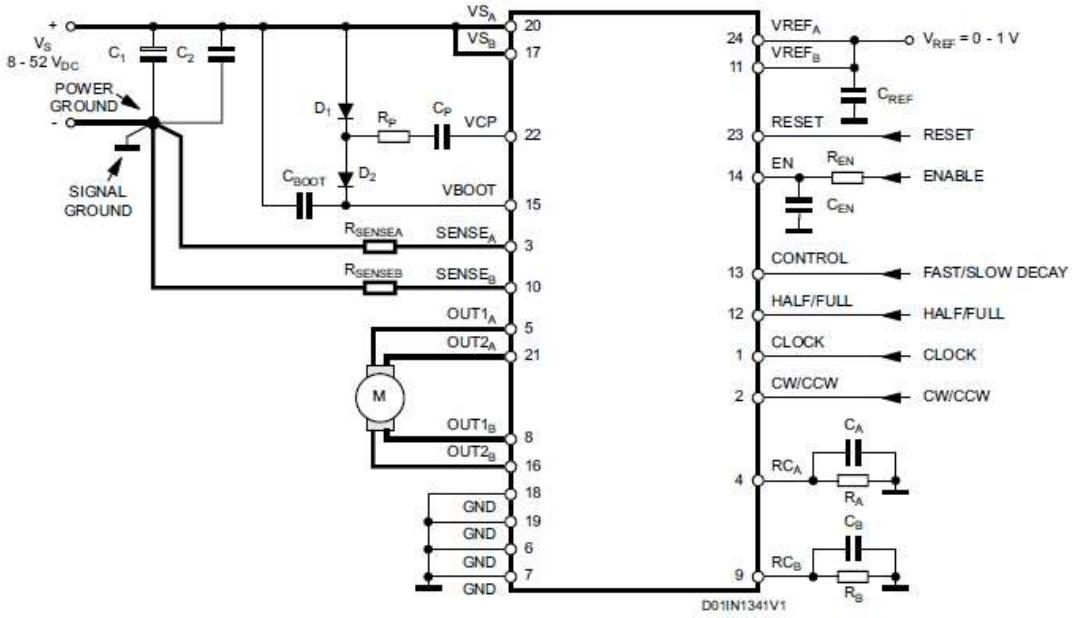


Рис. 4.6. Схема подключения драйвера шагового двигателя DRV8825

Таблица 4.5. Параметры элементов электрической схемы рис. 6

Component	Value	Component	Value
C_1	100 µF	D_1	1N4148
C_2	100 nF	D_2	1N4148
C_A	1 nF	R_A	39 KΩ
C_B	1nF	R_B	39 KΩ
C_{BOOT}	220 nF	R_{EN}	100 KΩ
C_P	10 nF	R_P	100 Ω
C_{EN}	5.6 nF	R_{SENSEA}	0.3 Ω
C_{REF}	68 nF	R_{SENSEB}	0.3 Ω

Ниже приведены три примера которые подходят для всех рассмотренных типов драйверов шаговых двигателей.

Пример 1:

```
#define pinEnable 4 // Вывод активации драйвера
#define pinStep   2 // Вывод для указания количетсва шагов
#define pinDir 3 // Вывод для указания направления вращения
void setup () {
Serial.begin(9600); // Активация порта UART
Serial.println("Test DRV8825 ro DRV4988");
pinMode (pinEnable, OUTPUT); // Вывод настроен на передачу сигналов
pinMode (pinDir, OUTPUT); // Вывод настроен на передачу сигналов
pinMode (pinStep, OUTPUT); // Вывод настроен на передачу сигналов
```

```
}
void loop () {
int i = 0;
digitalWrite (pinDir, HIGH); // Вращение по часовой стрелке
digitalWrite (pinStep, LOW); // шаги отсутствуют
digitalWrite (pinEnable, HIGH); // активация драйвера
// вращение шагового двигателя на 200 шагов
for (i = 0; i <200; i++) {
Serial.println(i);
digitalWrite (pinStep, HIGH);
delay (10); // Задержка определяет скорость вращения
digitalWrite (pinStep, LOW);
delay (10);
}
// Изменим направления вращения
digitalWrite (pinDir, LOW); // Вращение против часовой стрелки
// Вращение шагового двигателя на 200 шагов с большей скоростью
for (i = 0; i < 200; i++) {
```

Продолжение примера 1

```
Serial.println(i);
digitalWrite (pinStep, HIGH);
delay (1);
digitalWrite (pinStep, LOW);
delay (1);
}
// Вывод в монитор порта сообщения о блокировке шагового
// двигателя на 5 секунд
Serial.println("Axe bloque + attendre 5 sec");
Delay (5000);
// Вывод в монитор порта сообщения о разблокировании шагового
// двигателя на 5 секунд
Serial.println("Deblocage axe");
digitalWrite (pinEnable, LOW); // Деактивация двигателя
delay (5000);
}
```

Пример 2:

```
#define STEPPER_MICROSTEPS 1 // Количество микрошагов
// Программа для шагового двигателя с мах током 1А и 7.5 град/шаг:
#define STEPPER_PAUSE 200 // Пауза между шагами 200 микросекунд
#define STEPPER_DEGREES_PER_STEP 1.8 // Шаг 1.8 градуса
#define EN_PIN 4 // Вывод активации драйвера
#define STEP_PIN 2// Вывод для указания количества шагов
```

```
#define DIR_PIN 3 // Вывод для указания направления вращения
void setup () {
pinMode (DIR_PIN, OUTPUT); // Вывод настроен на передачу сигналов
pinMode (STEP_PIN, OUTPUT); // Вывод настроен на передачу сигналов
pinMode (EN_PIN, OUTPUT); // Вывод настроен на передачу сигналов
Serial.begin(9600); // Активация порта UART
}
void loop () {
digitalWrite (EN_PIN, HIGH);
// Поворот шагового двигателя на 3600 градусов с определённой скоростью
rotateDeg (3600, 0.1);
Serial.println(millis ());
Delay (2000); // Пауза 2 секунды
rotateDeg (-3600, 0.1); // Поворот шагового двигателя на 3600
// градусов с определённой скоростью в другую сторону
Serial.println(millis ());
```

Продолжение примера 2

```
Delay (2000); // Пауза 2 секунды
// Поворот шагового двигателя на 1600 шагов с определённой скоростью
// для поворота двигателя на один оборот необходимо 200 шагов
Serial.println(millis ());
rotateDeg (1600, 0.2);
Serial.println(millis ());
delay (2000);
// Поворот шагового двигателя на 1600 градусов с определённой скоростью
// в другую сторону
rotate (-1600, 0.2);
delay (2000);
}
void rotate (int steps, float speed) {
// Скорость находится в пределах от 0.01 до 1
// 1 - максимальная скорость, но нее для всех шаговых двигателей
int dir = (steps > 0)? HIGH: LOW;
steps = abs(steps);
digitalWrite (DIR_PIN, dir);
float usDelay = (1/speed) * STEPPER_PAUSE;
for (int i = 0; i < steps; i++) {
digitalWrite (STEP_PIN, HIGH);
delayMicroseconds(usDelay);
```

```
digitalWrite (STEP_PIN, LOW);
delayMicroseconds(usDelay);
}
}
void rotateDeg (float deg, float speed) {
// Скорость находится в пределах от 0.01 до 1
// 1 - максимальная скорость, но нее для всех шаговых двигателей
int dir = (deg > 0)? HIGH: LOW;
digitalWrite (DIR_PIN, dir);
int steps = abs(deg)/ (STEPPER_DEGREES_PER_STEP / STEPPER_MI-
CROSTEPS);
float usDelay = (1/speed) * STEPPER_PAUSE;
for (int i = 0; i < steps; i++) {
digitalWrite (STEP_PIN, HIGH);
delayMicroseconds(usDelay);
digitalWrite (STEP_PIN, LOW);
delayMicroseconds(usDelay);
}
}
```

Пример 3:

```
#include <Stepper.h> // Подключение библиотеки для шаговых двигате-
лей
#define pinEnable 4 // Вывод активации драйвера
const int stepsPerRevolution = 800;// Количество шагов двигателя, умно-
женное на 4
Stepper myStepper (stepsPerRevolution, 2, 3); //
void setup () {
myStepper.setSpeed(60); // Установка скорости вращения шагового дви-
гателя
Serial.begin(9600); // Активация порта UART
pinMode (pinEnable, OUTPUT); // Вывод настроен на передачу сигналов
digitalWrite (pinEnable, HIGH); // Активация драйвера
}
void loop (){
Serial.println("clockwise"); // Вывод сообщения о вращении вправо
myStepper.step(stepsPerRevolution); // Поворот на 200 шагов вправо
delay (500); //Пауза 0,5 секунды
Serial.println("counterclockwise"); // Вывод сообщения о вращении влево
myStepper.step(-stepsPerRevolution); // Поворот на 200 шагов влево
delay (500); //Пауза 0,5 секунды
}
```

Задание для самостоятельной работы

1. Запрограммируйте отладочную плату Arduino UNO вместе с платой (к лабораторной работе №4), чтобы шаговый двигатель

вращался влево (10 оборотов +№варианта), а после 5 секундной паузы вращался вправо (10 оборотов +№варианта) с большей в два раза скоростью.

2. Определить максимальную скорость вращения шагового двигателя на плате (к лабораторной работе №3), максимальное количество шагов и максимальное перемещение в мм.

Контрольные вопросы

1. Опишите базовые характеристики драйвера двигателей A4988.
2. Какие можно выделить особенности A4988?
3. Как происходит соединение с источником питания?
4. С помощью чего можно добиться увеличение размера шага двигателя?
5. Назовите главную особенность DRV8825, при которой полевые транзисторы H-моста отключаются.

Лабораторная работа № 5.
Отображение текста на графическом LCD дисплее WG12864A на базе контроллера KS0107

В этой лабораторной будет проведено подключение графического LCD дисплея к микроконтроллеру. Для среды Arduino IDE уже существует библиотека с примером по подключению графического LCD-дисплея WG12864A.

В итоге, мы напишем программу для микроконтроллера, которая будет управлять графическим LCD дисплеем и отображать текстовую информацию, а также научимся создавать новые шрифты для дисплея.

Цель работы: приобрести практические навыки по управлению графическим LCD дисплеем и использовать его для вывода текстовых сообщений.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программы указанному в примере.
4. Выполнить задание для самостоятельной работы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.

Теоретические сведения

Обычно для вывода информации сигнального дисплея на HD44780 более чем достаточно. Но иногда нужно нарисовать картинку, график или хочется сделать красиво, с модными

менюшками. Тут на помощь приходят графические дисплеи. Одним из самых простых и доступных является дисплей на контроллере KS0107 или аналоге. Например, WG12864A от Winstar. Сам дисплей вполне свободно достается, имеет довольно большой размер (диагональ около 80мм) и разрешение 128x64 пикселя. Монохромный (рис. 5.1).

Рис. 5.1. ЖКИ WG12864A на базе контроллера KS0107

Подключение

Управление дисплеем параллельное. Есть шина данных и линии задания направления и вида данных. Это, кстати, один из минусов – требует очень много проводов управления. Занимает почти 16 линий. Но зато и очень прост в работе.

Итак, если взять тот, что у меня WG12864A-TGH-VNW то у него следующая распиновка (рис. 5.2).

Назначение выводов контроллера:

– Vdd и Vss это питание, оно у него пятивольтовое.

– Vee – источник отрицательного напряжения. Там примерно минус 5 вольт. Есть не на всех моделях этих дисплеев, у Winstar о наличии такой фенечки говорит буква V в маркировке. WG12864A-TGH-VNW.

– Vo – напряжение регулировки контраста. Туда подается либо 0…5 вольт, либо от -5 до 5, в зависимости от модели и температурного диапазона. Обычно более морозостойкие дисплеи требуют отрицательное напряжение.

Схема включения простая:

– D/I – Данные/команда. Логический уровень на этом выводе определяет предназначение кода на шине данных. 1 – данные, 0 – команда.

– R/W – Чтение/Запись. Уровень на этой ноге задает тип действия. 1 чтение, 0 запись.

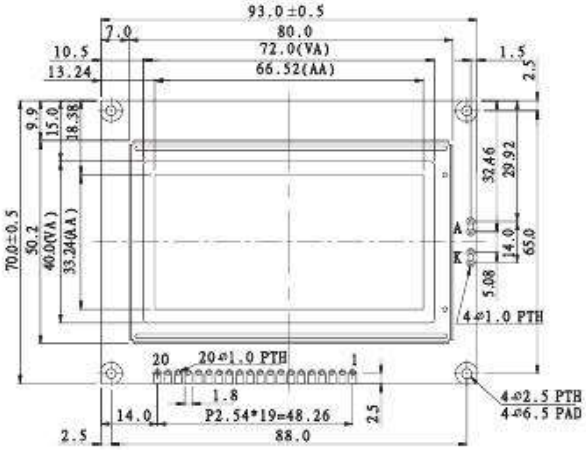
– E – Строб, синхронизирующий импульс. Дрыг этой вожжи вверх-вниз проворачивает шестеренки в интерфейсе контроллера.

– DB0..7 – Шина данных, на которую мы выдаем нужный нам код.

– CS1 и CS2 – выбор контроллера.

– RST – сигнал сброса. Ноль на этой линии сбрасывает контроллеры в ноль. Но не сбрасывает видеопамять, только текущую адресацию.

– A и K – питание светодиодной подсветки. Там, как правило, обычные светодиоды, так что напрямую туда 5 вольт подавать нельзя. Только через ограничительный резистор. Ток потребления подсветки весьма велик, около 200мА, падение напряжения в районе 4 вольт. На пяти вольтовом питании ограничетельный резистор должен быть порядка 5-10 Ом.



Pin NO.	Symbol	Function
1	Vss	GND
2	Vdd	Power supply (+5V)
3	Vo	Contrast Adjustment
4	D/I	Data/instruction
5	R/$\overline{W}$	Data read/write
6	E	H→L Enable signal
7	DB0	Data bus line
8	DB1	Data bus line
9	DB2	Data bus line
10	DB3	Data bus line
11	DB4	Data bus line
12	DB5	Data bus line
13	DB6	Data bus line
14	DB7	Data bus line
15	CS1	Chip select for IC1
16	CS2	Chip select for IC2
17	$\overline{RST}$	Reset
18	Vee	Negative voltage output
19	A	Power supply for LED+(4.2V) RA=0Ω
20	K	Power supply for LED (0V)

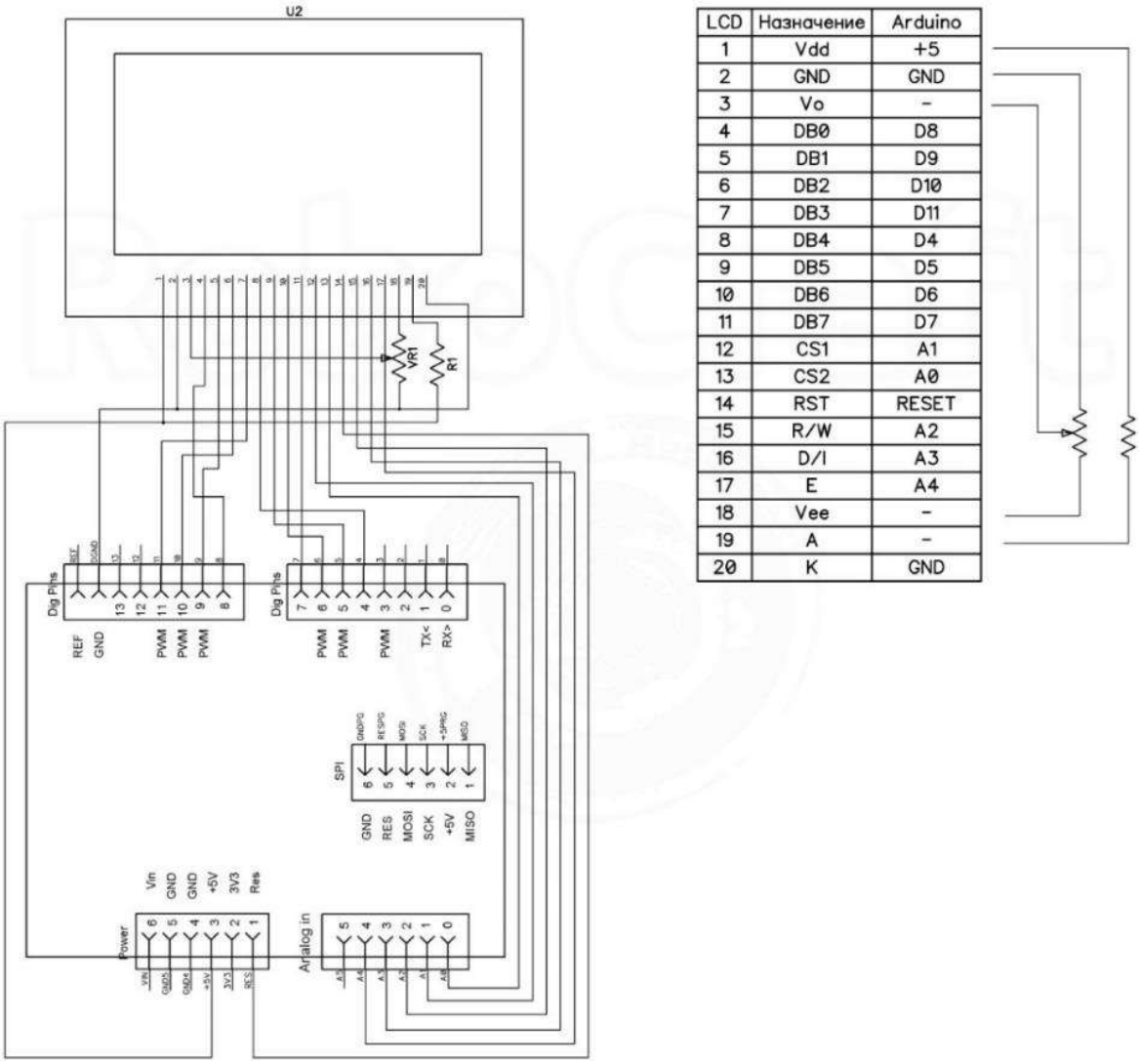
Рис. 5.2. Распиновка ЖКИ WG12864A на базе контроллера KS0107

Способ подключения дисплея к отладочным платам Arduino можно посмотреть в файлах конфигурации библиотеки по адресу: …\Мои документы\Arduino\libraries\openGLCD\config\ks0108\PinConfig_ks0108-Uno. В файле нас интересует назначение портов на отладочной плате рис. 5.3.

```
#define glcdPinData0        8
#define glcdPinData1        9
#define glcdPinData2        10
#define glcdPinData3        11
#define glcdPinData4        4
#define glcdPinData5        5
#define glcdPinData6        6
#define glcdPinData7        7

/* Arduino pins used for Control
 * default assignment uses the first five analog pins
 */

#define glcdPinCSEL1        A0
#define glcdPinCSEL2        A1

#if NBR_CHIP_SELECT_PINS > 2
#define glcdPinCSEL3         3   // third chip select if needed
#endif

#if NBR_CHIP_SELECT_PINS > 3
#define glcdPinCSEL4         2   // fourth chip select if needed
#endif

#define glcdPinRW           A2
#define glcdPinDI           A3
#define glcdPinEN           A4 // (A4 is also I2C SDA)
// Reset  - uncomment the next line if glcd module reset is connected to an Arduino pin
//#define glcdPinRES          A5    // optional s/w Reset control (A5 is also I2C SCL)
```

Рис. 5.3. Порядок назначения портов на отладочной плате Arduino-Uno для подключения к ЖКИ WG12864A на базе контроллера KS0107

Подключается дисплей довольно просто (если, конечно, не считать количества проводков). Как и в случае с символьным экраном кроме линий связи с мк понадобятся потенциометр регулировки контраста (обратите внимание на оригинальное подключение) и токоограничительный резистор для подсветки. Хотя некоторые его не ставят. Напрасно, кстати – падение на светодиодах подсветки, как гласит даташит, не более 4.6 вольт, и подавая туда все 5 вольт, рискуем их пожечь безвозвратно (рис. 5.4).



LCD	Назначение	Arduino
1	Vdd	+5
2	GND	GND
3	Vo	-
4	DB0	D8
5	DB1	D9
6	DB2	D10
7	DB3	D11
8	DB4	D4
9	DB5	D5
10	DB6	D6
11	DB7	D7
12	CS1	A1
13	CS2	A0
14	RST	RESET
15	R/W	A2
16	D/I	A3
17	E	A4
18	Vee	-
19	A	-
20	K	GND

Рис. 5.4. Схема подключения ЖКИ WG12864A на базе контроллера KS0107 к отладочной плате Arduino-Uno

Контроллер CS0107 он может организовать матрицу только 64x64. А у нас в дисплее вдвое большая 128x64. Значит стоят два контроллера. Один отвечает за правую половину экрана, другой за левую.

Он представляет собой этакую микросхему памяти, где все введенные данные отображаются на дисплее. Каждый бит это точка. Кстати, для отладки удобно юзать, выгружая туда разные данные, а потом разглядывая этот дамп).

Карта дисплея выглядит, как на рис. 5.5.

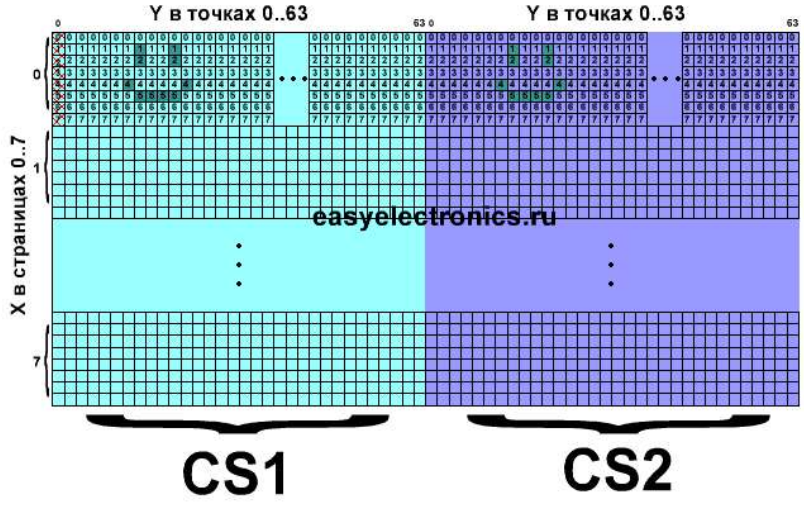


Рис. 5.5. Карта дисплея ЖКИ WG12864A на базе двух контроллеров KS0107

Байты укладываются в два контроллера страницами по 64 байта. Всего 8 страниц на контроллер. Так что для того, чтобы выставить точку с координатами на экране, например, X = 10, Y=61 надо вычислить в каком контроллере она находится. Первый до 63, второй после, если адрес во втором контроллере, то надо вычесть 64 из координаты. Затем вычислить страницу и номер бита. Страница это X/8, а номер бита остаток от деления (X%8). Потом нам надо считать нужный байт из этой страницы (если мы не хотим затронуть остальные точки), выставить в нем наш бит и вернуть байт на место (рис. 5.6).

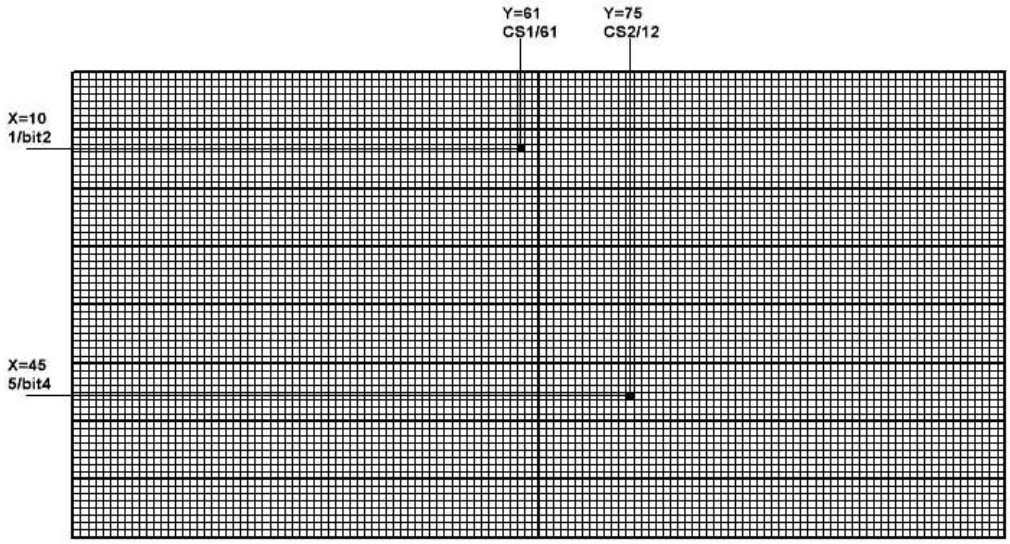


Рис. 5.6. Отображение двух пикселей на двух контроллерах

Протокол обмена

Тут все просто, без каких либо изысков. Выставляем на линиях RW, DI что мы хотим сделать, линиями CS1 и CS2 выставляем к кому обращаемся. На шину данных выдаем нужное число и поднимаем-опускаем линию строба. Опа! Впрочем, есть одна тонкость. Для чтения данных строб нужно дернуть дважды, т.к. предварительно данные должны попасть в регистр-защелку. Для чтения же флага состояния такой изврат не нужен. Вот примеры временных диаграм для разных режимов (рис. 5.7).

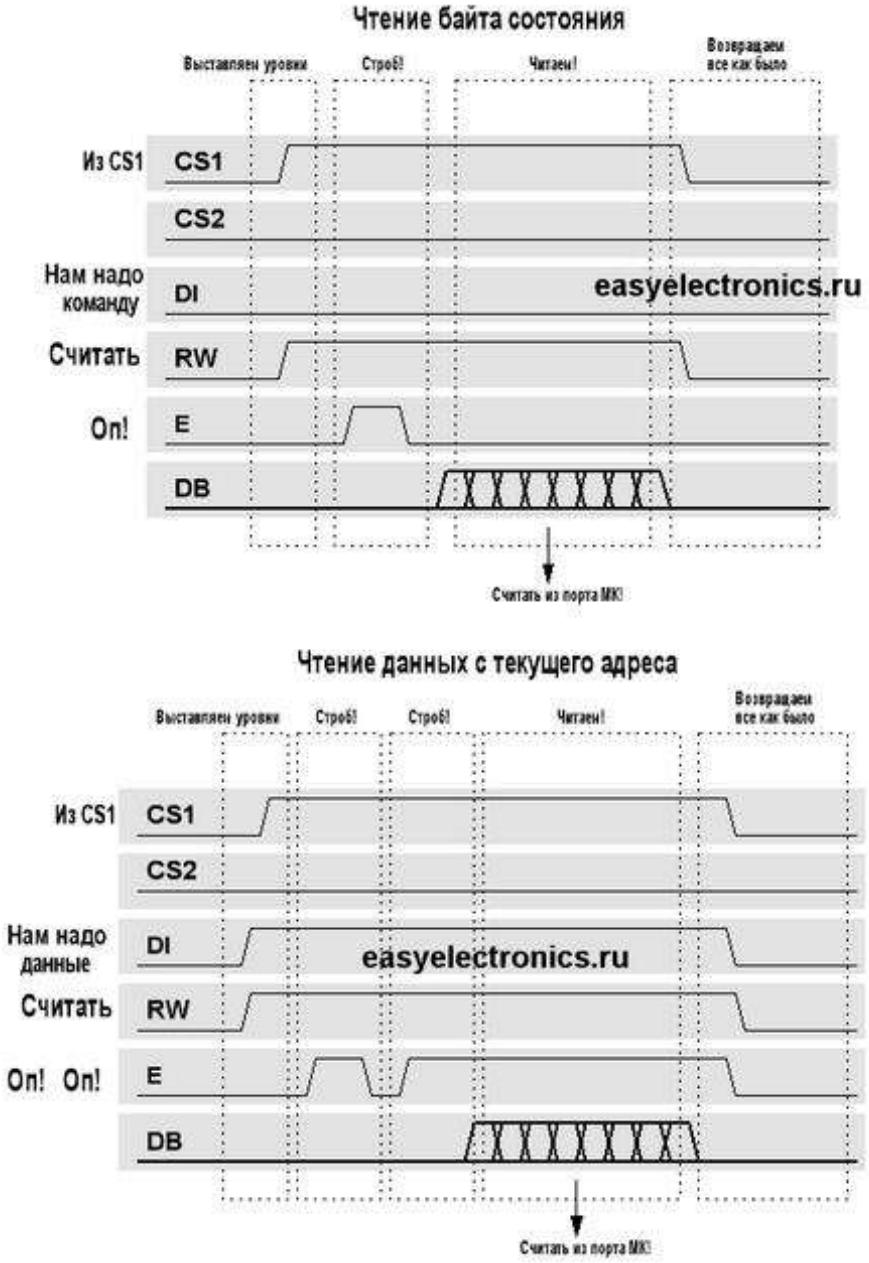


Рис. 5.7. Чтение данных с дисплея WG12864A

Запись данных представленна на рис. 5.8. Причем запись, в отличии от чтения, можно делать сразу в оба контроллера. Конечно одновременно писать данные в контроллер смысла имеет мало, разве что захочишь двойную картику получить. А вот команды обычно пишут срзау в оба.

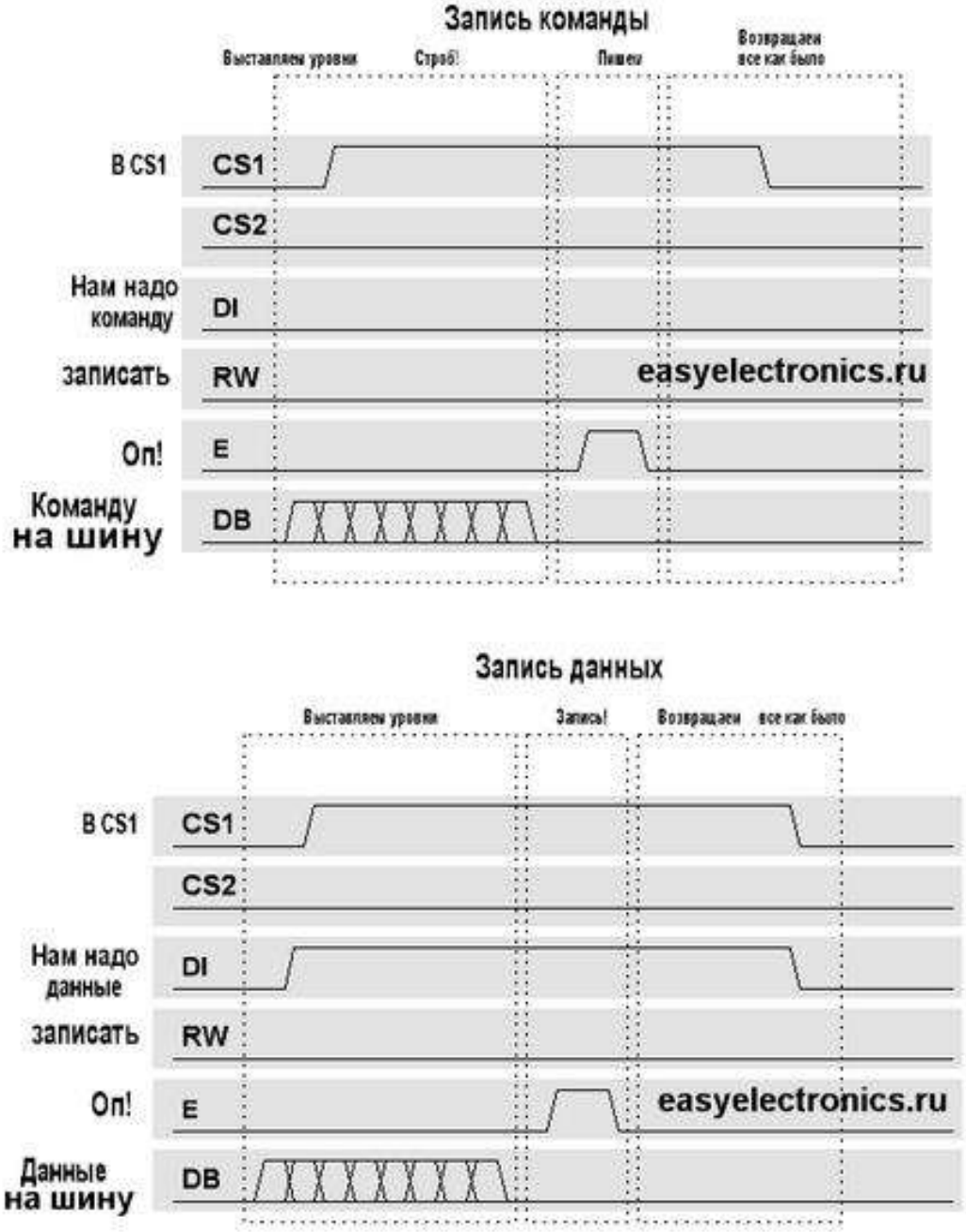


Рис. 5.8. Запись данных в дисплей WG12864A

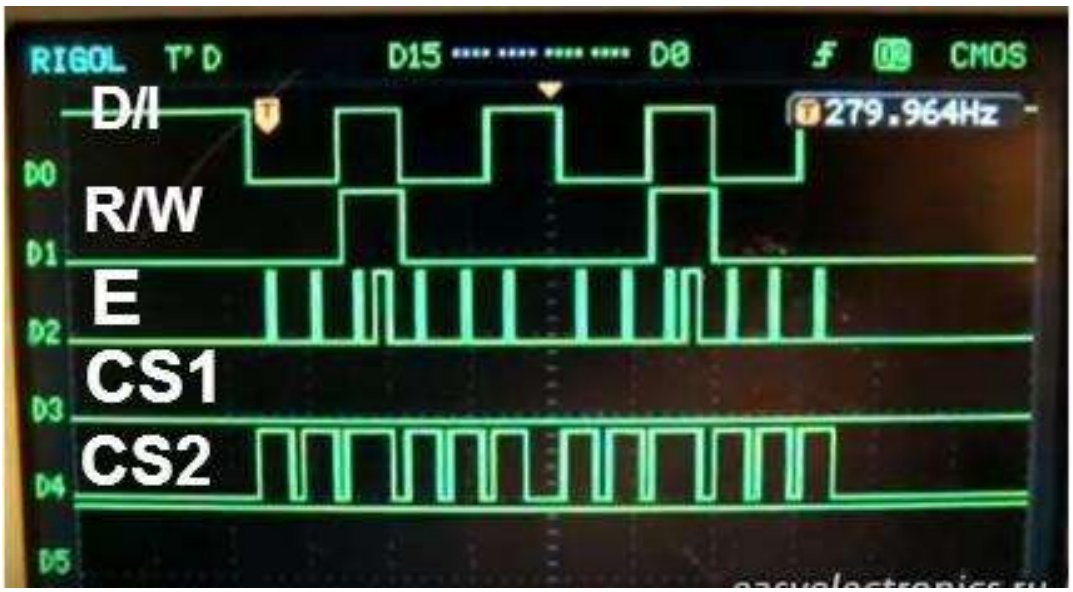


Рис. 5.9. ТаДиаграмма временных задержек сигналов дисплея WG12864A

Кстати, еще есть одна особенность – выводы CS могут быть инверсными (рис. 5.9). Это зависит от модели дисплея. Так что это надо учитывать. Временные диаграммы, т.е. сдвиг фронтов между собой по времени может быть разным у разных контроллеров. Где то быстрей, где то медленней. Но в целом 1мкс обычно хватает. В реале обычно меньше. Лучше поглядеть в даташите на конкретный контроллер (не дисплей, в даташите на дисплей обычно редко есть описание самого контроллера).

Там обычно есть таблица вида 5.1.

Таблица 5.1. Таблица временных задержек сигналов дисплея WG12864A

Characteristic	Symbol	Min	Typ	Max	Unit
E cycle	tcyc	1000	–	–	ns
E high level width	twhE	450	–	–	ns
E low level width	twIE	450	–	–	ns
E rise time	tr	–	–	25	ns
E tall time	tf	–	–	25	ns
Address set-up time	tas	140	–	–	ns
Address hold time	tah	10	–	–	ns
Data set-up time	tdsw	200	–	–	ns
Data delay time	tddr	–	–	320	ns
Data hold time (write)	tdhw	10	–	–	ns
Data hold time (read)	tdhr	20	–	–	ns

Где указаны максимально допустимые временные интервалы. Если они будут меньше или больше, то дисплей скорей всего не будет работать. Или будет работать с ошибками. Если у вас на дисплей в процедуре заливки или очистки экрана полезли всякие левые пиксели и прочие артефакты – значит тайминги не выполняются. Также рекомендую проверять тайминги на чтение и на запись. Делается это просто – производим последовательно сначала чтение, а потом запись обратно. Если ничего не изменилось – значит все ок. Появились искажения? Крутите временные задержки. Только рекомендую читать не пустоту вида 0x00, а что нибудь более веселое залить, например, шахматную доску из пикселей. По очереди 0x55 и 0xAA.

Система команд. Она тут простейшая (табл. 5.2).

Таблица 5.2. Система команд дисплея WG12864A

Команда	D/I	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Назначение
Отображение ВКЛ/ВЫКЛ	L	L	L	L	H	H	H	H	H	L/H	Управляет вкл/выкл отображения. Не влияет на внутреннее состояние и данные ОЗУ изображения. L: ВЫКЛ H: ВКЛ
Установить Адрес	L	L	L	H	Адрес Y (0 ~ 63)						Заносит адрес Y в счётчик адреса Y
Установить Страницу (адрес X)	L	L	H	L	H	H	H	Страница (0 ~ 7)			Заносит адрес X в регистр адреса X
Начальная Строка Отображения	L	L	H	H	Начальная строка отображения (0 ~ 63)						Скроллинг вверх. На сколько пикселей сдвинуть адресное пространство. При этом уехавшее вверх, за экран, вылезет снизу.

Продолжение таблицы 5.2

Чтение Состояния	L	H	BUSY	L	ON/OFF	RESET	L	L	L	L	Чтение состояния. BUSY L: Готовность H: Выполняется команда ON/OFF L: Отображение ВКЛ H: Отображение ВЫКЛ RESET L: Нормальный режим H: Сброс
Запись Данных Изображения	H	L	Данные для записи								Записывает данные (DB0:7) в ОЗУ данных изображения. После записи инструкции, адрес Y увеличивается на 1 автоматически.
Чтение Данных Изображения	H	H	Данные для чтения								Читает данные (DB0:7) из ОЗУ данных изображения на шину данных. После чтения адрес Y сам увеличивается на 1 автоматически

Инициализация дисплея элементарная, в отличии от HD44780, где надо переключать режимы, включать-выключать разные курсоры и отображения.

Надо после сброса задать начальные координаты (адрес точки и страница), значение скролинга (обычно 0) и включить отображение. При включении на дисплее может быть мусор. Поэтому отображение обычно включают после очистки видеопамяти.

Обычно ициализация, по командам, выглядит так:

– 0x3F – включить;

– 0x40 – Адрес Y=0;

– 0xB8 – Страница X=0;

– 0xC0 – Скролл = 0 (т.е. с самого верха).

Работа с пикселями медленная штука. Это же каждый надо считать, изменить, закинуть обратно. На обновление всего экрана в попиксельном режиме уйдет прорва времени. При полной загрузке процессора только на заливку попиксельно всего экрана уходит чуть ли не треть секунды.

Куда эффективней работать с дисплеем блоками. Т.е. берем и последовательно записываем сразу куски страниц, на глубину в несколько байт. Например, таким образом удобно рисовать текст. Правда при попиксельном выводе, когда текст попадает между страницами получается неудобно, приходится делать вдвое больше работы. Но все равно намного быстрей чем попиксельно. На порядок.

Рассмотрим пример отображения текста на графическом дисплее, для этого нам понадобится библиотека openGLCD установленная в папку по адресу …\Мои документы\Arduino\libraries\openGLCD.

Для работы с экраном предназначен ряд процедур:

– ***GLCD.Init(invert)*** – инициализация структур для нормального или инверсного режима;

– ***GLCD.GotoXY(x,y)*** – перемещает курсор в указанную позицию. 0,0 – позиция крайнего верхнего левого пикселя;

– ***GLCD.ClearScreen()*** – очистка экрана. // Графические функции (цвет WHITE чистит пиксели, цвет BLACK заполняют пиксели);

– ***GLCD.DrawCircle(x, y, radius, color)*** – рисование окружности с центром в позиции x,y;

– ***GLCD.DrawLine(x1,y1,x2,y2,color)*** – рисование линии из точки x1,y1 в x2,y2;

– ***GLCD.DrawVertLine(x, y, length, color)*** – рисование вертикальной линии;

– ***GLCD.DrawHoriLine(x, y, length, color)*** – рисование горизонтальной линии;

– ***GLCD.DrawRect(x, y, width, height, color)*** – рисование прямоугольника;

– ***GLCD.DrawRoundRect(x, y, width, height, radius, color)*** – рисование прямоугольника с закруглёнными краями;

– ***GLCD.FillRect(x, y, width, height, color)*** – рисование заполненного прямоугольника;

– ***GLCD.InvertRect(x, y, width, height)*** – инвертирование пикселей в области заданного прямоугольника;

– ***GLCD.SetInverted(invert)*** – установка режима рисования в инвертированном режиме;

– ***GLCD.SetDot(x, y, color)*** – рисование точки;

– ***GLCD.DrawBitmap(bitmap, x, y, color)*** – рисование картинки в указанном месте // Функции по работе с текстом;

– ***GLCD.SelectFont(font, color)*** – выбор шрифта, по умолчанию чёрного цвета;

– ***GLCD.PutChar(character)*** – печать символа в текущую позицию;

– ***GLCD.Puts(string)*** – печать строки в текущей позиции курсора;

– ***GLCD.Puts_P(string)*** – напечатать строку из программной памяти, в текущую позицию курсора;

– ***GLCD.PrintNumber(number)*** – печать цифры в текущую позицию курсора;

– ***GLCD.CursorTo(x, y)*** – перевод курсора в желаемую позицию.

Вот и всё! Вместе с библиотекой поставляется набор интересных примеров, которые безусловно помогут вам в вашей работе!

В папке библиотеки openGLCD имеются различные типы шрифтов различных размеров которые находятся по адресу (после установки библиотеки openGLCD) ….\Мои документы\Arduino\libraries\openGLCD\fonts\ Русский шрифт есть только в системном шрифте System5x7 (рис. 5.10).

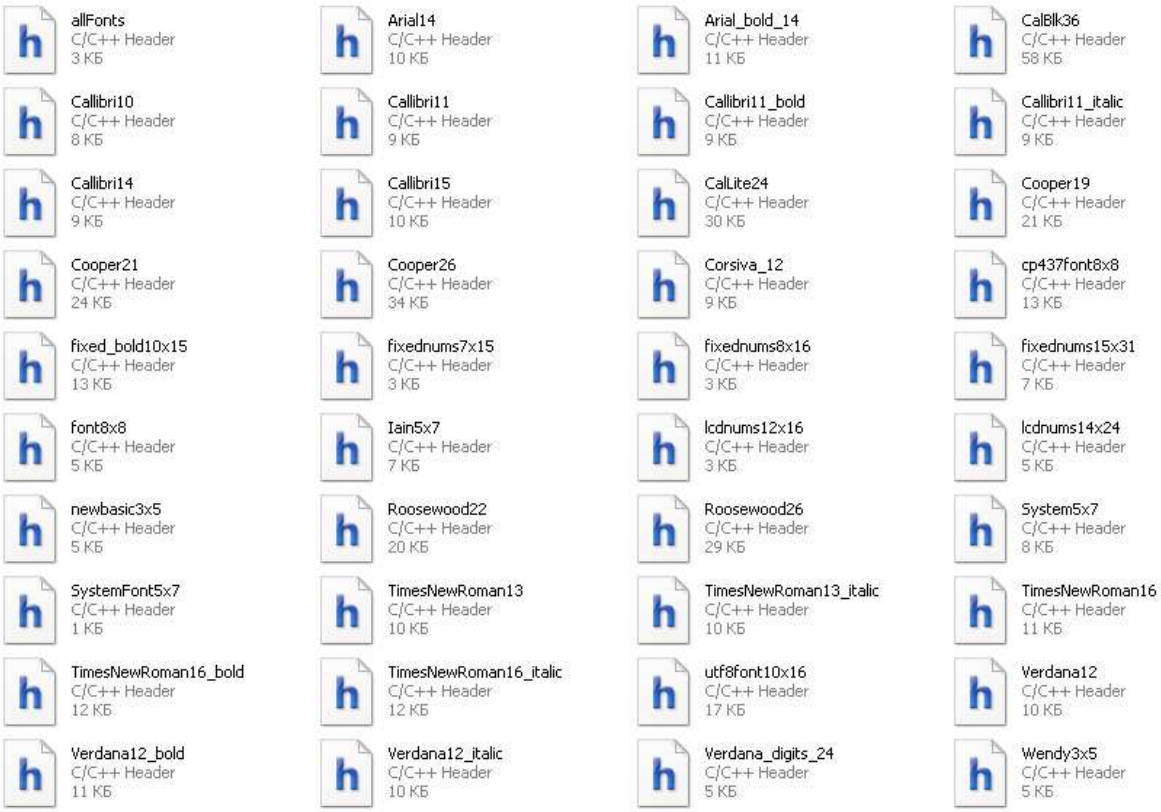


Рис. 5.10. Список шрифтов библиотеки. openGLCD

Пример:

```
#include <openGLCD.h>
char t = 0; // объявляем переменную для выводимых символов
char Y = 193; // для отображения буквы Ё большой
char y = 192; // и малой
void setup()
{
GLCD.Init(); // Инициализация дисплея
GLCD.SelectFont(System5x7); // Выбор параметра шрифта
// выводим в цикле все доступные символы один раз
GLCD.CursorTo(0, 0); // устанавливаем курсор в начало дисплея
for (int i = 0; i <= 191; i++)  {
t = i;
GLCD.print(t); // Выводим на экран все символы Font(System5x7)
}
GLCD.print(y); GLCD.print(Y);  // Вывод букв "ё" и "Ё"
}
void loop()
{
}
```

Задание для самостоятельной работы

1. Доработайте программный код, предложенный в примере, и обеспечьте отображение на *LCD* индикаторе отображение напряжения, считываемое с аналогового порта A3 в виде «Напряжение = ***».

2. Напишите свою фамилию, имя и отчество на ЖК-дисплее русскими буквами.

Создание шрифтов для LCD дисплея. Для того чтобы русифицировать другие шрифты библиотеки openGLCD или создать свои шрифты можно создать их с помощью приложения GLCDFontCreator2. Для запуска приложения необходимо запустить файл на start.bat, после чего появится окно программы, представленное ниже (рис. 5.11).

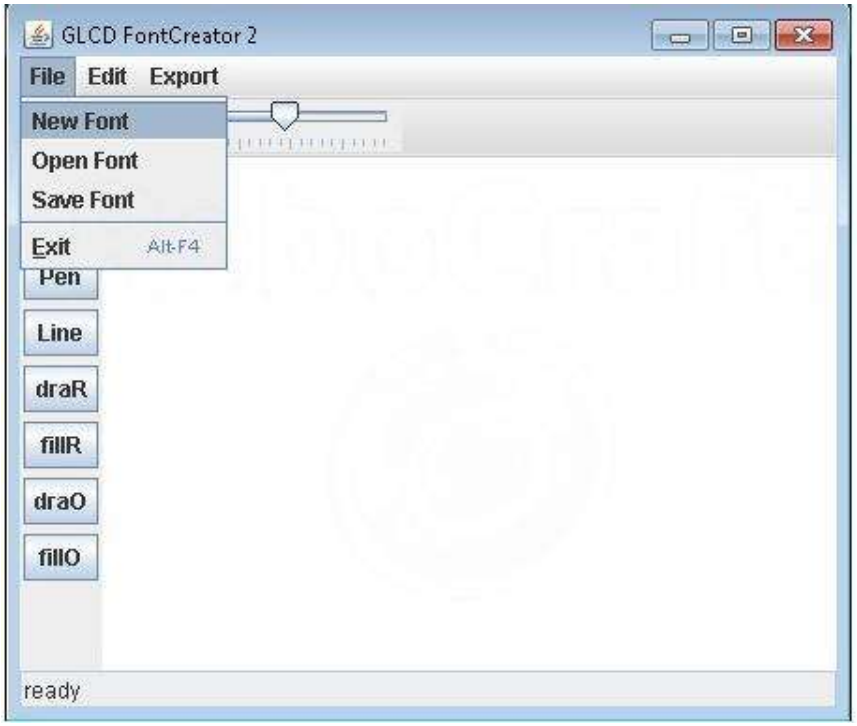


Рис. 5.11. Окно программы GLCDFontCreator2

После запуска программы создаем новый шрифт (программа редактировать существующие шрифты не может), для этого нажимаем New Font (рис. 5.12). Далее в появившемся окне указываем имя будущего шрифта.

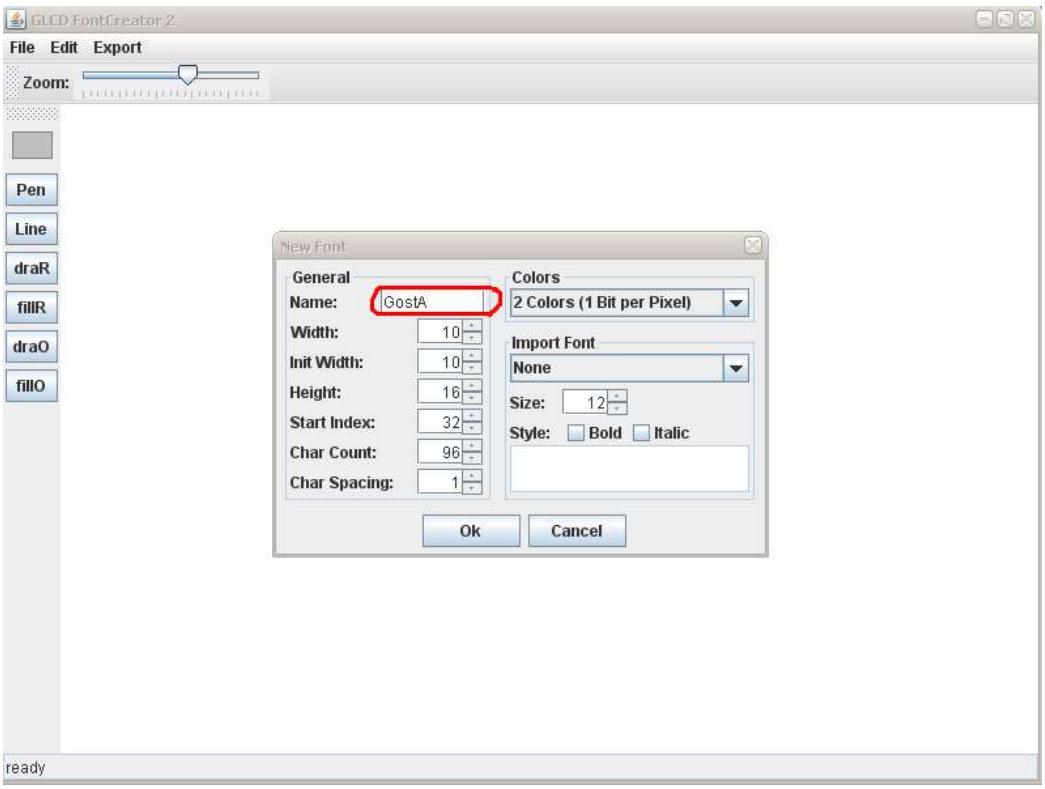


Рис. 5.12. Создание файла с именем GostA

Далее задаем ширину и высоту будущего шрифта в пикселях (рис. 5.13).

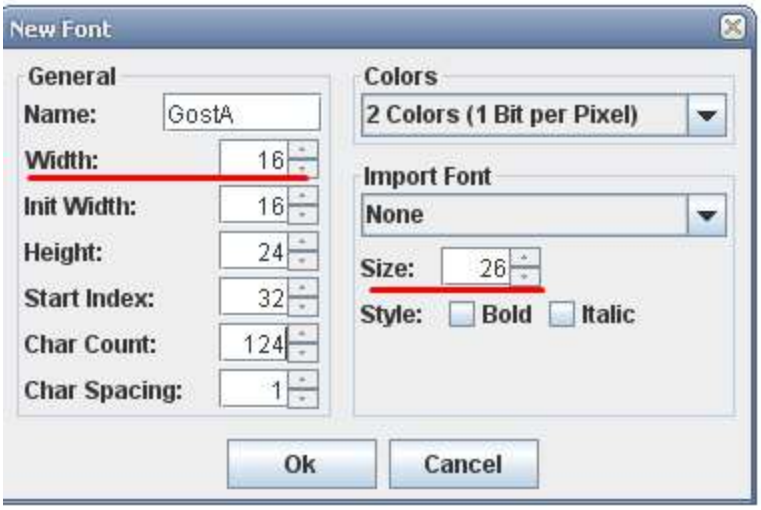


Рис. 5.13. Окно задания параметров

Указываем начальный порядковый номер символов создаваемого шрифта и количество символов в создаваемом шрифте (рис. 5.14).

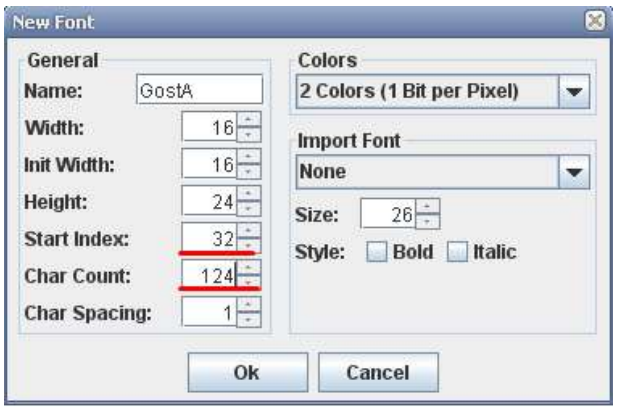


Рис. 5.14. Окно задания параметров

Выберем шрифт из уже установленных в системе (рис. 5.15).

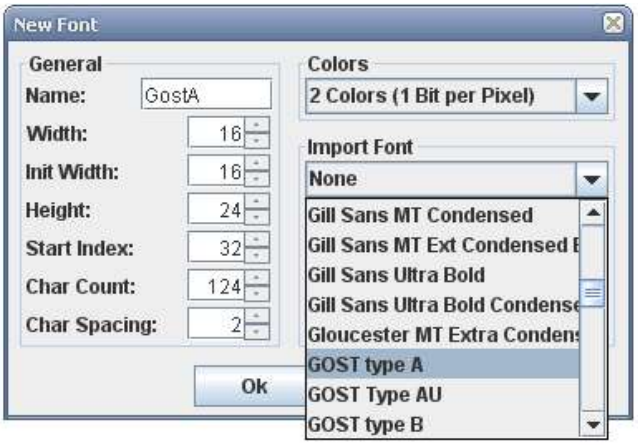


Рис. 5.15. Окно задания параметров

При необходимости можно сделать шрифт утолщенным или наклонным (рис. 5.16).

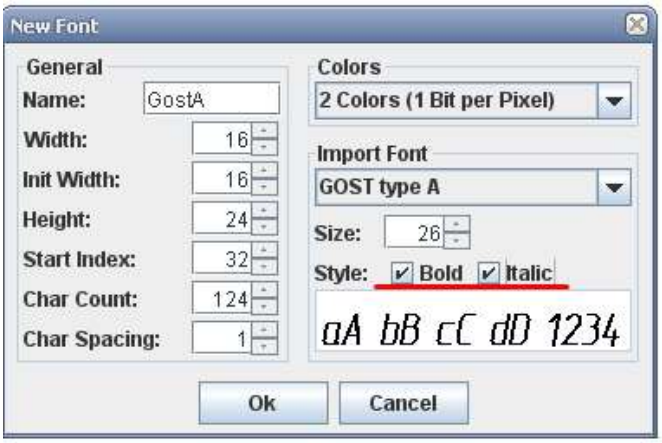


Рис. 5.16. Окно задания параметров

После нажатия на кнопку *OK* появится таблица, в которой приведены номера символов в шрифте в десятичном и шестнадцатеричном виде, а также приведены символы таблицы ASCII для соответствующего шрифта (как правило отображаются только символы до 127-го номера) (рис. 5.17).

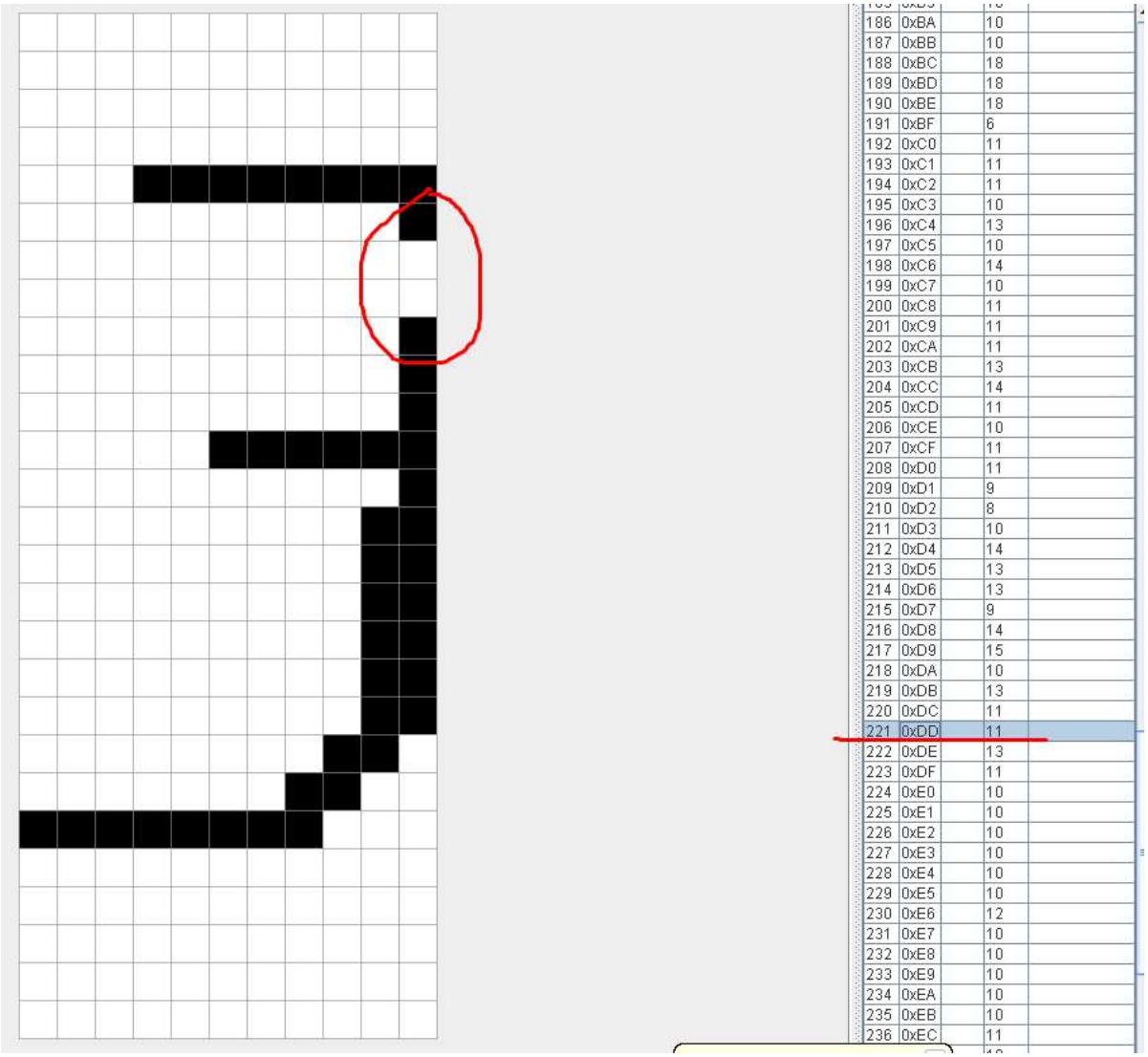


Рис. 5.17. Таблица номеров символов

Под номером 221 находится пиксельное изображение буквы «Э». Как видите программа отвела всего 11 пикселей на ширину символа и поэтому часть символа не прорисовалось. Для редактирования двойным нажатием на ячейке с цифрой 11 (в подчеркнутой на рис. 5.17 строке) открываем окно, в котором указывается ширина символа. Вводим новое значение равное 12 пикселям (рис. 5.18).

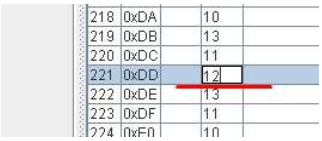


Рис. 5.18. Окно задания параметров

После нажатия клавиши *ENTER* появится двенадцатый столбец для отображения символа (рис. 5.19).

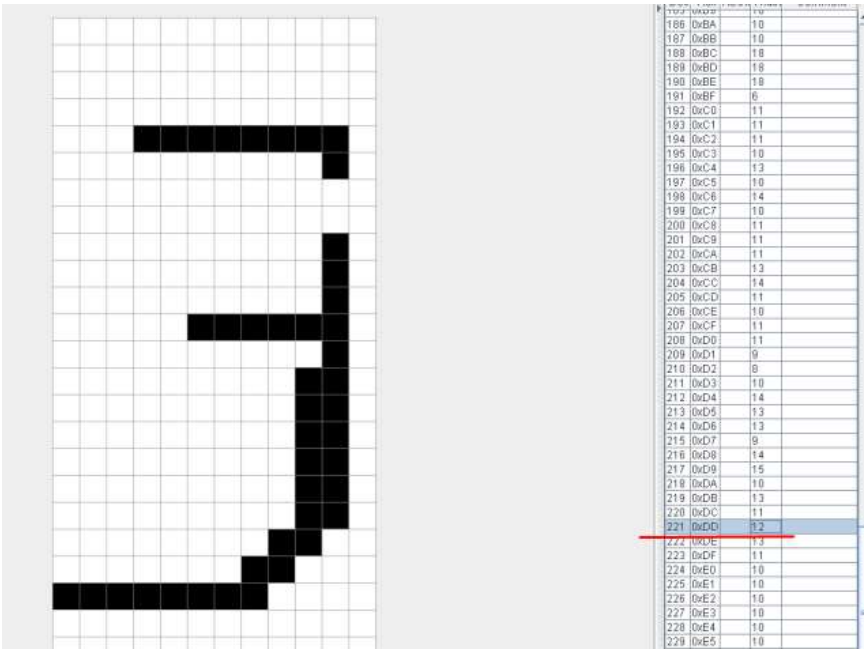


Рис. 5.19. Окно задания параметров

Далее необходимо дорисовать недостающие пиксели, для этого дважды нажимаем на иконку, представленную на рис. 5.20.

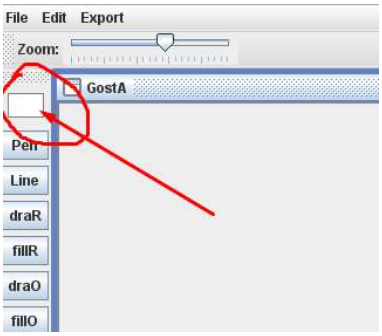


Рис. 5.20. Окно задания параметров

В появившемся окне (рис. 5.21) стрелкой указан текущий цвет для редактирования.

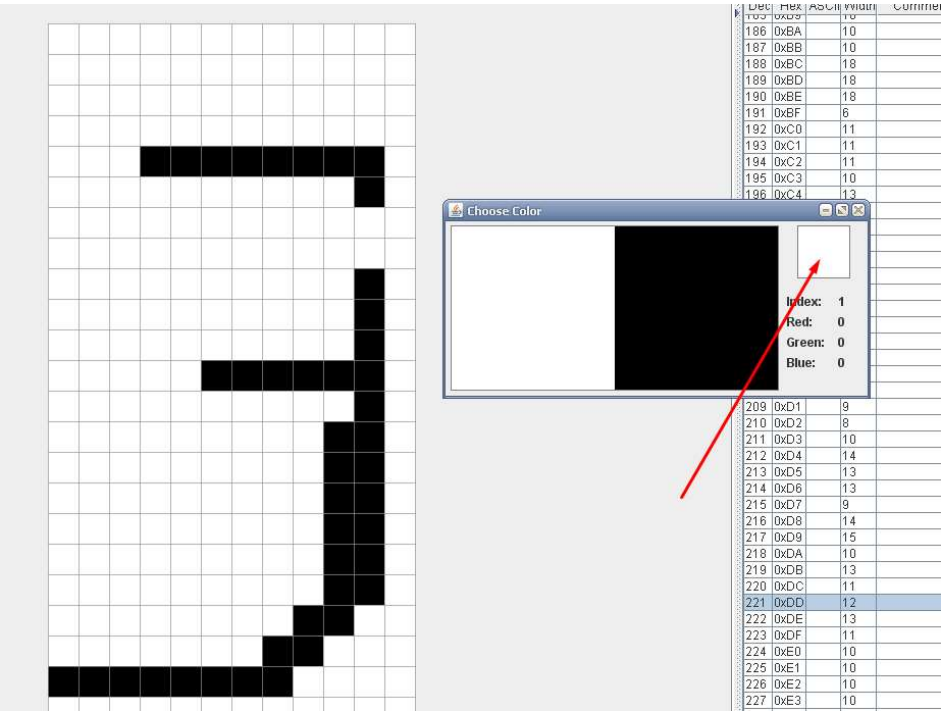


Рис. 5.21. Окно задания параметров

Если выбран белый цвет, то можно зарисовывать черные пиксели отображаемого символа, а после того, как дважды нажать на черное поле в окне «choose Color» можно будет дорисовывать черные пиксели в окне отображения символа (рис. 5.22).

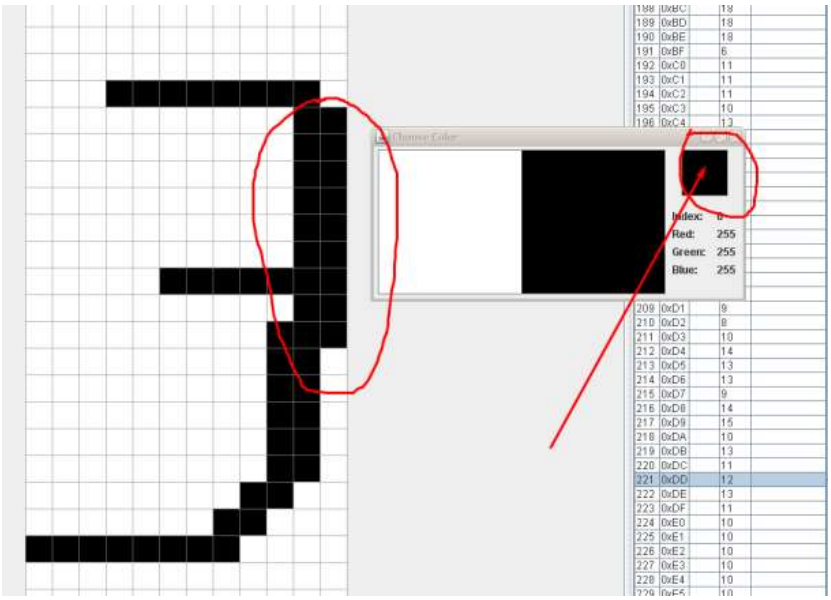


Рис. 5.22. Окно задания параметров

После того как символ отредактирован необходимо проверить остальные символы на наличие ошибок и при необходимости их отредактировать.

После редактирования экспортируем символы нажатием *Export Font* в меню программы (рис. 5.23).

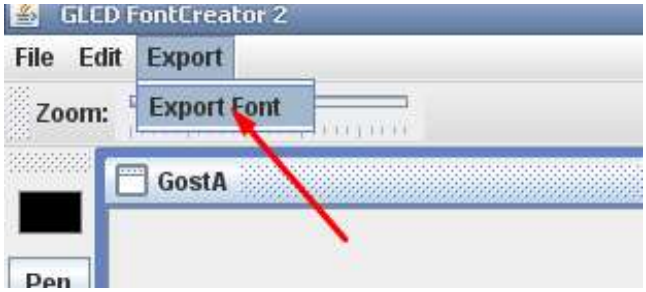


Рис. 5.23. Окно экспорта символов

Указываем имя создаваемого шрифта обязательно с расширением «.h» после чего нажимаем на кнопку Save и сохраняем созданный шрифт (по умолчания в папку программы GLCDFontCreator2).

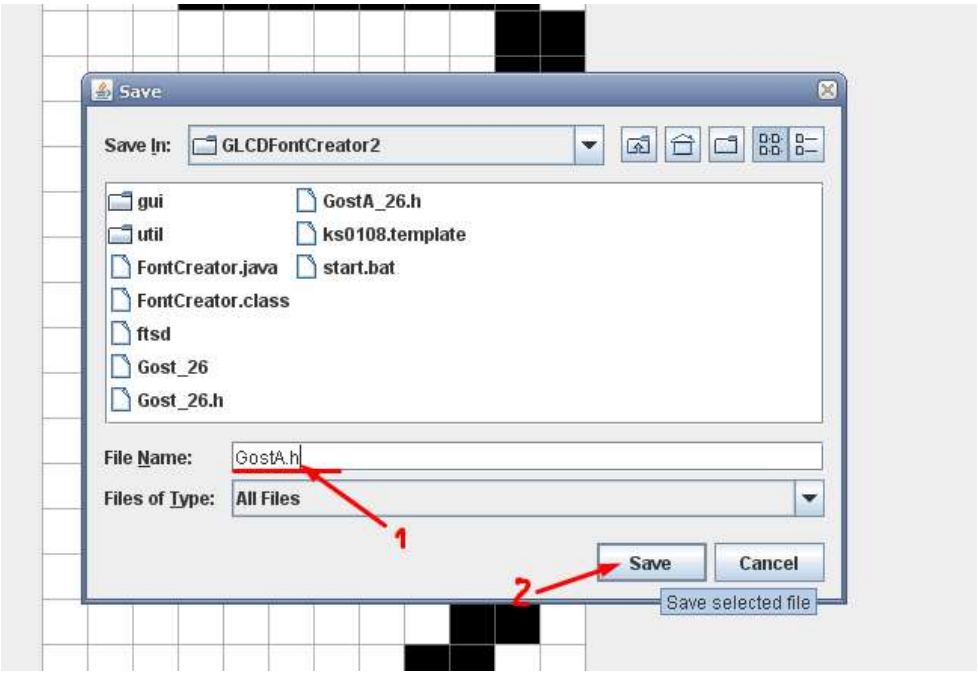


Рис. 5.24. Окно задания параметров

После сохранения файла скопируем его в папку шрифтов библиотеки openGLCD по адресу …\Мои документы\Arduino\libraries\openGLCD\fonts.

Если в тексте программ мы укажем созданный нами шрифт, то компилятор выдаст ошибку. На рис. 5.24 приводится отличие рабочего файла шрифта (слева) и созданного программой GLCDFontCreator2 (справа), отличия подчеркнуты. Таким образом, например:

вместо ***#ifndef GOSTA_H***	должна быть ***#ifndef _GOSTA_H***
вместо ***#define GOSTA_H***	должна быть ***#define _GOSTA_H***
вместо ***#define GOSTA_WIDTH 16***	должна быть ***#define GOSTA_WIDTH 16***
вместо ***#define GOSTA_HEIGHT 26***	должна быть ***#define_GOSTA_HEIGHT 26***
вместо ***static uint8_t GostA[] PROGMEM***	должно быть ***GLCDFONTDECL(GostaA)***

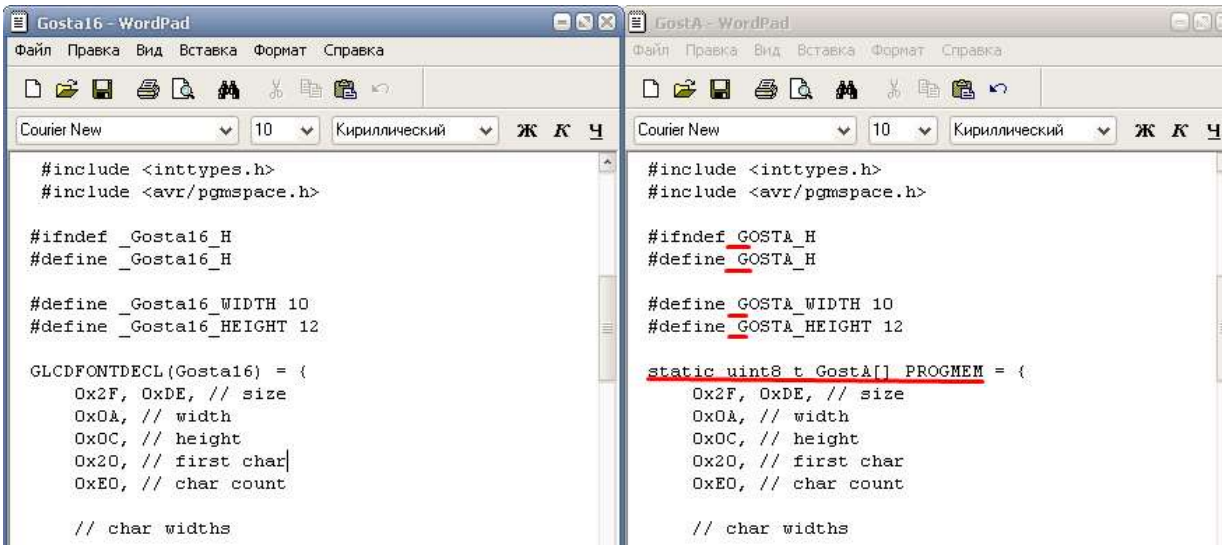


Рис. 5.25. Окно задания параметров

После внесения изменений в файл и сохранении изменений необходимо добавить созданный нами файл к существующим (рис. 5.25). Для этого в папке font библиотеки openGLCD откроем и отредактируем файл allFont.h.

```
#ifndef _allFonts_h_
#define _allFonts_h_

#include "System5x7.h"          // system font (fixed width)
#include "SystemFont5x7.h"      // backward compability System5x7 header
#include "Iain5x7.h"          // similar to system5x7 but proportional
#include "Arial14.h"            // proportional font
#include "Arial_bold_14.h"      // Bold proportional font
#include "Corsiva_12.h"
#include "Verdana_digits_24.h"  // large proportional font contains [0-9] and :

#include "Callibri10.h"
#include "Callibri11.h"
#include "Callibri11_bold.h"
#include "Callibri11_italic.h"
#include "Callibri14.h"
#include "Callibri15.h"
#include "Cooper19.h"
#include "Cooper21.h"
#include "Cooper26.h"
#include "TimesNewRoman13.h"
#include "TimesNewRoman13_italic.h"
#include "TimesNewRoman16.h"
#include "TimesNewRoman16_bold.h"
#include "TimesNewRoman16_italic.h"
#include "Verdana12.h"
#include "Verdana12_bold.h"
#include "Verdana12_italic.h"
#include "Roosewood22.h"
#include "Roosewood26.h"
#include "GostaA.h"
```

Рис. 5.26. Окно allFont.h

Добавим новую строку ***#include "GostaA.h"*** тем самым добавив новый шрифт в библиотеку openGLCD.

Пример:

```
#include<openGLCD.h>
char t = 0; // объявляем переменную для выводимых символов
void setup ()
{
GLCD.Init (); // Инициализация дисплея
}
void loop ()
{
for (int i = 30; i <= 126; i++)// Перебираем в цикле символы
{
t = i;
GLCD.SelectFont(CalBlk36); // Выбор параметра шрифта
GLCD.CursorTo(0, 0); // Устанавливаем курсор в начало дисплея
GLCD.print(t); // выводим на экран символы Font(CalBlk36)
```

Продолжение примера

```
GLCD.SelectFont(fixednums15x31); // Выбор параметра шрифта
GLCD.CursorTo(6, 1); // Устанавливаем курсор в начало дисплея
GLCD.print(t); // выводим на экран символы Font(fixednums15x31)

GLCD.SelectFont(Cooper26); // Выбор параметра шрифта
GLCD.CursorTo(4, 0); // Устанавливаем курсор в начало дисплея
GLCD.print(t); // выводим на экран символы Font(Cooper26)

GLCD.SelectFont(Roosewood26); // Выбор параметра шрифта
GLCD.CursorTo(7, 0); // Устанавливаем курсор в начало дисплея
GLCD.print(t); // выводим на экран символы Font(Roosewood26)

GLCD.SelectFont(Verdana_digits_24); // Выбор параметра шрифта
GLCD.CursorTo(0, 1); // Устанавливаем курсор в начало дисплея
GLCD.print(t); // выводим на экран символы Font(Verdana...)

GLCD.SelectFont(Cooper26); // Выбор параметра шрифта
GLCD.CursorTo(3, 1); // Устанавливаем курсор в начало дисплея
GLCD.print("t="); // выводим на экран надпись "t="
GLCD.print(i); // и десятичное значение отображаемого символа
Font(Cooper26)

GLCD.SelectFont(CalLite24); // Выбор параметра шрифта
GLCD.CursorTo(5, 0); // Устанавливаем курсор в начало дисплея
GLCD.print(t); // выводим на экран символы Font(CalLite24)

delay(500);  // пауза 0,5 секунды
GLCD.ClearScreen(); // очистка дисплея
}
}
```

Необходимо учесть при написании скетча, что использование большого количества разных шрифтов может привести к полному расходованию памяти программ микроконтроллера, так в приведенном выше примере использовалось шесть шрифтов и было израсходовано 98 процентов памяти (рис. 5.27).

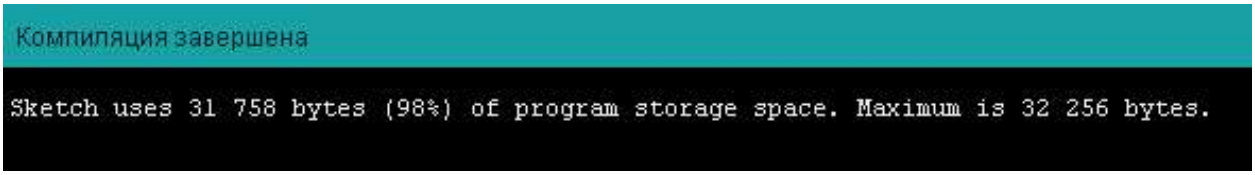


Рис. 5.27. Окно компиляции

Задание для самостоятельной работы

1. Создайте свой собственный шрифт и выведите его весь на ЖК-дисплей.
2. Напишите свою фамилию, имя и отчество на ЖК-дисплее русскими буквами, созданным вами шрифтом.

Контрольные вопросы

1. Назовите основные задачи графических дисплеев.
2. Какой основной минус управления дисплеем?
3. Перечислите назначение выводов ЖКИ WG12864A на базе контроллера KS0107.
4. Назовите основные процедуры для работы с экраном.
5. Почему работать с дисплеем блоками эффективнее, чем по-пиксельно?

Лабораторная работа № 6.
Отображение рисунка на графическом LCD дисплее WG12864A на базе контроллера KS0107

В этой лабораторной будет проведено подключение графического LCD дисплея к микроконтроллеру. Для среды Arduino IDE уже существует библиотека с примером по подключению графического LCD-дисплея WG12864A.

В итоге, мы напишем программу для микроконтроллера, которая будет управлять графическим LCD дисплеем и отображать различные рисунки.

Цель работы: приобрести практические навыки по управлению графическим LCD дисплеем и использовать его для вывода текстовых сообщений.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программы указанному в примере.
4. Выполнить задание для самостоятельной работы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.

Теоретические сведения

Вывод графических изображений. Библиотека u8glib позволяет выводить на дисплей произвольные графические изображения – рисунки, иконки, логотипы и прочее. Однако, сделать это не так просто, как может показаться на первый взгляд.

Подготовленный рисунок должен быть переведен в двоичный код и вставлен в текст программы в виде двоичного массива. Далее при помощи специальных команд содержимое этого массива может быть выведено на дисплей.

Как подготовить рисунок:

1. Сначала его надо нарисовать в любом графическом редакторе, поддерживающем сохранение в формате BMP. Рисунок должен быть монохромный, т. е. сохранять картинку надо с цветностью в 1 бит.

2. Далее нам потребуется приложение processing-1.5.1, с помощью которого запустим файл библиотеки openGLCD, файл находится ао адресу *…\Мои документы\Arduino\libraries\openGLCD\bitmaps\utils\glcdMakeBitmp* (рис. 6.1).

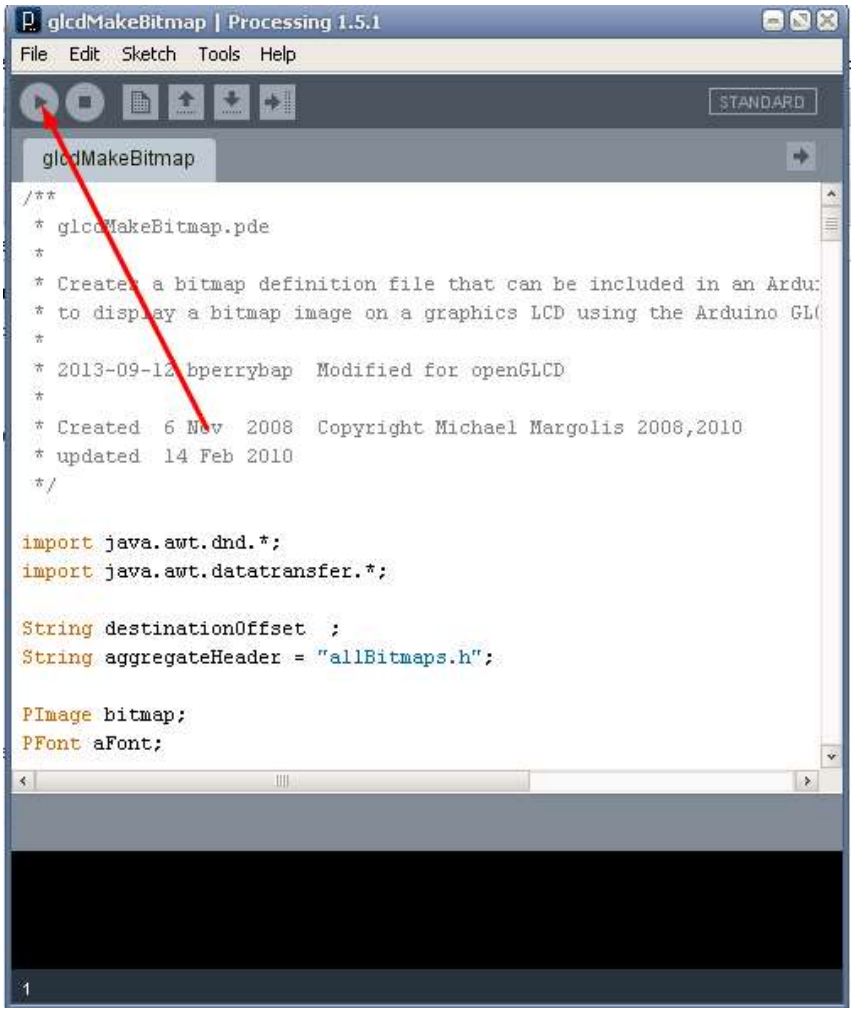


Рис. 6.1. Запуск программы glcdMakeBitmap

После нажатия на иконку «play» (рис. 6.1) после запуска появится окно программы glcdMakeBitmap, рис. 6.2.

Рис. 6.2. Окно программы glcdMakeBitmap

3. Далее требуется перетащить в окно программы подготовленный рисунок (рис. 6.3).

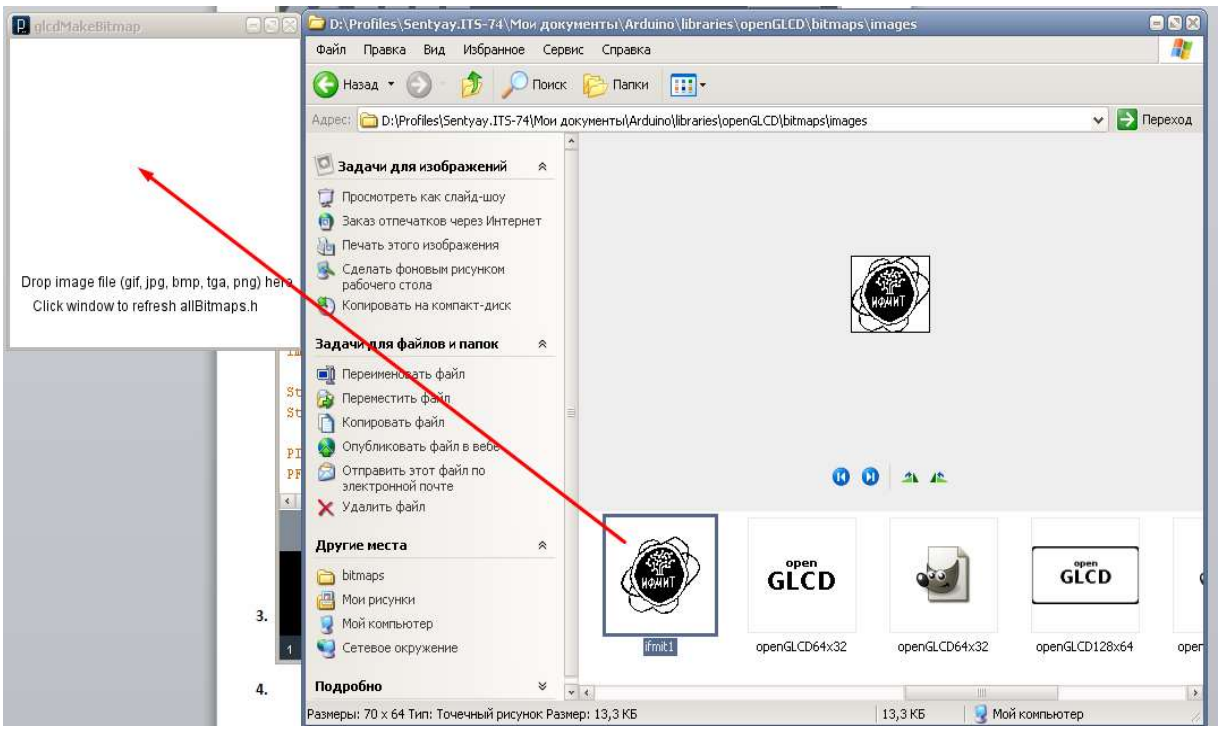


Рис. 6.3. Перемещение рисунка в окно программы glcdMakeBitmap

После перемещения в окне программы glcdMakeBitmap появится отображение рисунка с названием файла, который создается автоматически в библиотеке openGLCD по адресу …\Мои документы\Arduino\libraries\openGLCD\bitmaps (рис. 6.4).

Рис. 6.4. Отображение рисунка в программе glcdMakeBitmap с указанием поддерживаемых форматов и имени созданного файла

4. Теперь можно вывести картинку на дисплей.

Пример:

```
#include<openGLCD.h>//подключение графической библиотеки для дисплея
#include<openGLCD_Buildinfo.h>
#include<openGLCD_Config.h>
#include<bitmaps/ifmit1.h>//подключаем файл с картинкой
#define DELAY 1000// устанавливаем задержку 1 секунду
void setup ()
{
  GLCD.Init (); // Инициализация дисплея
}
void loop ()
{
  GLCD.ClearScreen(); //очистка дисплея
  delay(DELAY);//задержка 1 секунда
  GLCD.DrawBitmap(ifmit1, GLCD.CenterX - 32, 0); //отрисовка рисунка по центру
  delay(DELAY);//задержка 1 секунда
}
```

Задание для самостоятельной работы

1. Создайте рисунок при помощи программы Paint или Photoshop размером 128 на 64 пикселя.
2. При помощи программы *glcdMakeBitmap* конвертируйте рисунок для дальнейшего вывода его на дисплей.
3. Добавьте в рисунок свою фамилию, имя и отчество (также рисунком).
4. Создайте простую анимацию, на дисплее, чередуя отображение на нем нескольких рисунков.

Контрольные вопросы

1. Для чего служит библиотека u8glib?
2. Как правильно подготовить рисунок для вывода на дисплей?
3. Как правильно подключить графический LCD дисплей к микроконтроллеру?
4. В чем сложность использования библиотеки u8glib?
5. Как и зачем устанавливается задержка?

Лабораторная работа №7.
Инфракрасное дистанционное управление устройствами

В этой лабораторной узнаем о дистанционном управлении устройствами при помощи инфрокрасных пультов, о реализации этого способа управления в микроконтроллерах *Atmega328* в среде програмирования Arduino IDE.

В итоге, мы напишем программу для микроконтроллера, которая будет распозновать коды отправленные с дистанционного пульта и включать и выключать светодиоды на отладочной плате.

Цель работы: приобрести практические навыки по управлению устройствами при помощи дистанционных пультов.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программы указанному в примере.
4. Выполнить задание для самостоятельной работы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.

Теоретические сведения

Общее. Инфракрасное дистанционное управление (КДУ) применяется последние лет 30 практически во всей бытовой электронной аппаратуре. Иногда используется двусторонняя связь объекта управления и пульта (например, в кондиционерах с пульта передаются команды, а обратно текущие параметры), но мы не будем ее рассматривать в данной статье. Также мы не будем рассматривать

вопросы типа «что такое инфракрасный свет» (К) и т.п. Достаточно знать, что это свет. Весьма небольшой мощности и невидимом человеческому глазу диапазоне (но прекрасно наблюдаемому любой видеокамерой или цифровым фотоаппаратом). Как и всякий свет, в земных условиях он распространяется по прямой, т.е. не огибает предметы, не проходит через непрозрачные объекты, но в той или иной степени от них отражается.

Как и всякое излучение, характеризуется своей величиной (интенсивностью и т.п., но мы договорились не вдаваться в физику), которая зависит от типа источника и режима его работы, и падает с расстоянием. Поэтому расстояние, на котором можно уверенно принимать сигнал зависит от мощности источника и чувствительности приемника, плюс несколько побочных факторов. Для излучения используются К-светодиоды, работающие в импульсном режиме (ток в импульсе достигает нескольких ампер), для приема – фотодиоды с усилителями. Последние как правило, в аппаратуре закрываются К-фильтрами (на практике – красным или дымчатым пластиком) для снижения чувствительности к обычному солнечному или электрическому свету. Как всякий фильтр, он частично задерживает и полезное к излучению, поэтому от фильтра также зависит дальность работы ИКДУ. Почему в свое время было выбрано для массового использования именно КДУ, а не что-то другое? Да, были (и есть) системы, основанные на радиосвязи. Возможно даже в будущем они станут преобладать над К-системами, благодаря развитию таких способов коммуникаций как Bluetooth, ZigBee, MiWi (это не описка, не имелось в виду Wi-Fi!) и подобных. Но три с лишним десятилетия назад – К оказалось самым надежным, и что самое важное – самым дешевым видом беспроводной односторонней связи на короткие расстояния в пределах одного помещения. Ведь прежде всего оно начало использоваться в бытовых телевизорах. Наиболее массово к используется и поныне, однако среди ИКДУ царит полный балаган – каждая фирма разрабатывала свои собственные протоколы и работала на разных частотах модуляции. В результате мы имеем около 2-х десятков(!) совершенно несовместимых между собою систем, из которых наиболее массово, к счастью, используются 6-7.

Самые первые К-системы работали по принципу: «есть

излучение/нет излучения». Т.е. передавалась одна команда – «включить» или «выключить». Естественно, такой вариант совершенно не функционален, и не обладает приемлемой помехозащищенностью. Первое – потребовало разработки специальных последовательных протоколов связи. Второе – использованию модуляции (рис. 7.1), т.е. грубо говоря – излучение (ток) светодиода модулируется дважды, сначала несущей частотой (в районе 30-60 кГц), которая в свою очередь, модулируется последовательностью битов команд. Замечу что существовали КДУ системы и без модуляции, например, протокол ITT.

Исторически сложилось, что, когда передается «1» это называют «Mark», когда передача отсутствует – это называют «Space». В рассматриваемых далее системах КДУ вовремя Space на светодиод ничего не подается, он выключен, а вовремя Mark – подается несущая частота. Т.е. светодиод вспыхивает с частотой 30-50кГц.

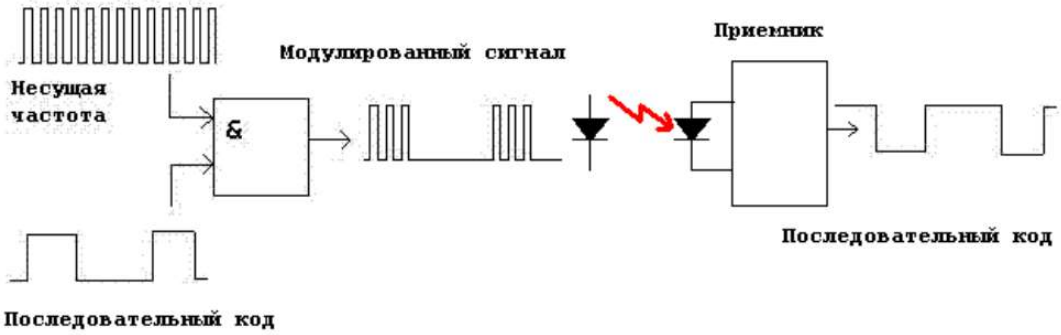


Рис. 7.1. Модуляция инфракрасного сигнала

Со стороны приемника – сигнал усиливается и демодулируется. В результате на выходе имеем исходную последовательность импульсов. Как правило, все современные приемники выдают инверсный сигнал, т.е. состояние Mark соответствует логическому нулю, а состояние Space – логической единице. В реальности – «К-приемник» несколько более сложное устройство чем просто фотодиод и усилитель. В нем имеется также ограничитель уровня (лимитер) и автоматическая регулировка усиления (АРУ), полосовой фильтр и демодулятор (детектор) с компаратором на выходе. Выпускаются готовые К-приемники таким фирмам как Vishay, Siemens, Sharp, а также многочисленными китайскими «тружениками».

Учтите, что с протоколами без модуляции такие К-

приемники использовать нельзя, но мы такие и не будем рассматривать в этой лабораторной (по разным причинам, в т. ч. и из-за присущих без модуляционным системам недостатков). Для правильной работы АРУ, в начале посылки практически во всех модуляционных протоколах передается «преамбула».

Вряд ли стоит останавливаться на кодировании, т.е. изготовлении самого пульта ИКДУ – во-первых, это сложно чисто конструктивно, ибо изготовить нормальный красивый пластмассовый корпус с кнопками и батарейным отсеком – задача не простая (и не для радиолюбителей). Во-вторых – эта задача существенно проще задачи декодирования, достаточно взять описание протокола, и по нему все сделать. В декодировании сложнее, особенно, кода надо обеспечить работу с заранее неизвестным пультом (даже если известен стандарт, по которому он работает). В первую очередь потому, что разброс у пультов достаточно велик, и иногда даже выходит за рамки позволенного. Простой пример – пульт от CD-плеера Philips CD800. Работает он по самому что ни на есть стандартному протоколу RC5, использующего манчестерское кодирование. При таком кодировании, импульсы могут иметь только две длительности – T и 2T, тем не менее, начальный импульс, который должен иметь одинаковые Mark и Space длительностью T, получается весьма несимметричный (рис. 7.2). Как видно на осциллограмме ниже – первый импульс чуть больше1мс, затем пауза 0.74мс, затем широкий импульс примерно 1.8мс. По стандарту – первые два должны были бы быть 0.88 мс.

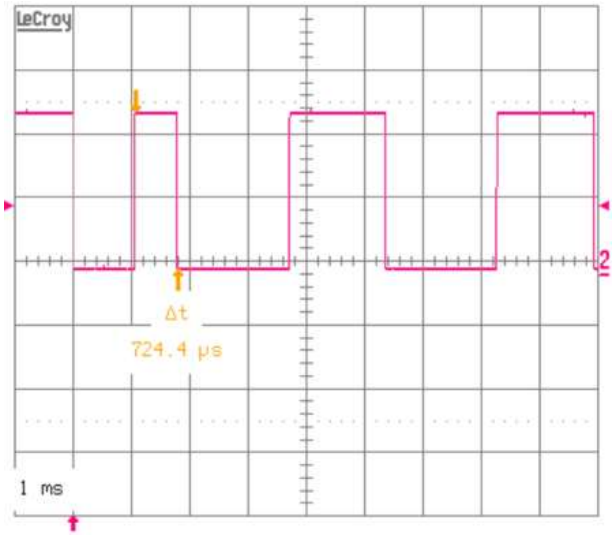


Рис. 7.2. Временная диаграмма сигнала с пульта

Во вторую очередь – когда встает вопрос универсальности. Т.е. когда хочется иметь свой аппарат, который можно настроить на любой имеющийся пульт. Здесь есть несколько вариантов: или просто пульту говорится примерно следующее: «ты работаешь в кодировке NEC для телевизора», после чего микроконтроллер пульта все делает «по стандарту». К сожалению, это имеет много недостатков. Например, я не смог настроить таким методом один такой «универсальный» пульт на телевизор – точнее вроде бы все работает, только кнопки «Громкость –» и «Громкость+» работают наоборот. Согласитесь, это несколько напрягает. Другие варианты работают по принципу – «посвети имеющимся пультом в приемник». В «универсальных пультах» как правило, используется простая оцифровка импульсов с приемника, с выделением начала и конца посылки, и запоминание этой последовательности (иногда делаются простые преобразования для уменьшения объема данных) в памяти. Т.е. своего рода «цифровой магнитофон». И лишь в некоторых делается по-настоящему «разбор полетов» – анализ типа протокола и полное декодирование. Оба эти способа имеют как свои достоинства, так и недостатки.

Первый способ «оцифровка» более простой, но требует достаточно большого запоминающего устройства для хранения оцифрованных данных (есть способы сильно сократить объем, но все равно он не мал), и хорошо работает, когда нет ограничения на размер памяти и «хранителя» – например именно такой способ я использовал когда-то для управления видеомагнитофоном от компьютера для записи рентгенограмм и УЗИ в медицинской диагностике. Аналогичное решение (и также на видеомагнитофоне) использовали разработчики стримера «Арвид».

Для использования готовых «чужих» пультов своих конструкциях, (и при отсутствии памяти для запоминания оцифрованных последовательностей), наиболее интересна настройка на имеющийся пульт с «разбором полетов», т.е. с полным декодированием. Поэтому в данной лабораторной будут описаны некоторые наиболее часто встречающиеся протоколы (кодировки), и способы программного декодирования на простых микроконтроллерах.

Описания протоколов. В Интернете масса описаний очень распространенной филипсовской кодировки RC5, но практически отсутствует информация по другим протоколам. Или же она

настолько мизерна и разбросана по разным местам, что понять что-то сложно. Собрав все что можно воедино, постараемся дать, наиболее полную информацию для того, чтобы этим можно было пользоваться. Не будем вдаваться в 7-уровневую модель OSI, разделим условно каждый протокол на две части – битовое кодирование и логический состав. Первое – это как кодируется передаваемый бит информации, а второе – что он означает. Рассмотрим сначала битовое кодирование (рис. 7.3). Все протоколы можно условно разделить на две категории – использующих бифазное кодирование («Манчестерский» код) и использующие временной код. При бифазном кодировании (используется в филипсовских RC5 иRC6, Nokia NRC17 и других), каждый бит передается в течении фиксированного времени, значение бита – определяется направлением перехода в середине этого времени. На диаграмме ниже видно, как это происходит – показана передача как разных бит, так и нескольких одинаковых бит подряд. Как мы видим – в середине битового интервала всегда есть переход, направление которого определяет значение бита. Между битами – переход есть только при передаче одинаковых бит. Это доставляет некоторые неудобства при декодировании, т.к. не всегда известен момент перехода от одного бита к другому. Было бы все просто, если бы длительности импульсов были всегда одинаковы (как это требуется по стандарту). Но на практике это не так, и разброс весьма существенный.

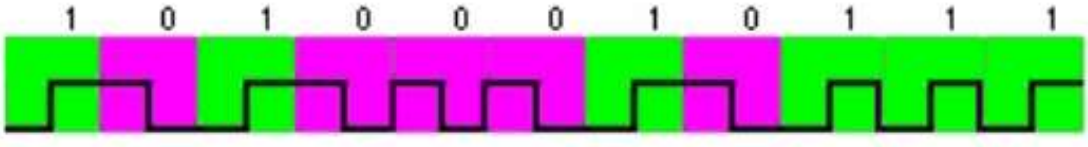


Рис. 7.3. Битовое кодирование RC-5, RC-6

Кодирование временными интервалами (Sony, NEC, Samsung, etc.) предполагает *ВСЕГДА* наличие импульса на каждом передаваемом бите, а за ним паузы (или наоборот), что абсолютно четко определяет начало и конец передаваемого бита, а его значение определяется соотношением времени импульса и времени паузы (собственно, это разновидность ШМ – широтно-импульсной модуляции с переменной частотой). В отличие от манчестерского кодирования, длительность передачи нуля и единицы разная, следовательно, и общая

длительность передаваемой команды может быть разной (есть способы этого избежать, но об этом ниже).

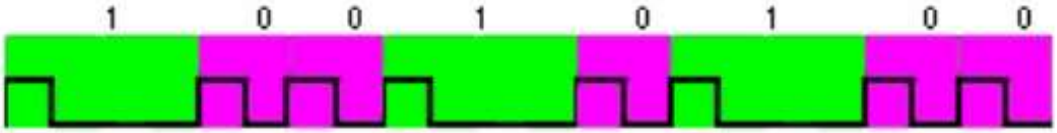
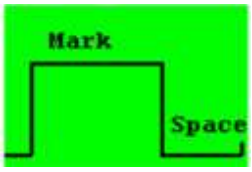


Рис. 7.4. Битовое кодирование Sony, NEC, Samsung

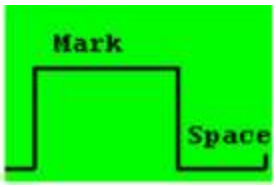
Почти во всех протоколах, в начале каждой команды передается стартовый бит (его еще часто называют «преамбула» или «AGC pulse» – импульс, для установки автоматической регулировки усиления в приемнике). Это импульс и пауза определенной длительности. К огромному счастью, у большинства протоколов временные параметры стартового импульса различны, что позволяет по первому же импульсу детектировать тип протокола (но учтите, что это не в 100% случаев!) Ниже представлена таблица 7.1 известных длительностей преамбул.

Таблица 7.1. Длительности преамбул

	Протокол	Mark time, ms	Space time, ms	Примечание
	NEC	9	4.5	Используется и многими другими
	NEC-Repeate	9	2.25	При повторе
	JVC	8.4	4.2	При повторе преамбула отсутствует
	Samsung	4.5	4.5	
	RCA	4	4	
	Panasonic (JAP)	3.6	1.5	
	Funai-2	3.2	3.2	
	RC6	2.67	0.89	
	Sony (SIRC)	2.4	0.6	
	RC5 Nokia NRC17	0.89 0.5	0.89 2.5	
	RC-MM	0.42	0.28	

В следующей таблице 7.2 даны длительности информационных бит при кодировании временными интервалами.

Таблица 7.2. Длительности информационных бит

		1		0	
	Протокол	Mark time, ms	Space time, ms	Space time, ms	Mark time, ms
	NEC	0.56	1.6	0.56	0.56
	JVC	0.53	1.6	0.53	0.53
	Samsung	0.65	1.5	0.65	0.65
	RCA	0.5	2	0.5	0.5
	Panasonic (JAP)	0.4	1.2	0.4	0.4
	Funai-2	0.8	2.4	0.8	0.8
	RC6	0.45	0.45	0.45	0.45
	RC6 Trailer	0.9	0.9	0.9	0.9
	Sony (SIRC)	1.2	0.6	0.6	0.6
	RC5	0.9	0.9	0.9	0.9
	Nokia NRC17	0.5	0.5	0.5	0.5
	RC-MM	0.17	0.28/0.44/0.62/0.78 (см. Описание протокола)		

Протокол NEC. Один из самых распространенных протоколов. Встречается в аппаратуре таких фирм, как Funai, Akai, Fisher, Goldstar, Hitachi, Kenwood, Onkio, Teac, Yamaha, Sanyo, Canon, Orion, Apex, Eltax, и много других. Этот протокол настолько распространен в аппаратуре из страны Восходящего Солнца, что его часто называют «японский протокол». В каждой посылке передается 8 бит адреса и 8 бит команды (рис. 7.5), причем адрес и команда передаются дважды – в прямом и инверсном виде (это кроме проверки валидной передачи, делает одинаковой общую длительность любой посылки).

Рис. 7.5. Кодировка сигнала по протоколу NEC

При непрерывном нажатии на кнопку, только первая посылка передается как показано на рисунке выше. Все остальные передаются в виде преамбулы с укороченной паузой и завершающим импульсом (рис. 7.6), с периодом 110 мс.

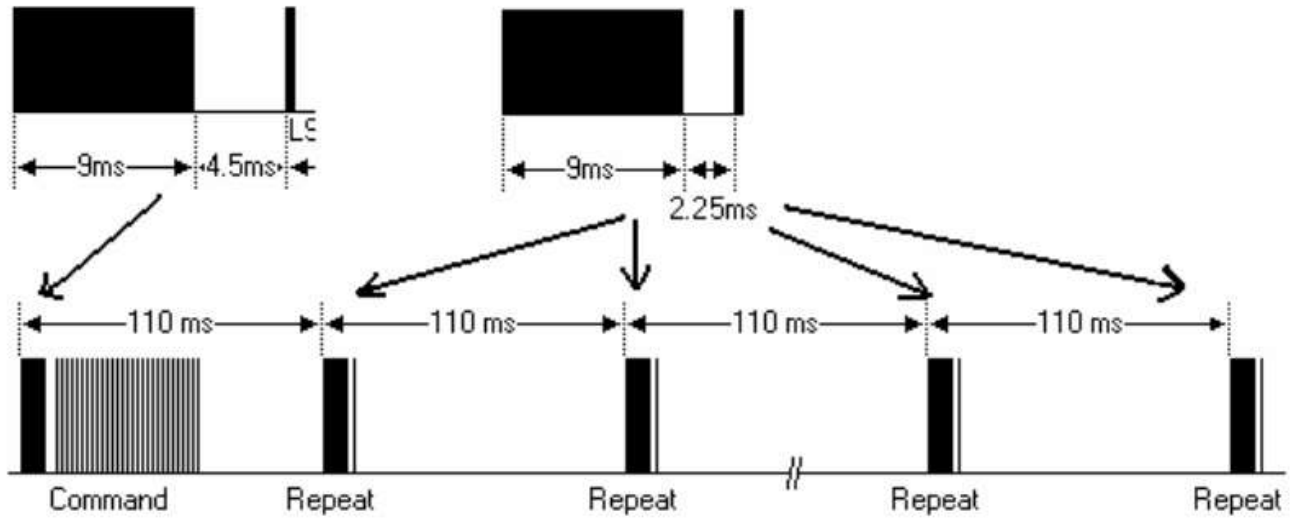


Рис. 7.6. Кодировка сигнала при непрерывном нажатии на кнопку

Кодирование бит – длительностью паузы (1.6 мс или 0.56 мс) между импульсами фиксированной длительности (0.56 мс) показаны на рис. 7.7.

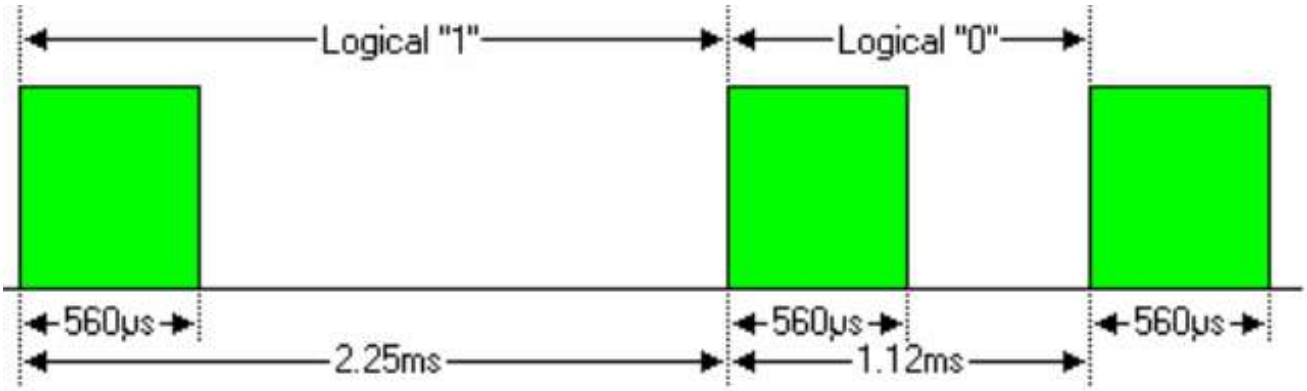


Рис. 7.7. Длительность импульсов логического 0 и 1

Протокол JVC. Очень похож на NEC, и даже имеет сходные длительности. Но адрес и команда передаются только один раз. При повторе – посылка повторяется без преамбулы (рис. 7.8).

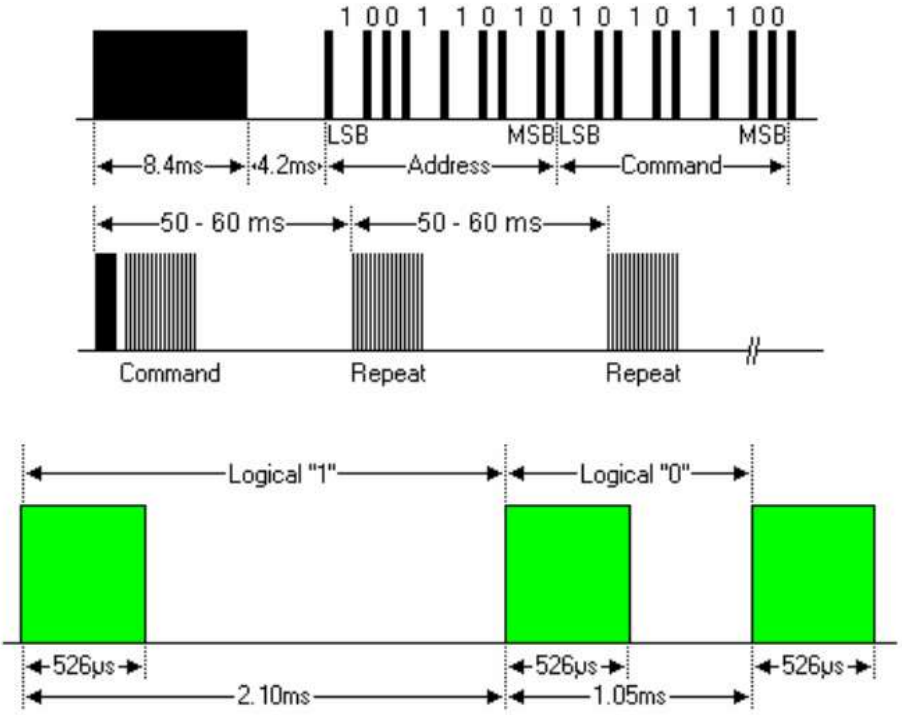


Рис. 7.8. Кодировка сигнала по протоколу JVC

Протокол NOKIA NRC17. 8 бит команды, 4 бита адреса и 4 бита «суб-кода» передаются бифазным (манчестерским) кодированием передаваемых бит (рис. 7.9).

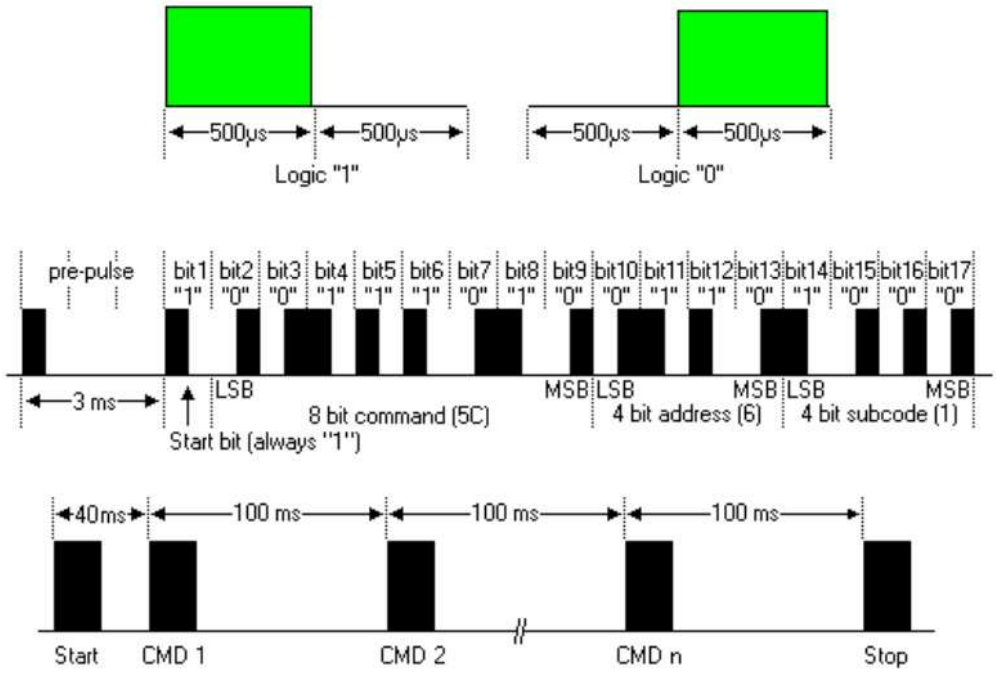


Рис. 7.9. Кодировка сигнала по протоколу NOKIA NRC17

Особенность протокола – для индикации разряженной батареи, преамбула передается с удлиненной на 1 мс паузой (т.е. пауза не 2.5, а 3.5 мс и длительность преамбулы 4 мс вместо 3 мс).

Протокол SAMSUNG. Этот протокол также очень похож на NEC (рис. 7.10), но инверсным повторяется только команда. Адрес просто повторяется. Почему сделано именно так – загадка.

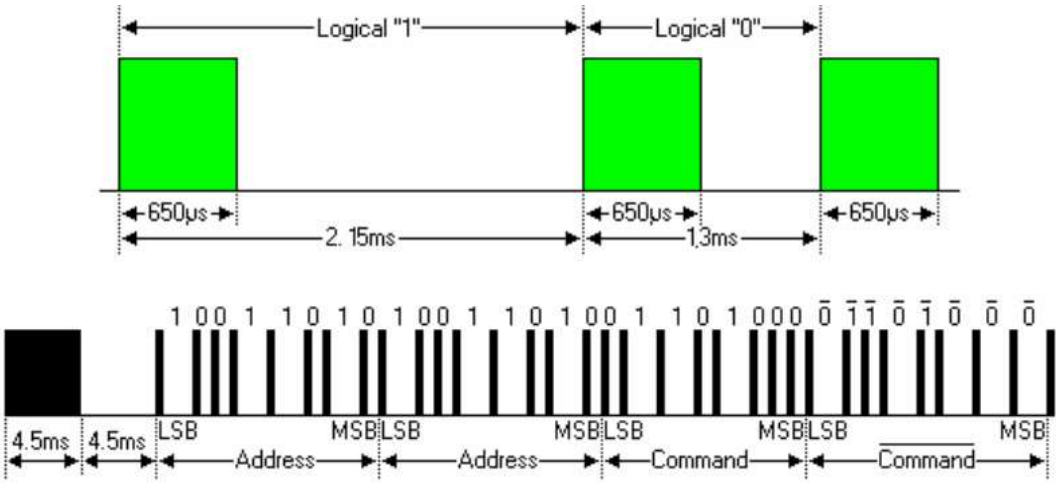


Рис. 7.10. Кодировка сигнала по протоколу SAMSUNG

В качестве повтора используется преамбула и два импульса (рис. 7.11).

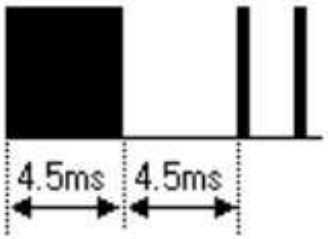


Рис. 7.11. Кодировка сигнала при повторе

Данный протокол очень любят использовать в телевизорах марки «Noname» сделанных в Китае.

Протокол SHARP. В отличие от других протоколов, в этом во 1-х отсутствует преамбула, а во 2-х – каждая команда передается двумя посылками, с периодом 40 мс, при этом во второй посылке биты адреса и команды инверсные (рис. 7.12). Кодирование битов –

ШИМ. В конце каждой посылки передаются дополнительные 3 бита – один для расширения системы команд и два контрольных.

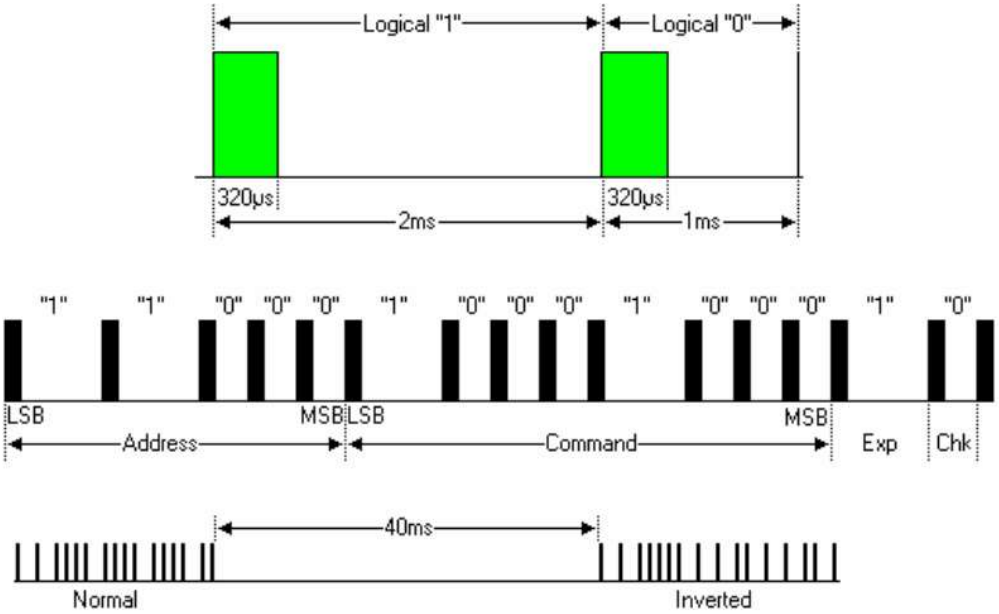


Рис. 7.12. Кодировка сигнала по протоколу SHARP

Протокол SONY (SIRC). Существуют три версии данного протокола, использующие соответственно 12, 15 и 20 информационных бит. Наиболее распространен12-битный протокол, который состоит из 5 бит адреса и 7 бит команды, передающихся методом ШИМ, и предшествующей им преамбуле (рис. 7.13).

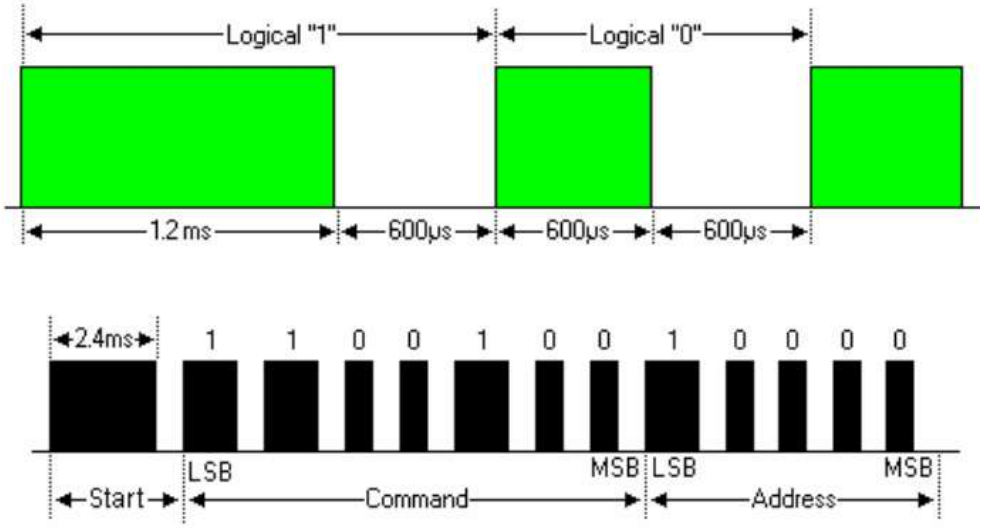


Рис. 7.13. Кодировка сигнала по протоколу SONY (SIRC)

При непрерывном нажатии на кнопку, передаются одинаковые посылки с периодом 50 мс.

Протокол RCA. Кодирование – ШМ, 4 бита адреса и8 бит команды передаются в каждой посылке дважды, в прямом и инверсном виде. Это позволяет иметь неизменную общую длительность посылки (рис. 7.14).

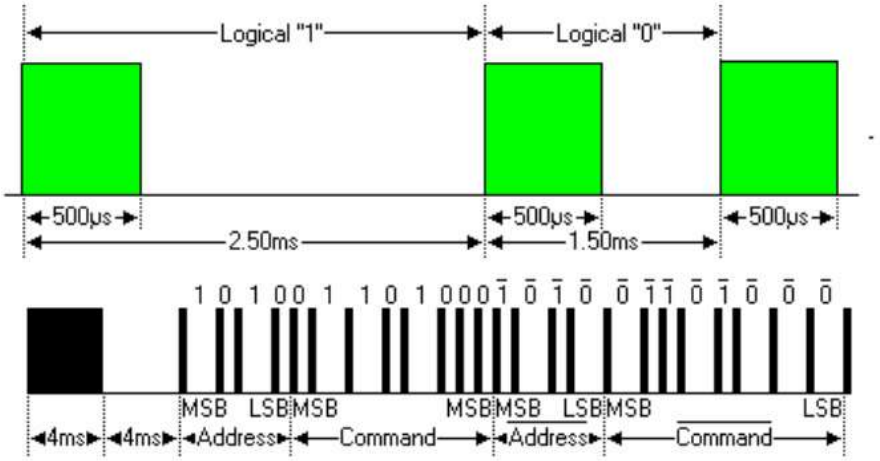


Рис. 7.14. Кодировка сигнала по протоколу RCA

При длительном нажатии – повторяется та же посылка с периодом 60 мс.

Протокол Panasonic (REC80, JAP). Очень «длинный» протокол – информационная длина 48 бит, из которых первые 32 бита являются постоянными для конкретного пульта, т.е. адресом. И только младшие 8 бит зависят от нажатой кнопки, т.е. являются командой (рис. 7.15).

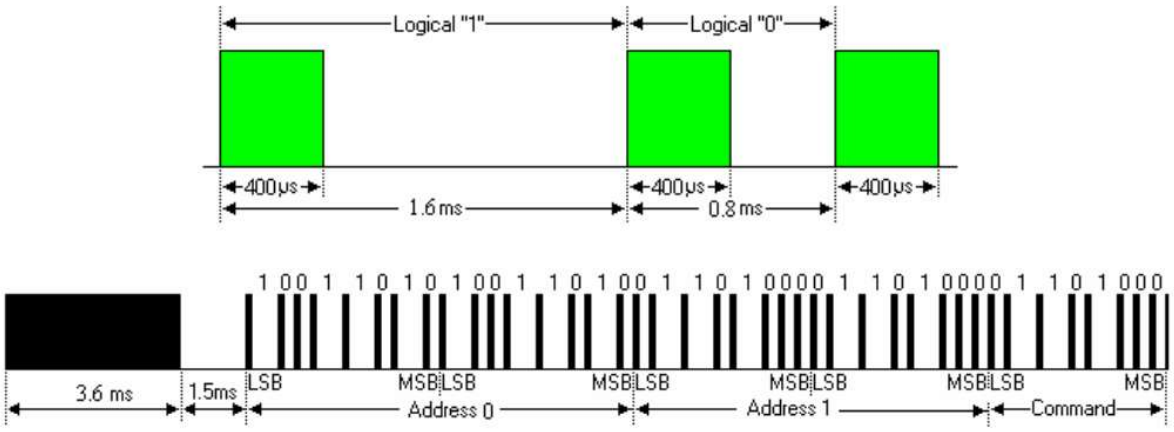


Рис. 7.15. Кодировка сигнала по протоколу Panasonic (REC80, JAP)

При длительном нажатии кнопки, повторяются одинаковые посылки через 70-80 мс.

Протокол Philips RECS-80. Хотя этот протокол и был разработан Филипсом, он не использует манчестерское кодирование (рис. 7.16).

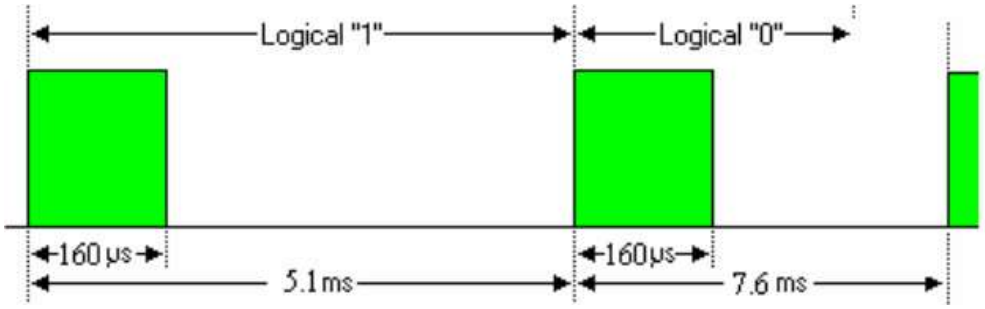


Рис. 7.16. Длительность импульсов логического 0 и 1

В целом, протокол довольно сложный, и имеет «обычный» и «расширенный» варианты. В обычном варианте передаются стартовый бит, «toggle bit», 3 бита адреса и 6 бит команды. Расширенный режим отличается наличием двух стартовых бит и четырех бит адреса (рис. 7.17).

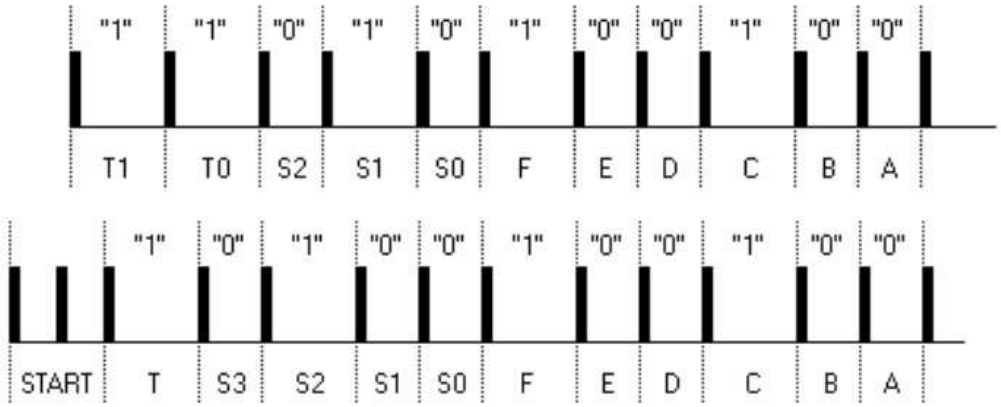


Рис. 7.17. Кодировка сигнала по протоколу Philips RECS-80

Филипс, как всегда, отличился оригинальностью, и ввел почти во все свои протоколы т.н. «toggle bit». Этот бит меняется каждый раз, когда кнопка была отпущена и снова нажата. При непрерывном нажатии кнопки– постоянно передается один и тот же код. Это позволяет легко отличить длительное нажатие от подряд идущих

кратковременных. Даже если была нажата одна и та же кнопка. Способ безусловно неплохой, но как видим – все остальные прекрасно обходятся и без этого.

Протокол Philips RC5. Один из самых распространенных протоколов в мире, Филипс RC5 использует уже знакомое нам бифазное (манчестерское) кодирование информационных битов (рис. 7.18).

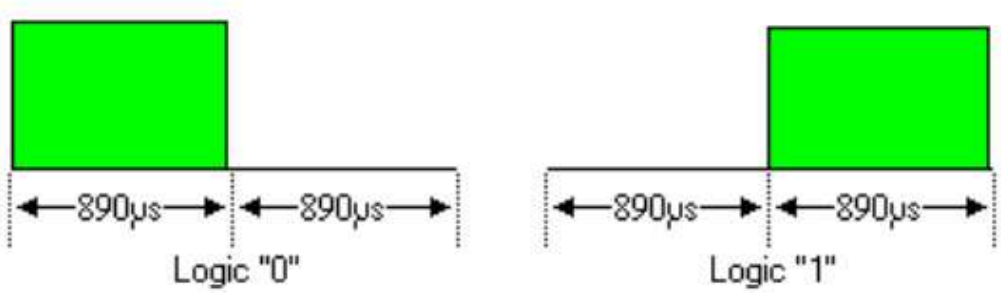


Рис. 7.18. Длительность импульсов логических 0 и 1

Протокол содержит 5 бит адреса и 6 бит команды, предваряется стартовым битом и битом «toggle» (рис. 7.19).

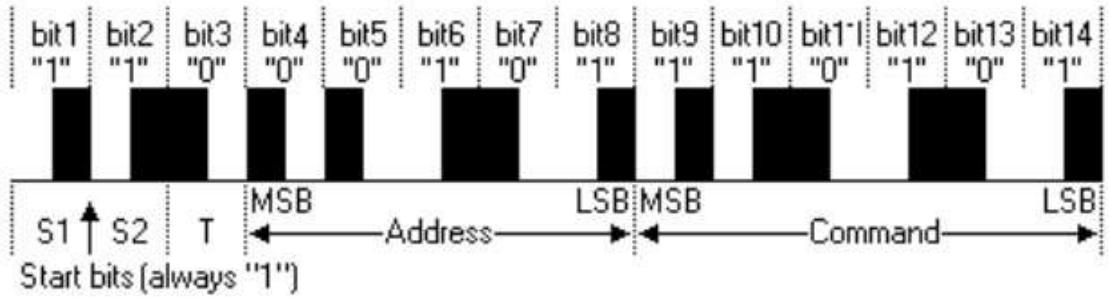


Рис. 7.19. Кодировка сигнала по протоколу Philips RC5

При длительном нажатии передаются одинаковые посылки, при отпускании кнопки – в следующей посылке инвертируется и бит toggle.

Протокол Philips RC6. Немного более сложный чем RC5 – филипповский протокол RC6. Он использует преамбулу из длинного импульса и короткой паузы, за которой следуют информационные биты. Которые в свою очередь, могут быть одинарной или двойной длительности (рис. 7.20).

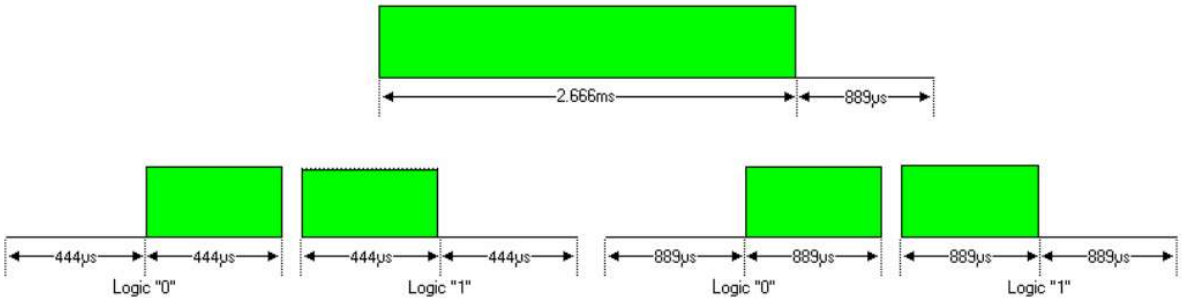


Рис. 7.20. Длительность импульсов логических 0 и 1

Каждая посылка состоит из четырех частей – заголовка, адреса, команды паузы. Заголовок начинается с преамбулы, за которой следует стартовый бит и три бита режима работы (все биты кроме преамбулы, перелаются одинарной длительностью, т.е. 444 мкс.). Завершается преамбула битом «toggle», который передается удвоенной длительности (890 мкс.). Далее следую восемь бит адреса и восемь бит команды. Завершается каждая посылка обязательной паузой, длительностью 2.7 мс (рис. 7.21).

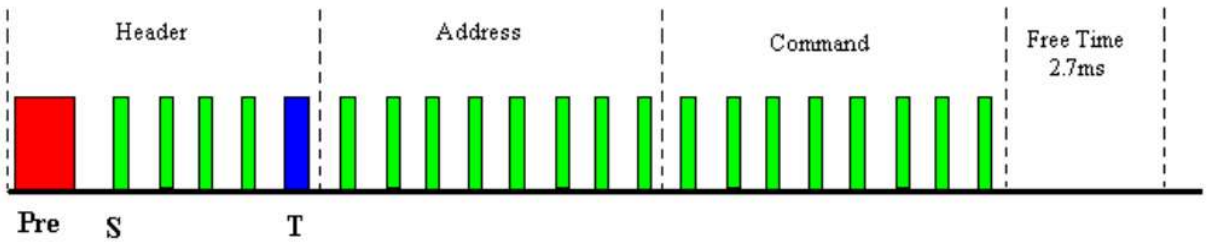


Рис. 7.21. Кодировка сигнала по протоколу Philips RC6

Программное декодирование. Существуют много способов обработки сигнала с фотоприемника микроконтроллером или компьютером. Проще всего декодировать протоколы с кодированием длительностью – нужно измерить длительность единицы и нуля, после чего можно однозначно сделать вывод о передаваемом бите – единица это или ноль. Аналогичным способом декодируется и стартовый бит(преамбула). В отличие от манчестерского кодирования, нам точно известно начало каждого передаваемого бита, и известно, что должно быть изменение сигнала где-то между ними.

При манчестерском кодировании – обязательно только изменение состояния в середине бита, а вот на границе битов перехода может и не быть – если передаются разные биты.

Существует много алгоритмов декодирования манчестерского код, например, алгоритм со «стейтмашиной»: http://www.clearwater.com.au/rc5/, можно и другими способами – суть от этого сильно не меняется.

Мы применим следующий алгоритм: считать длительности нуля и единицы, но запоминать при этом что было в прошлый раз, середина бита или граница между битами. И смотреть какой был предыдущий импульс – длинный (2T) или короткий (T). Мы совершенно точно знаем, что переход из 0 в 1 соответствует передаваемой единице, а переход из 1 в 0 – передаваемому нулю. Но это если он был в середине бита. Если же это переход на границе битов – его следует просто проигнорировать. Для определения, где же был этот переход достаточно посмотреть на длительность импульса, а также помнить, что детектировалось в последний раз – середина или граница. Если импульс был длинный – мы однозначно находимся посередине бита (тут можно проверить – перед этим могла быть тоже только середина, иначе это ошибка!), и направление перехода указывает на передаваемый бит. Если импульс был короткий, а перед этим была середина – значит мы однозначно на границе битов и игнорируем переход (но запоминаем что он был на границе!). Если же импульс был короткий, а перед этим была граница – значит мы на середине.

Для удобства реализации любых алгоритмов декодирования на микроконтроллерах (MCU), сигнал с фотоприемника нужно завести на вход внешнего прерывания и использовать таймер, присутствующим с большинства MCU, для подсчета времени между фронтами.

Для среды Arduino IDE есть специальная библиотека IRremote в которой реализованы алгоритмы распознавания кодировок сигналов. Далее ознакомимся с библиотечными примерами (IRrecvDumpV2 и IRsendDemo), которые приведены ниже. Запрограммируйте микропроцессор примером из библиотеки IRrecvDumpV2 приведенным ниже и нажимая кнопки пульта (рис. 7.22) запишите коды кнопок.

Рис. 7.22. Пульт дистанционного управления

Пример:

```
#include <IRremote.h> //Подключаем библиотеку IRremote
int recvPin = 19; //Фотоприемник подключен
IRrecv irrecv(recvPin); //к выводу 11
void setup ()
{
Serial.begin(9600); // Обмен данными с компьютером на скорости 9600bps
irrecv.enableIRIn(); // Запускаем сканирование сигнала
}
void ircode (decode_results *results)
{
// Определение типа кодировки и кода
if (results->decode_type == PANASONIC) {//Если результат декодирован
Serial.print(results->address, HEX); //Печатаем адрес в шеснадцатиричном виде
Serial.print(":");
}
Serial.print(results->value, HEX); // Печатаем код в шеснадцатиричном виде
}
void encoding (decode_results *results) // Отображение типов кодировок
{
switch (results->decode_type) {
default:
```

```
case UNKNOWN: Serial.print("UNKNOWN"); break;
case NEC: Serial.print("NEC"); break;
case SONY: Serial.print("SONY"); break;
case RC5: Serial.print("RC5"); break;
case RC6: Serial.print("RC6"); break;
case DISH: Serial.print("DISH"); break;
case SHARP: Serial.print("SHARP"); break;
case JVC: Serial.print("JVC"); break;
case SANYO:   Serial.print("SANYO"); break;
case MITSUBISHI: Serial.print("MITSUBISHI"); break;
case SAMSUNG: Serial.print("SAMSUNG"); break;
case LG: Serial.print("LG"); break;
case WHYNTER: Serial.print("WHYNTER"); break;
case AIWA_RC_T501: Serial.print("AIWA_RC_T501"); break;
case PANASONIC: Serial.print("PANASONIC"); break;
case DENON: Serial.print("Denon"); break;
}
}
void dumpInfo (decode_results *results)
{
if (results->overflow) {
Serial.println("IR code too long. Edit IRremoteInt.h and increase RAW-
BUF");
return;
}
//Отображение стандарта кодировки
Serial.print("Encoding: ");
encoding(results);
Serial.println("");

// Отображение кода и его длинны
Serial.print("Code: ");
ircode(results);
Serial.print(" (");
Serial.print(results->bits, DEC);
Serial.println(" bits)");
}
void dumpRaw (decode_results *results) // Выгружаем структуру деко-
дируемого результата.
{
// Отображаем данные в формате временных задержек
Serial.print("Timing [");
```

Продолжение примера

```
Serial.print(results->rawlen-1, DEC);
Serial.println("]: ");
for (int i = 1; i < results->rawlen i++) {
unsigned long x = results->rawbuf[i] * USECPERTICK;
if (! (i & 1)) {
Serial.print("-");
if (x < 1000) Serial.print(" ");
if (x < 100)   Serial.print(" ");
Serial.print(x, DEC);
} else {
Serial.print(" ");
Serial.print("+");
if (x < 1000) Serial.print(" ");
if (x < 100)   Serial.print(" ");
Serial.print(x, DEC);
if (i < results->rawlen-1) Serial.print(", ");
}
if (! (i % 8)) Serial.println("");
}
Serial.println("");
}
void dumpCode (decode_results *results) // Dump out the decode_results structure.
{// Вывод кода
Serial.print("unsigned int "); // Вывод типа
Serial.print("rawData ["); // структуры
Serial.print(results->rawlen - 1, DEC); // и размера кода
Serial.print("] = {");
for (int i = 1; i < results->rawlen; i++) {//Дамп данных
Serial.print(results->rawbuf[i] * USECPERTICK, DEC);
if (i < results->rawlen-1) Serial.print(",");
if (! (i & 1)) Serial.print(" ");
}
Serial.print("};"); // окончание вывода данных
Serial.print(" // ");
encoding(results);
Serial.print(" ");
ircode(results);
Serial.println(""); // Вывод новой строки
 if (results->decode_type! = UNKNOWN) {// в случае если тип данных не определен
if (results->decode_type == PANASONIC) {// Некоторые протоколы не имеют адреса
```

Продолжение примера

```
Serial.print("unsigned int addr = 0x");
Serial.print(results->address, HEX); //Выводится сообщение об отсут-
ствии адреса
Serial.println(";");
}
Serial.print("unsigned int data = 0x"); // Однако все протоколы содер-
жат определенные данные
Serial.print(results->value, HEX); //которые будут выведены в шестна-
дцатеричном виде
Serial.println(";");
}
}
void loop ()
{
decode_results results;

if (irrecv.decode(&results)) {  // В случае считывания IR кода
dumpInfo(&results); // выводим результат о типе кодировки, а также
dumpRaw(&results); // в формате длительностей импульсов
dumpCode(&results); // и в цифровом формате,
Serial.println(""); // выводим пустую строку и
irrecv.resume(); // подготавливаемся к следующему считыванию
}
}
```

Для отправки сигнала загрузите пример библиотеки IRsendDemo приведенный ниже

Пример:

```
#include <IRremote.h> // Подключаем библиотеку IRremote
IRsend irsend;
void setup ()
{
}
void loop () {
for (int i = 0; i < 3; i++) {
irsend.sendSony (0xa90, 12);
// Отправка кода длинной 12 бит кодировкой Sony
delay (40);
}
delay (5000); // пауза 5 секунд
}
```

Задание для самостоятельной работы

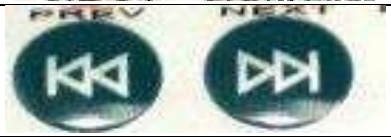
1. Запрограммируйте 19 (A5)-й вывод отладочной платы Arduino UNO. для приема инфракрасного сигнала с пульта и по нажатию кнопки пульта включайте и выключайте светодиод на выводе (согласно таблице 7.3).

2. Запрограммируйте 19 (A5)-й вывод отладочной платы Arduino UNO. для приема инфракрасного сигнала с пульта и по нажатию кнопки пульта уменьшайте и увеличивайте яркость светодиода на выводе (согласно таблице 7.3).

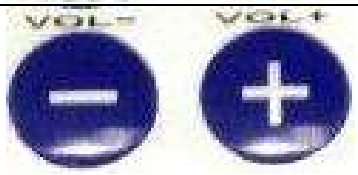
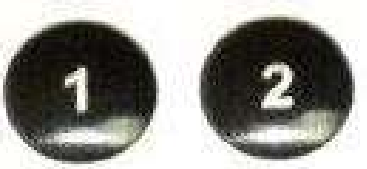
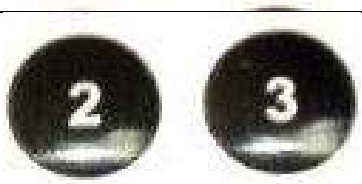
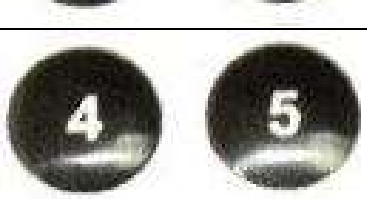
3. Запрограммируйте 19 (A5)-й вывод отладочной платы Arduino UNO. для приема инфракрасного сигнала с пульта и по нажатию кнопки изменяйте цвета 3-х цветного светодиода.

4. Используя две отладочные платы запрограммируйте на одной Arduino UNO 19 (A5)-й вывод для приема сигнала, а на другой Arduino UNO 3-й вывод для передачи сигнала включения и выключения светодиода на выводе (согласно таблица 7.3).

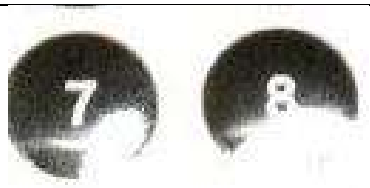
Таблица 7.3. Варианты заданий

№ варианта	Кнопки пульта	Светодиод подключен к выводу	3-х цветный светодиод подключен к выводу
1		3	3,5,6
2		5	9,10,11
3		6	3,5,6

Продолжение таблицы 7.3.

4		7	9,10,11
5		9	3,5,6
6		10	9,10,11
7		11	3,5,6
8		12	9,10,11
9		13	3,5,6
10		3	9,10,11
11		5	3,5,6
12		6	9,10,11

Продолжение таблицы 7.3.

13		7	3,5,6
14		9	9,10,11
15		10	3,5,6
16		11	9,10,11
17		12	3,5,6
18		13	9,10,11

Продолжение таблицы 7.3.

19		3	3,5,6
20		5	9,10,11
21		6	3,5,6
22		7	9,10,11
23		9	3,5,6
24		10	9,10,11

Продолжение таблицы 7.3.

25		11	3,5,6
26		12	9,10,11
27		13	3,5,6
28		3	9,10,11
29		5	3,5,6
30		6	9,10,11

Контрольные вопросы

1. Где чаще всего используется инфракрасное дистанционное управление?
2. Что такое «Mark» и «Space»?
3. Что включается в битовое кодирование и что такое логический состав?
4. Дайте характеристику кодировки NEC.
5. Основное отличие протокола JVC от NEC?

Лабораторная работа № 8.
Считывание бесконтактных карт радио-меток RFID RC522 и ключей IButton

В этой лабораторной узнаем о считывании информации с носителей контактным и безконтактным способами, о реализации этих способов на микроконтроллерах *Atmega328* в среде програмирования Arduino IDE.

В итоге, мы напишем программу для микроконтроллера, которая будет считывать коды с ключей и меток RFID. При считывании определенного кода будем включать и выключать светодиоды на отладочной плате.

Цель работы: приобрести практические навыки по управлению устройствами при помощи контактных ключей и бесконтактных меток RFID.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программы указанному в примере.
4. Выполнить задание для самостоятельной работы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.

Теоретические сведения

RFID-метки и работа с ними. Многим, конечно же, знакомы карточки и брелоки, которые нужно подносить к считывателю, а они при этом пропускают на работу или дают проехать в метро. Такие

брелоки и карточки используют технологию под названием RFID – Radio Frequency ID entification. Большую часть карточки или брелока, которые далее мы будем называть RFID-метками, занимает антенна. Также в метке содержится очень маленький чип, реализующий всю логику. В считывателе также есть антенна, притом регулярно передающая сигнал и, следовательно, создающая электромагнитное поле. При поднесении метки в это поле на ее антенне индуцируется ток, который и питает метку. Теперь метка и считыватель могут пообщаться друг с другом при помощи радиосигнала, используя какой-то свой протокол и модуляцию сигнала.

Стоит отметить, что это описание касалось пассивных меток (рис. 8.1). Бывают и активные метки, имеющие собственный источник питания, а также полупассивные. Что же касается радиосигнала, в RFID сигнал типично передается на частоте 125 КГц или 13.56 МГц. Существует множество стандартов передачи сигнала и их реализаций. Довольно распространенным является стандарт ISO/IEC 14443 и его реализация MIFARE от компании NXP Semiconductors. Еще одним известным стандартом является NFC, основанный на ISO 14443. Одна из его реализаций называется NTAG, также от NXP Semiconductors. Стоит отметить, что в общем случае реализации одного стандарта от разных производителей могут быть не вполне совместимы друг с другом и содержать расширения, которых нет в самом стандарте.

Рис. 8.1. RFID модуль RC522, карта, брелок

Для Arduino существует несколько модулей для работы с RFID. Пожалуй, самым дешевым и в то же время самым популярным, является модуль под названием RC522 (рис. 8.1) на базе чипа MFRC522 от NXP. Чип MFRC522 поддерживает технологии MIFARE и NTAG, радиосигнал передается на частоте 13.56 МГц. С микроконтроллером чип общается по SPI. Однако прямо по SPI ходить в чип нам не придется, так как для работы с модулем существует готовая библиотека MFRC522 или rfiid-master. Ее можно установить прямо из Arduino IDE.

Основные характеристики RC-522:

– основан на микросхеме MFRC522;

– напряжение питания: 3.3 V;

– потребляемый ток:13-26 mA;

– потребляемый ток в ждущем режиме: 10-13 ma;

– потребляемый ток в спящем режиме: менее 80 мкА;

– рабочая частота: 13.56 MHz;

– дальность считывания: 0 ~ 60 мм;

– интерфейс: SPI, максимальная скорость передачи 10 МБит/с;

– размер: 40 мм х 60 мм;

– чтение и запись RFID-меток.

Подключение модуля к Arduino Uno осуществляется так:

– пины 3.3 V и GND Arduino (рис. 8.2) подключаем к аналогичным пинам модуля;

– пины с 9 по 13 Arduino (рис. 8.2) подключаем к пинам RST, SDA, MOSI, MISO и SCK модуля соответственно.

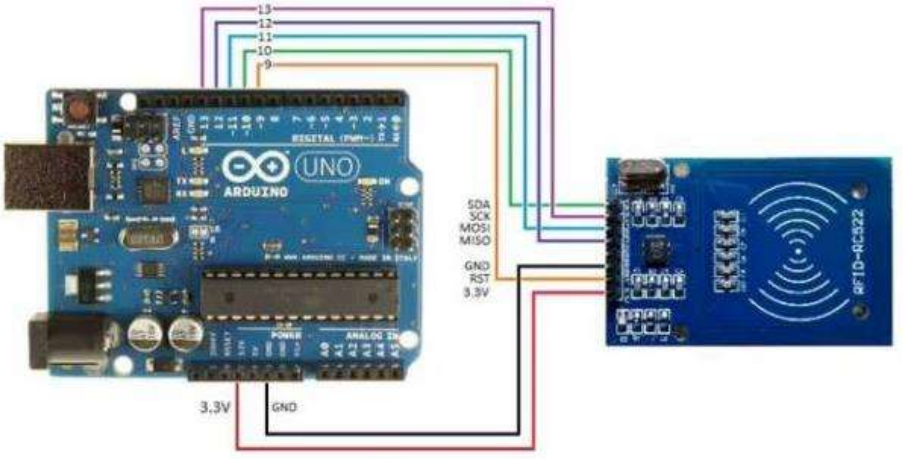


Рис. 8.2. Подключение модуля MFRC522 к Arduino Uno R3

Для проверки, в качестве программы для Arduino, воспользуемся примером из библиотеки rfiid-master. Пример называется «cardDisplay» и показывает версию прошивки контроллера MFRC522. Он может определять только известные версии, но даже если тест прошёл, это не означает что весь модуль полностью работоспособен – могут быть физические причины его неработоспособности, такие как повреждённая обвязка, сломанная антенна или нерабочая радио-метка. Далее приводится текст программы данного примера.

Пример:

```
#include <SPI.h>//Подключение библиотеки для передачи данных по протоколу SPI
#include <RFID.h>//Подключение библиотеки для считывания данных с радиочастотной метки
#define SS_PIN 10//Вывод 10 задействован для обмена данными по протоколу SPI
#define RST_PIN 9//Вывод 9 задействован для обмена данными по протоколу SPI
RFID rfid (SS_PIN, RST_PIN); //Подключение выводов 9 и 10 для считывания данных с радиочастотной метки
//Объявление переменных:
int serNum0;//Для хранения 0-х считываемых данных с радиочастотной метки
int serNum1;//Для хранения 1-х считываемых данных с радиочастотной метки
int serNum2;//Для хранения 2-х считываемых данных с радиочастотной метки
int serNum3;//Для хранения 3-х считываемых данных с радиочастотной метки
int serNum4;//Для хранения 4-х считываемых данных с радиочастотной метки
void setup ()
{
Serial.begin (9600); //начало обмена данными по протоколу UART
SPI.begin(); //начало обмена данными по протоколу SPI
rfid.init(); //Инициализация RFID модуля
}
void loop ()
{
if (rfid.isCard()) {//Если обнаружена радиочастотная метка
if (rfid.readCardSerial()) {// и считанные данные отличаются от предыдущих
if (rfid.serNum[0] != serNum0
&& rfid.serNum[1] != serNum1
&& rfid.serNum[2] != serNum2
```

Продолжение примера

```
&& rfid.serNum[3] != serNum3
&& rfid.serNum[4] != serNum4
) {
/* то происходит перезапись данных в переменных serNum0-serNum4 */
Serial.println(" ");
Serial.println("Card found");
serNum0 = rfid.serNum[0];
serNum1 = rfid.serNum[1];
serNum2 = rfid.serNum[2];
serNum3 = rfid.serNum[3];
serNum4 = rfid.serNum[4];
//Выводится в монитор порта номер карточки в десятичном виде
Serial.println("Cardnumber:");
Serial.print("Dec: ");
Serial.print(rfid.serNum[0],DEC);
Serial.print(", ");
Serial.print(rfid.serNum[1],DEC);
Serial.print(", ");
Serial.print(rfid.serNum[2],DEC);
Serial.print(", ");
Serial.print(rfid.serNum[3],DEC);
Serial.print(", ");
Serial.print(rfid.serNum[4],DEC);
Serial.println(" ");
//Выводится в монитор порта номер карточки в шеснадцатиричном виде
Serial.print("Hex: ");
Serial.print(rfid.serNum[0],HEX);
Serial.print(", ");
Serial.print(rfid.serNum[1],HEX);
Serial.print(", ");
Serial.print(rfid.serNum[2],HEX);
Serial.print(", ");
Serial.print(rfid.serNum[3],HEX);
Serial.print(", ");
Serial.print(rfid.serNum[4],HEX);
Serial.println(" ");
} else {
/* Выводятся точки в монитор порта. */
Serial.print(".");
}
}
}
rfid.halt();
}
```

Протокол 1-Wire и ключ iButton. 1-Wire это однопроводной интерфейс, разработан фирмой Dallas Semiconductor (ныне MAXIM) в конце 90-х годов. Этот интерфейс интересен тем, что для двустороннего обмена требуется всего одна линия! (отсюда и название). Правда, ещё требуется общий провод (земля).

Причём, на эту одну линию можно повесить несколько устройств, а ассортимент таких устройств очень широк (от датчиков температуры до широко распространённых ключей-таблеток iButton).

Кроме того – протокол очень прост и легко реализуется на МК программное. Ниже (рис. 8.3) представлена блок-схема аппаратной реализации 1-Wire.

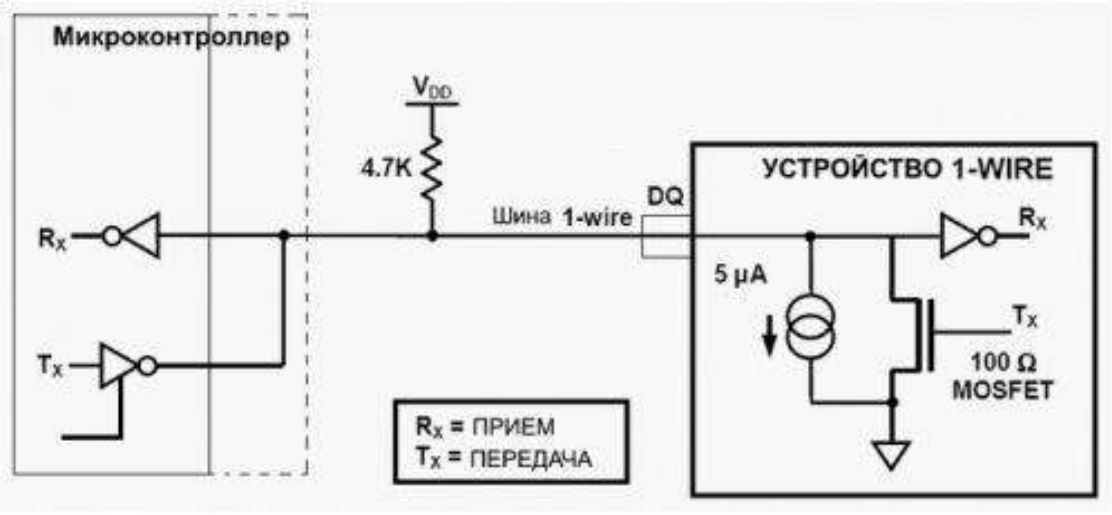


Рис. 8.3. Блок-схема аппаратной реализации 1-Wire

Вывод DQ устройства представляет собой вход КМОП-логического элемента, который может быть зашунтирован (замкнут на общий провод) полевым транзистором. Сопротивление канала этого транзистора в открытом состоянии – около 100 Ом. Когда транзистор заперт – имеется небольшой ток утечки (примерно 5 мкА) на общий провод.

Обратите внимание, что шина 1-Wire должна быть подтянута отдельным резистором к напряжению питания (может быть от 3 V до 5 V – нужно уточнять по характеристикам подключаемого устройства).

Сопротивление этого резистора 4.7k, однако, это значение

подходит только для достаточно коротких линий.

Если шина используется для подключения устройств на большее расстояние, то сопротивление подтягивающего резистора необходимо уменьшить (сопротивление зависит от величины максимального втекающего тока линии DQ конкретного устройства 1-Wire).

Примечательный момент – некоторые устройства 1-Wire могут использовать т.н. «паразитное питание»/фантомное питание (Parasite power) – т.е. питание устройства осуществляется от линии данных.

Электропитание осуществляется за счёт заряда встроенного конденсатора, который заряжается во время наличия высокого уровня напряжения на линии данных.

Опять же, следует учитывать, что связь с устройствами, использующими паразитное питание возможно только на коротких линиях. На длинных линиях возможны непонятные побочные эффекты.

Основные положения:

– передача информации возможна только выдачей низкого уровня в линию, т.е. замыканием ее на общий провод. В высокий логический уровень линия вернется сама, из-за наличия подтягивающего резистора (теперь становится понятно, что *наличие внешнего подтягивающего резистора – обязательное условие работы 1-Wire);*

– обмен происходит по инициативе ведущего устройства (обычно – микроконтроллера);

– обмен информацией начинается с подачи импульса сброса (*RESET* pulse) на линию;

– 1-wire устройства предусматривают «горячее» подключение;

– при подключении, 1-wire устройство выдаёт в линию DQ импульс присутствия (*PRESENCE* pulse). Этот же импульс выдаётся при обнаружении сигнала RESET;

– обмен информации ведётся так называемыми тайм-слотами – один слот содержит один бит информации;

– данные передаются побайтно – бит за битом, начиная с младшего байта. Достоверность данных (проверка отсутствия искажений) гарантируется путем подсчета циклической контрольной суммы (CRC).

Алгоритм подсчета CRC должен быть одинаковым как для

МК, так и для любого устройства 1-Wire. Он стандартизирован и описан в документации (рис. 8.4).

Рис. 8.4. Временная диаграмма обмена данными микроконтроллера по протоколу 1-Wire

Как видно по диаграмме – МК формирует импульс RESET, переводя в низкий логический уровень шину 1-Wire и удерживая её не менее 480 микросекунд. Затем МК «отпускает» шину и напряжение возвращается к высокому логическому уровню (время зависит от ёмкости линии и сопротивления подтягивающего резистора). Протокол 1-Wire ограничивает это время диапазоном от 15 до 60 микросекунд, что и влияет на выбор подтягивающего резистора (на время возврата линии к высокому уровню большее влияние оказывает ёмкость лини, но, чаще всего, мы изменить её не можем). Обнаружив импульс RESET, ведомое устройство формирует ответный импульс PRESENCE. Для этого устройство прижимает линию DQ к земле и удерживает от 60 до 240 микросекунд.

Затем устройство так же «отпускает» шину. После этого устройству еще дается время для завершения внутренних процедур инициализации, таким образом, МК должен приступить к любому обмену с устройством не ранее, чем через 480 микросекунд после завершения импульса RESET. Таким образом процедура инициализации, с которой начинается обмен данными между устройствами, длится минимум 960 микросекунд (рис. 8.5).

Теперь рассмотрим процедуры обмена битами информации, которые осуществляются определенными тайм-слотами (определенная, жестко лимитированная по времени последовательность смены уровней сигнала в линии 1-Wire).

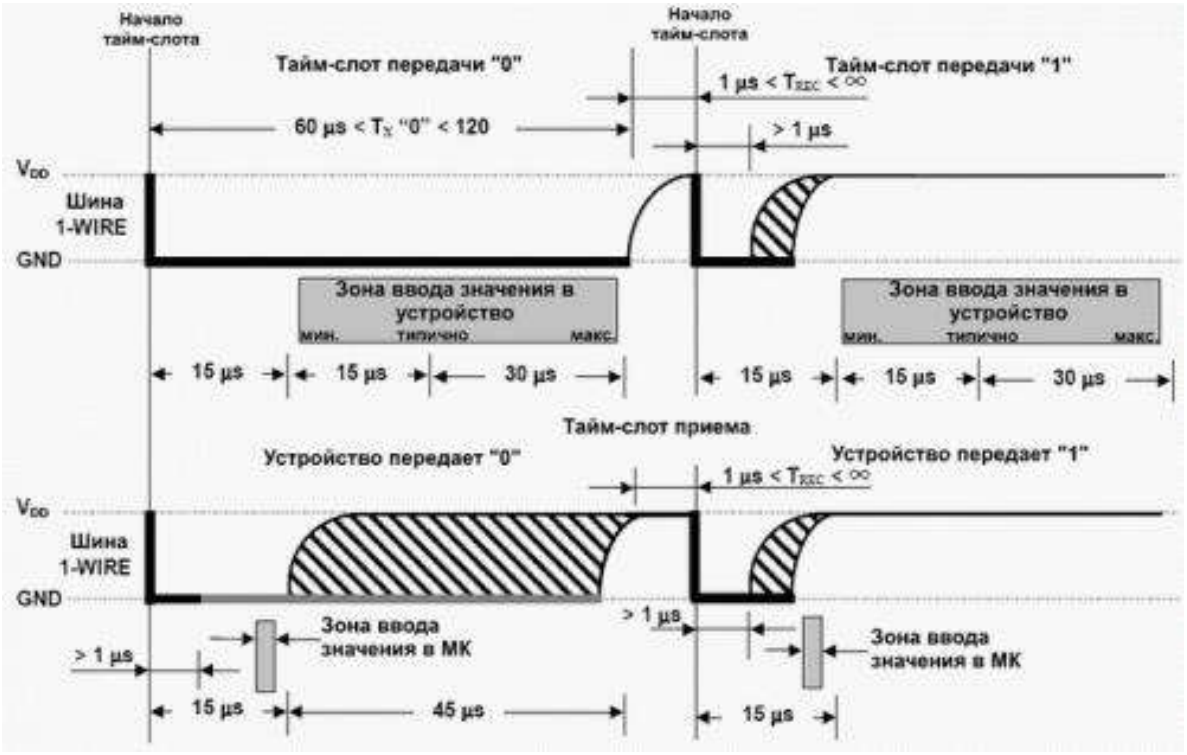


Рис. 8.5. Временная диаграмма обмена битами информации по протоколу 1-Wire

Различают 4 типа тайм-слотов:

1. передача «1» от МК;
2. передача «0» от МК;
3. прием «1» от устройства;
4. прием «0» от устройства.

Тайм-слот всегда начинает МК, прижимая шину к земле.

Длительность тайм-слота находится в пределах от 60 до 120 микросекунд.

Между тайм-слотами всегда должен быть интервал не менее 1 микросекунды (определяется параметрами ведомого устройства).

Тайм-слоты передачи отличаются от тайм-слотов приема поведением МК:

– при передаче МК только формирует сигналы,

– при приеме МК еще и опрашивает уровень сигнала в линии 1-Wire.

Тайм-слот передачи «0» заключается просто в прижимании шины 1-Wire к земле в течение всей длительности тайм-слота.

Передача «1» осуществляется путем «отпускания» шины 1-Wire со стороны МК не ранее чем через 1 микросекунду после начала тайм-слота, но не позже, чем через 15 микросекунд.

Ведомое устройство опрашивает уровень в шине 1-Wire в течение временного интервала (показанного в виде серого прямоугольника), т.е. начиная с 15-й микросекунды от начала тайм-слота и заканчивая 60-й микросекундой от начала (для большинства устройств – около 30-й микросекунды от начала тайм-слота).

Заштрихованная область – это область «нарастания» уровня в шине 1-Wire, которая зависит от емкости линии и сопротивления подтягивающего резистора.

Тайм-слоты приема информации отличаются тем, что МК формирует только начало тайм-слота (так же, как при передаче «1»), а затем управление уровнем шины 1-Wire берет на себя устройство, а МК осуществляет ввод этого уровня так же в определенной зоне временных интервалов.

Зона эта, как видно из рисунка, довольно мала. Т.к. заштрихованная область – область неопределенности, поэтому для ввода, микроконтроллеру остается даже не промежуток, а скорее конкретный момент, когда он должен ввести уровень сигнала из линии. Этот момент времени – 14-я или 15-я микросекунда от начала тайм-слота.

Резюмируем:

– МК начинает тайм-слот, прижимая шину 1-Wire к логическому «0» в течение 1 микросекунды.

– Последующий уровень зависит от типа тайм слота: для приема и передачи «1» уровень должен стать высоким, а для передачи «0» – оставаться низким вплоть до конца тайм-слота, т.е. от 60 до 120 микросекунд.

– принимая данные, МК должен считать уровень в шины 1-Wire в промежутке от 13-й до 15-й микросекунде тайм-слота.

– МК должен обеспечить интервал между тайм-слотами не менее 1 микросекунды (лучше – больше, максимальное значение не ограничено).

Для достижения нужных временных интервалов нужно следовать простым правилам:

– все сигналы, которые должен формировать МК, следует формировать по принципу необходимого минимума длительности (т.е. немного больше, чем указанная минимальная длительность);

– от устройства следует ожидать сигналов по принципу наихудшего (т.е. ориентироваться на самые худшие варианты временных параметров сигнала).

Теперь настало время разобраться с протоколом обмена информации.

Каждое устройство 1-Wire обладает уникальным идентификационным 64-битным номером, программируемым на этапе производства микросхемы (это относится не только к нашему ключу-таблетке, а ко всем устройствам 1-Wire).

Фирма-производитель гарантирует, что не найдется двух микросхем с одинаковым идентификационным номером.

Не трудно посчитать, что устройств одного типа может быть выпущено.

281 474 976 710 655 (десятичное представление 0xFFFFFFFFFFFF – 48 бит – 6 байт идентификационного номера) – т.е. 281 биллион – довольно много.

Предположим, что на шине 1-Wire имеется более одного устройства.

В этом случае перед МК встают 2 проблемы: определение количества имеющихся устройств и выбор (адресация) одного конкретного из них для обмена данными.

Номера некоторых устройств наносятся прямо на корпус микросхем (например, для наших ключей-таблеток – iButton), а номера других можно определить при помощи специальных программ или устройств.

Итак, предположим, что мы знаем номера всех устройств 1-Wire на шине.

Алгоритм работы с ними следующий:

1. МК посылает, импульс RESET, и все имеющиеся устройства выдают PRESENCE.

2. МК посылает в шину команду, которую принимают все устройства. Определено несколько общих команд (таблица 8.1) для всех типов 1-Wire-устройств, а также есть команды, уникальные для отдельных типов устройств.

Таблица 8.1. Общие команды для всех типов 1-Wire-устройств

Команда	Значение байта	Описание
SEARCH ROM	0xF0	Поиск адресов – используется при универсальном алгоритме определения количества и адресов подключенных устройств
READ ROM	0x33	Чтение адреса устройства – используется для определения адреса единственного устройства на шине
MATCH ROM	0x55	Выбор адреса – используется для обращения к конкретному адресу устройства из многих подключенных
SKIP ROM	0xCC	Игнорировать адрес – используется для обращения к единственному устройству на шине, при этом адрес устройства игнорируется (можно обращаться к неизвестному устройству)

3. После того, как МК выдаст команду READ ROM, от устройства поступит 8 байт его собственного уникального адреса – МК должен их принять.

Любая процедура обмена данными с устройством должна быть завершена полностью либо прервана посылкой сигнала RESET.

4. Если отправлена команда MATCH ROM, то после нее МК должен передать 8 байт адреса конкретного устройства, с которым будет осуществляться последующий обмен данными.

5. Приняв эту команду, каждое устройство сравнивает передаваемый адрес со своим собственным. Все устройства, адрес которых не совпал, прекращают анализ и выдачу сигналов в линии 1-Wire, а опознавшее адрес устройство продолжает работу.

Теперь все данные, передаваемые МК будут попадать только к этому «адресованному» устройству.

Если устройство одно на шине – можно ускорить процесс взаимодействия с ним при помощи команды *SKIP ROM*. Поучив эту команду, устройство сразу считает адрес совпавшим, хотя никакого адреса за этой командой не следует.

Некоторые процедуры не требуют приема от устройства никаких данных, в этом случае команду *SKIP ROM* можно использовать для передачи какой-то информации сразу всем устройствам. Это

можно использовать, например, для одновременного запуска цикла измерения температуры несколькими датчиками-термостатами типа DS18S20.

Прием и передача байтов, как уже отмечалось, всегда начинается с младшего бита. Порядок следования байтов при передаче и приеме адреса устройства так же ведется от младшего к старшему.

Порядок передачи другой информации зависит от конкретного устройства.

Далее рассмотрим пример, в котором считывается идентификационный код ключа ibutton. Схема подключения ключа к отладочной плате приведена на рис. 8.6.

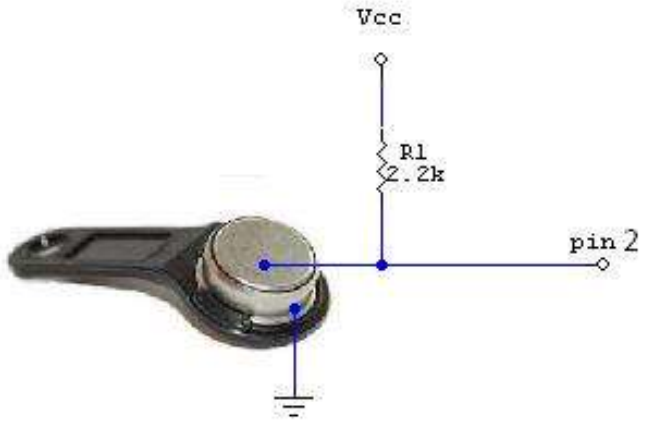


Рис. 8.6. Схема подключения ключа ibutton к отладочной плате Arduino Uno R3

После того как в отладочную плату загружен код, из приведенного ниже примера, и будет приложен ibutton ключ, в мониторе порта можно будет увидеть его идентификационный код.

Пример:

```
#include <OneWire.h>   // Подключение библиотеки OneWire
int IButtonPin = 2; // Пин iButton подключён к выходу 2
String IButtonNum, IButtonNumOld;
OneWire ds(IButtonPin); // сигнальный пин iButton подключён к выводу 2
long previousMillis = 0;
long interval = 1000; // задержка считывания ключа 1 секунда
void setup () {
pinMode (IButtonPin, INPUT);
Serial.begin(9600);
}
```

Продолжение примера

```
void loop () {
unsigned long currentMillis = millis ();
byte addr [8]; // объявление массива для хранения кода iButton
if (ds.search(addr)) { // Вывод кода iButton при его обнаружении
IButtonNum = String (addr [0], HEX) + " " + String (addr [1], HEX) + " "
+ String (addr [2],
HEX) + " " + String (addr [3], HEX) + " " + String (addr [4], HEX) + " " +
String (addr [5],
HEX) + " " + String (addr [6], HEX) + " " + String (addr [7], HEX);
if (String(IButtonNumOld) != String(IButtonNum)){IButtonNumOld = IBut-
tonNum;}
if (String(IButtonNumOld) == String(IButtonNum) && currentMillis - pre-
viousMillis > interval)
{previousMillis = currentMillis;
Serial.println(IButtonNum);}
ds. reset_search();
}
}
```

Задание для самостоятельной работы

1. Запрограммируйте 7-й вывод (красный светодиод) и 8-й вывод (зеленый светодиод) отладочной платы Arduino UNO, чтобы при считывании RFID метки с брелока или карточки кратковременно загорался зеленый светодиод, а в остальное время горел красный светодиод.

2. Запрограммируйте 7-й вывод (красный светодиод) и 8-й вывод (зеленый светодиод) отладочной платы Arduino UNO, чтобы при считывании кода с ключа ibutton кратковременно загорался зеленый светодиод, а в остальное время горел красный светодиод.

3. Запрограммируйте 7-й вывод (красный светодиод) и 8-й вывод (зеленый светодиод) отладочной платы Arduino UNO, чтобы при считывании RFID метки с брелока или карточки кратковременно загорался зеленый светодиод, а при считывании RFID метки с карточки кратковременно загорался красный светодиод.

4. Запрограммируйте 7-й вывод (красный светодиод) и 8-й вывод (зеленый светодиод) отладочной платы Arduino UNO, чтобы при считывании кода с ключа ibutton кратковременно загорался зеленый светодиод, а при считывании кода с любого постороннего ключа ibutton кратковременно загорался красный светодиод.

Контрольные вопросы

1. Опишите кратко технологию RFID.
2. Назовите основные характеристики RC-522.
3. Что такое 1-Wire?
4. Какие типы тайм-слотов Вы знаете?
5. Назовите общие команды для всех типов 1-Wire устройств.

Лабораторная работа № 9.
Управление устройством через Ethernet

В этой лабораторной узнаем о управлении устройством по сети интернет, о реализации этого способа управления на микроконтроллерах *Atmega328* в среде програмирования Arduino IDE.

В итоге, мы напишем программу для микроконтроллера, которая будет отправлять и принимать данные из сети и в зависимости от полученных данных на устройстве будут включатся или выключатся светодиоды, изменять свою яркость. Управление устройством производится с ВЕБ страницы в сети Ethernet.

Цель работы: приобрести практические навыки по управлению устройствами по сети.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Собрать схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программы указанному в примере.
4. Выполнить задание для самостоятельной работы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.

Теоретические сведения

Общие *сведения.* Плата расширения Ethernet Shield W5100 позволяет за считанные минуты подключить Arduino к глобальной сети Интернет. Для этого достаточно всего лишь подключить модуль к Arduino, подсоединить его к сети кабелем RJ-45 и выполнить

несколько простых действий. Для работы требуется основное устройство плата Arduino UNO R3 и Ethernet-контроллер:

– W5100 со встроенным буфером объемом 16 КБ;

– скорость соединения: 10/100 Мбит/с.

Взаимодействие с Arduino осуществляется через интерфейс SPI.

Описание. Плата расширения Arduino Ethernet построена на базе Ethernet-контроллера Wiznet W5100, который позволяет Arduino подключаться к сети Интернет. Wiznet W5100 поддерживает стек сетевых протоколов (IP) и позволяет работать как с TCP, так и с UDP-протоколами. При этом микросхема может обслуживать до четырех одновременно открытых сокет-соединений. Для написания программ, работающих с глобальной сетью через плату расширения, рекомендуется использовать библиотеку Ethernet. Для подключения платы расширения к Arduino предусмотрен специальный разъем, представляющий собой металлические выводы с одной стороны платы и гнезда – с другой стороны. Такая конструкция позволяет подключить к Arduino сразу несколько плат расширения, разместив их одну над другой.

Последняя версия платы расширения поддерживает стандартную распиновку 1.0, принятую в модели Arduino UNO R3.

Плата расширения Ethernet поддерживает технологию «Power over Ethernet» и имеет стандартный разъем RJ-45 со встроенной гальванической развязкой.

На плате предусмотрен разъем для подключения micro-SD карты памяти, что дает возможность хранения файлов и организации сетевого доступа к ним. Устройство совместимо с Arduino Uno и Mega (используется библиотека Ethernet). Для работы со встроенным microSD-кардридером служит библиотека SD. Для активизации кардридера с помощью этой библиотеки в качестве вывода SS следует указывать вывод 4. Самая первая версия платы расширения Ethernet содержала полноразмерный разъем для SD-карт, который в настоящее время не поддерживается.

В устройстве также реализована функция управления сбросом Ethernet-модуля W5100 при подаче питания. Необходимость в этой функции обусловлена тем, что предыдущие версии платы

расширения были несовместимыми с Arduino Mega, из-за чего приходилось вручную сбрасывать Ethernet-модуль после каждой подачи питания.

Текущая версия платы расширения поддерживает технологию Power over Ethernet (PoE) и может работать со специальным модулем, позволяющим получать энергию через Ethernet-кабель, который представляет собой обычную витую пару категории 5:

– модуль совместим со стандартом IEEE802.3af;

– низкий уровень выходных пульсаций и шума (100 мВ от пика до пика);

– диапазон входного напряжения от 36В до 57В;

– защита от перегрузок и коротких замыканий;

– выходное напряжение 9В;

– DC-DC преобразователь с высоким КПД: 75% при 50% нагрузке;

– изоляция между входом и выходом в 1500В.

Примечание: модуль Power over Ethernet не производится Arduino и является проприетарным аппаратным обеспечением. Для получения дополнительной информации, см. документацию.

Плата расширения поставляется без встроенного модуля PoE – он представляет собой отдельный компонент, подключаемый к плате.

Arduino взаимодействует с микросхемой W5100 и SD-картой памяти по одной SPI-шине (через разъем ICSP). На Duemilanove шина SPI занимает цифровые выводы 11, 12 и 13, а на Arduino Mega – 50, 51 и 52. На обеих платах в качестве выводов для активизации микросхемы W5100 или SD-карты памяти используются выводы 10 и 4 соответственно. Поэтому данные выводы не могут использоваться в качестве выводов общего назначения для выполнения каких-то других функций. Следует также помнить, что для корректной работы SPI-интерфейса аппаратный вывод SS Arduino Mega (53) должен быть всегда сконфигурирован как выход несмотря на то, что он не взаимодействует ни с W5100, ни с SD-картой памяти.

Примечание: поскольку оба устройства, W5100 и SD-карта памяти, подключены к одной SPI-шине, то в каждый момент времени активным может быть только одно из них. При использовании в

вашем проекте обеих устройств, распределение доступа к шине контролируется соответствующими библиотеками. В том случае, если одно из устройств в проекте не используется – необходимо явно деактивировать его. Для этого вывод, отвечающий за активизацию соответствующего устройства (4 – для SD-карты, 10 – для W5100), необходимо сконфигурировать как выход и подать на него высокий уровень сигнала.

В плате расширения (рис. 9.1) используется стандартный сетевой разъем RJ45.

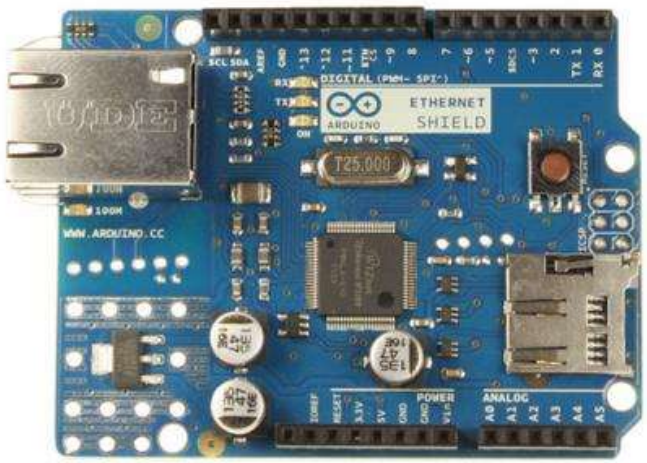


Рис. 9.1. Плата расширения Arduino Ethernet R3a – вид спереди

Кнопка сброса, расположенная на плате, осуществляет сброс Arduino вместе с контроллером W5100. Также на плате расширения расположено несколько светодиодов-индикаторов:

– PWR: показывает наличие питания основного устройства и платы расширения;

– LINK: светится, если есть соединение с сетью; мигает во время передачи или получения данных;

– FULLD: светится, если сетевое соединение поддерживает полнодуплексный режим работы;

– 100M: светится, если сетевое соединение относится к классу 100 Мб/с (в отличие от сетей 10 Мб/с);

– RX: мигает в процессе получении данных;

– TX: мигает в процессе отправки данных;

– COLL: информирует об обнаружении сетевых коллизий.

Плата расширения Arduino Ethernet может генерировать сигналы прерываний, которые позволяют уведомлять Arduino о различных событиях, возникающих в модуле W5100. Для этого на плате предусмотрена перемычка «INT», впаивание которой приведет к замыканию вывода INT микросхемы W5100 с цифровым выводом 2 Arduino. Однако следует помнить, что обработка таких событий не поддерживается библиотекой Ethernet.

Ниже приводится внешний вид интернет страницы (рис. 9.2), которая загружается с SD карты Ethernet шилда W5100. На интернет странице отображаются данные считываемые с платы расширения Arduino Ethernet, а также состояния выводов отладочной платы, Arduino UNO R3, на которые подключены светодиоды. На интернет странице отображаются значения температуры и влажности с датчика DHT11 отладочной платы и данные с аналогового порта A0. При нажатии на кнопки на интернет странице загораются светодиоды на отладочной плате, при этом цвет кнопки, на интернет странице, изменяется с красного на зеленый (красный – светодиод выключен, зеленый – светодиод включен), при повторном нажатии светодиоды отключаются. Положение ползунка на интернет странице задает интенсивность свечения светодиода отладочной платы.

Рис. 9.2. Внешний вид интернет страницы

В заголовке файла *Index.htm* задаются тип контента и кодировка текста (запросы серверу будут передаваться в кодировке UTF-8), а также указывается название иконки интернет страницы, которое будет запрашиваться у WEB сервера.

```
<html>

<head>
<meta http-equiv='content-type' content='text/html; charset=UTF-8'>
<link rel="shortcut icon" href="/favicon.ico" type="image/x-icon">
```

Далее указывается основной заголовок web – страницы (Лабораторная работа №9 «WEB контроль») и дается ссылка на каскадные таблицы стилей – файл *my.ccs*.

```
<title>Лабораторная работа №9 "WEB контроль"</title>
 <link type="text/css" rel="StyleSheet" href="/my.css" />
```

Далее в html файле следует скрип, в котором выполняется ряд функций. Функция GetFlameState() используется для обновления данных с датчиков и опроса состояния портов.

```
<script>
function GetFlameState () {
nocache = "&nocache=" + Math.random() * 1000000;
var request = new XMLHttpRequest ();
request.onreadystatechange = function() {
if (this.readyState == 4) {
if (this.status == 200) {
if (this.responseText != null) {
var arrayOfStrings = this.responseText.split(":");
```

Для параметра nocache устанавливается случайное значение из диапазона от 0 до 1000000, что обеспечивает разное содержимое каждого URL-адреса, что препятствует обслуживанию запросов из кеша. Далее проверяется статус запроса. Он может иметь любое из следующих значений:

– 0 – неинициализированное;

– 1 – загрузка;

– 2 – загружено;
– 3 – интерактивно;
– 4 – выполнено.

Далее проверяется код состояния HTTP (this.status == 200), возвращаемый сервером и данные, возвращаемые сервером в текстовом формате (this.responseText != null). Данные полученные от сервера разделены двоеточием.

```
document.getElementById("flame_txt").innerHTML= arrayOfStrings[0];
document.getElementById("temp_txt").innerHTML= arrayOfStrings[1];
document.getElementById("humid_txt").innerHTML= arrayOfStrings[2];
```

В массивах arrayOfStrings [0], arrayOfStrings [1] и arrayOfStrings [2] содержится информация, полученная от сервера о коде счинанном с датчика дыма, влажности и температуры.

Далее выполняется проверка массива arrayOfStrings [0], в котором хранится число, соответствующее уровню сигнала считанного с датчика дыма на отладочной плате.

```
if (arrayOfStrings [0] < 340) {
document.getElementById("flame_img").src = "flame.png";
} else {
document.getElementById("flame_img").src= "flame1.png";
}
```

При значении меньше 340 при отображении картинки состояния датчика дыма, будет отображаться картинка flame.png, а при значении больше 340 будет отображаться картинка flame1.png.

```
for (var i = 1; i < 5; i++)
if(arrayOfStrings[2+i] == "1")

document.getElementById("led_"+i).setAttribute("class","button_ena-
bled");
else

document.getElementById("led_"+i).setAttribute("class","button_disa-
bled");
```

После происходит проверка массива arrayOfStrings[] с 3-го по 7-й. Если значение массива равно единице, то изменяется стиль отображения кнопки (кнопка меняет свой цвет с красного на зеленый), а если значение массива равно нулю, то кнопка остается красного цвета или меняет свой цвет с зеленого на красный.

Далее один раз в секунду отправляется запрос на сервер в виде «GET» «ajax_flame».

```
request.open("GET", "ajax_flame" + nocache, true);
request.send(null);
setTimeout ('GetFlameState ()', 1000);
```

После чего следует функция *onClick(pin)* в которой формируются запросы в виде «GET» «\setpin?pin=» + номер вывода отладочной платы при нажатии на кнопку на интернет странице, а также функция PWM() в которой формируются запросы в виде «GET», «\setpin?pin=5?value=» + значение на которое перемещен ползунок (отвечающий за яркость свечения светодиода) на интернет странице.

```
function onClick(pin) {
var request = new XMLHttpRequest ();
request.open("GET", "\setpin?pin=" + pin , false);
request.send(null);
}
function PWM () {
value = document.getElementById("led_PWM").value;
var request = new XMLHttpRequest ();
request.open("GET", "\setpin?pin=5?value=" + value, false);
request.send(null);
}
```

Далее следует отрисовка картинок, текста, кнопок и ползунка на интернет страничке.

Для выполнения лабораторной работы скопируйте следующие файлы в корневой каталог mikro-SD:

– Index.htm;
– my.css;
– favicon.ico;
– flame.png;

– flame1.png;
– humid.png;
– temp.png.

Загрузите в отладочную плату скетч *internet_demo.ico.* Подключите сетевой кабель к Ethernet шилду W5100 и источник питания к отладочной плате. Откройте на любом из компьютеров WEB браузер и в строке адреса введите IP: 192.168.151.201. После загрузки страницы проверьте возможность управления светодиодами отладочной платы и изменение иконки датчика дыма при достижении сигнала на аналоговом входе A0 выше значения 340.

Комментарии, которые приводятся в файлах (internet_demo.ico, Index.htm, my.css), поясняют способ организации обмена данными между отладочной платой и удаленным компьютером.

Задание для самостоятельной работы

1. Измените стиль оформления интернет страницы (цвет фона, размер шрифта и т.д.).
2. В заголовке интернет страницы укажите свои фамилию, имя и отчество.
3. Добавьте пятую кнопку на интернет страницу для управления светодиодом отладочной платы (вывод 8).
4. Добавьте второй ползунок на интернет страницу для управления яркостью светодиода отладочной платы (вывод 9).
5. В папке для выполнения лабораторной работы есть пять картинок, отображающих уровень заряда батареи, используйте их для отображения изменения сигнала на аналоговом входе A0 отладочной платы.

Контрольные вопросы

1. Назовите основные возможности SPI.
2. Дайте описание платы расширения Arduino Ethernet.
3. Охарактеризуйте каждый из светодиодов-индикаторов, представленных на плате.
4. Каким образом плата расширения Arduino Ethernet генерирует сигналы прерываний?
5. Какое основное значение функции GetFlameState()?

Лабораторная работа № 10.

Обмен данными между двумя устройствами по шине I2C

В этой лабораторной узнаем о протоколе обмена данными I2C, о реализации этого протокола обмена данными в микроконтроллерах *Atmega328*, о способе осуществления обмена данными между двумя устройствами в среде програмирования Arduino IDE.

В итоге, мы напишем программу для двух микроконтроллеров, в которых будет осуществлен обмен данными между двумя устройствами.

Цель работы: приобрести практические навыки по организации управления сторонними устройствами и обменом данных между микроконтроллерами в разрабатываемых устройствах.

Последовательность выполнения работы:

1. Изучить теоретические сведения, приведенные в лабораторной работе.
2. Изучить принцип работы схемы на макетной плате для приведенных примеров.
3. Запрограммировать микроконтроллер согласно тексту программы указанному в примере.
4. Выполнить задание для самостоятельной работы.
5. Ответить на контрольные вопросы.

Содержание отчета:

1. Название лабораторной работы, ее цель.
2. Задание на лабораторную работу (по варианту).
3. Схемы подключения к микроконтроллеру.
4. Программный код для скетчей.
5. Вывод о проделанной работе.
6. Ответы на контрольные вопросы.

Теоретические сведения

Описание интерфейса I2C. Последовательный протокол обмена данными IIC (также называемый I2C – Inter-Integrated Circuits, межмикросхемное соединение) использует для передачи данных две двунаправленные линии связи, которые называются шина последовательных данных SDA (Serial Data) и шина тактирования SCL (Serial Clock). Также имеются две линии для питания. Шины SDA и SCL подтягиваются к шине питания через резисторы.

В сети есть хотя бы одно *ведущее устройство (Master),* которое инициализирует передачу данных и генерирует сигналы синхронизации. В сети также есть *ведомые устройства (Slave),* которые передают данные по запросу ведущего. У каждого ведомого устройства есть уникальный адрес, по которому ведущий и обращается к нему. Адрес устройства указывается в паспорте (datasheet). К одной шине I2C может быть подключено до 127 устройств, в том числе несколько ведущих. К шине можно подключать устройства в процессе работы, т.е. она поддерживает «горячее подключение» (рис. 10.1).

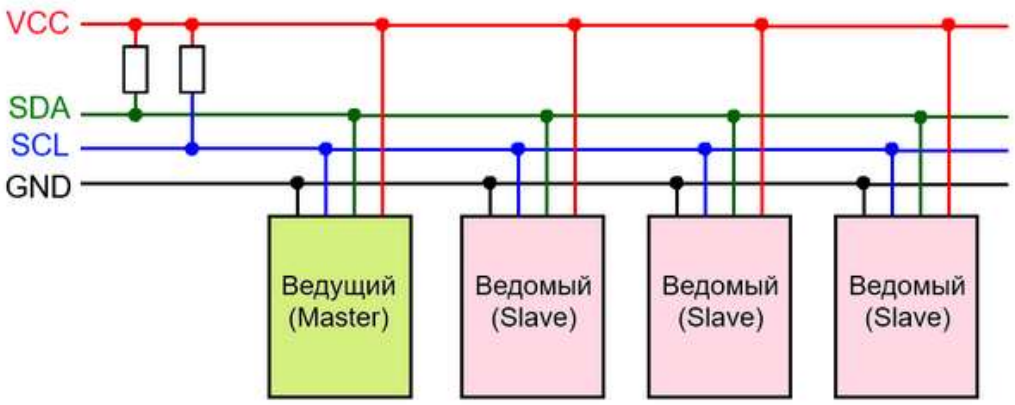


Рис. 10.1. Пример подключения устройств по шине I2C

Далее рассмотрим временную диаграмму обмена данными по протоколу I2C. Есть несколько различающихся вариантов, рассмотрим один из распространённых. Воспользуемся логическим анализатором, подключённым к шинам SCL и SDA.

Мастер инициирует обмен данными. Для этого он начинает генерировать тактовые импульсы и посылает их по линии SCL пакетом, состоящим из 9-ти импульсов. Одновременно на линии данных SDA он выставляет *адрес устройства*, с которым необходимо установить связь, которые тактируются первыми 7-ми тактовыми импульсами (отсюда ограничение на диапазон адресов: $2^7 = 128$ минус

нулевой адрес). Следующий бит посылки – это *код операции* (чтение или запись) и ещё один бит – *бит подтверждения* (ACK), что ведомое устройство приняло запрос. Если бит подтверждения не пришёл, на этом обмен заканчивается. Или мастер продолжает посылать повторные запросы.

Это проиллюстрировано на рисунке 10.2 ниже. Задача такая: подключиться к ведомому устройству с адресом 0x27 и передать ему строку «SOLTAU.RU». В первом случае, для примера, отключим ведомое устройство от шины. Видно, что мастер пытается установить связь с устройством с адресом 0x27, но не получает подтверждения (NAK). Обмен заканчивается.

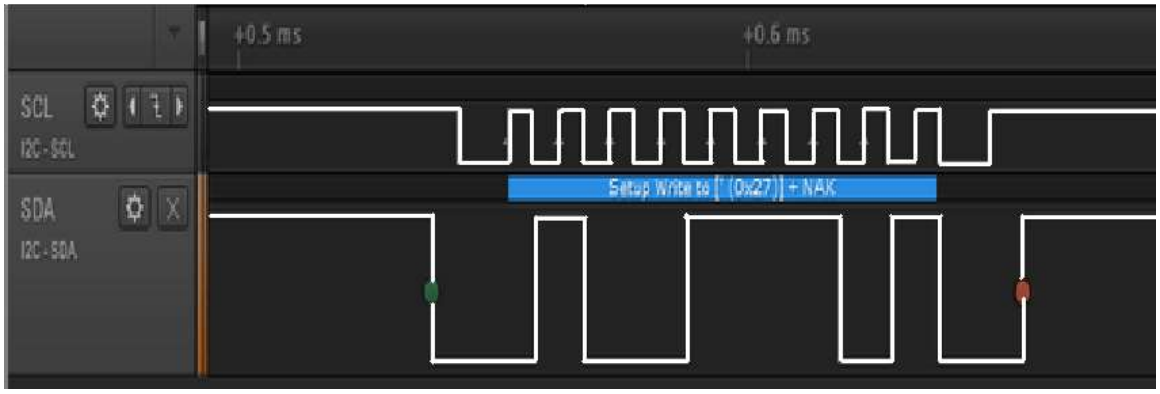


Рис. 10.2. Временная диаграмма при попытке установить соединение ведущего устройства с ведомым устройством по шине I2C

Теперь подключим к шине I2C ведомое устройство и повторим операцию. Ситуация изменилась (рис. 10.3). На первый пакет с адресом пришло подтверждение (ACK) от ведомого. Обмен продолжился. Информация передаётся также 9-битовыми посылками, но теперь 8 битов занимают *данные* и 1 бит – *бит подтверждения* получения ведомым каждого байта данных. Если в какой-то момент связь оборвётся и бит подтверждения не придёт, мастер прекратит передачу.

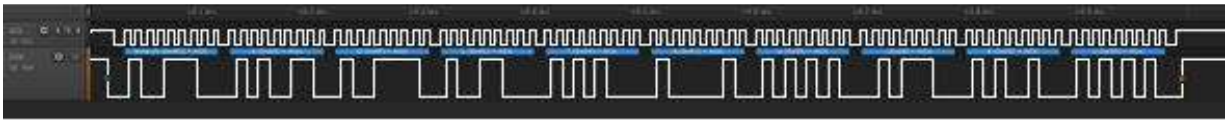


Рис. 10.3. Временная диаграмма обмена по протоколу I2C

***Библиотека «Wire»* для работы с I2C.** Для облегчения обмена данными с устройствами по шине I2C для Arduino написана стандартная библиотека **Wire**. Она имеет функции (таблица 10.1).

Таблица 10.1. Функции стандартной библиотеки Wire

Функция	Назначение
begin(address)	инициализация библиотеки и подключение к шине I2C; если не указан адрес, то присоединённое устройст во считается ведущим; используется 7-битная адресация;
requestFrom()	используется ведущим устройством для запроса определённого количества байтов от ведомого;
beginTransmission(address)	начало передачи данных к ведомому устройству по определённому адресу;
endTransmission()	прекращение передачи данных ведомому;
write()	запись данных от ведомого в ответ на запрос;
available()	возвращает количество байт информации, доступных для приёма от ведомого;
read()	чтение байта, переданного от ведомого ведущему или от ведущего ведомому;
onReceive()	указывает на функцию, которая должна быть вызвана, когда ведомое устройство получит передачу от ведущего;
onRequest()	указывает на функцию, которая должна быть вызвана, когда ведущее устройство получит передачу от ведомого.

Управление устройством по шине I2C. Рассмотрим диаграммы информационного обмена (рис. 10.4) с цифровым потенциометром AD5171, представленные в техническом описании.

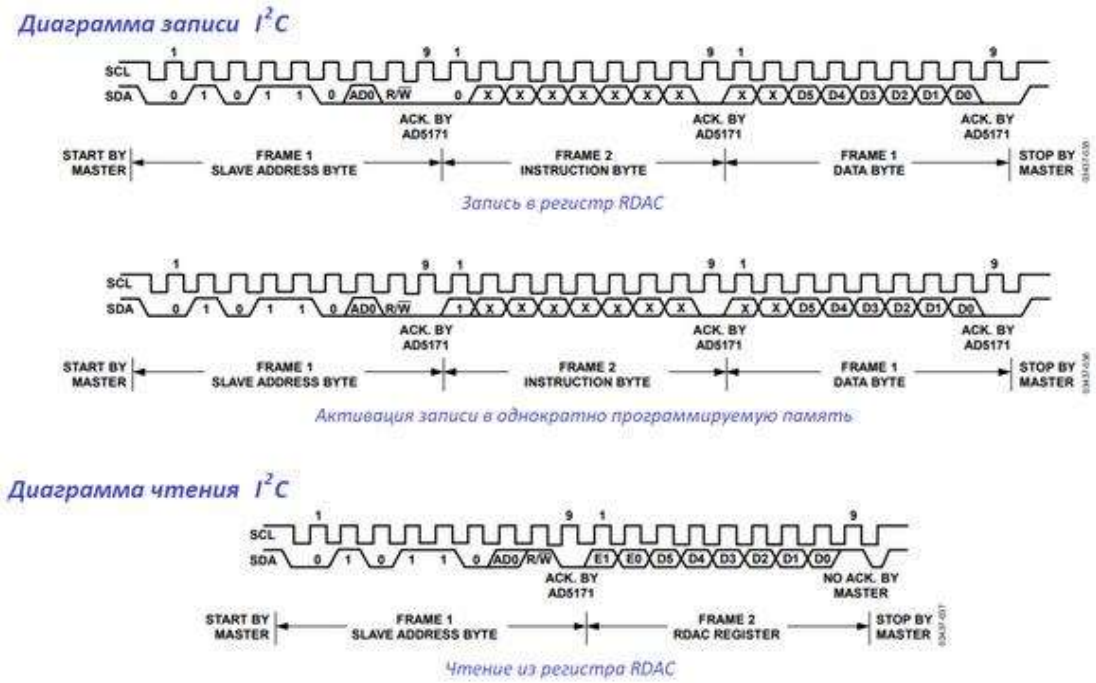


Рис. 10.4. Диаграмма чтения и записи по протоколу I2C

Библиотека iarduino_I2C_connect разработана для удобства соединения нескольких arduino по шине I2C.

Пример подключения библиотеки I2C:

```
#include <Wire.h>     // подключаем библиотеку для работы с шиной I2C
#include <iarduino_I2C_connect.h>// подключаем библиотеку для //соединения arduino по шине I2C
// Создание библиотеки объекта
iarduino_I2C_connect I2C2;
// можно указать свое название вместо I2C2
```

В библиотеке реализованы 4 функции:

Для ведомого:

Функция begin. Назначение: указание массива, элементы которого будут доступны для чтения/записи по шине I2C. Синтаксис: begin(массив). Параметр: массив (тип byte). Возвращаемые значения: Нет. *Примечание:* Вызывается 1 раз в начале кода.

Функция writeMask. Назначение: указание маскировочного массива, значение true элемента данного массива – разрешает менять значение соответствующего элемента в массиве, объявленном в функции begin(). Синтаксис: writeMask (маскировочный_массив). Параметр: маскировочный массив (тип bool). Возвращаемые значения: Нет. *Примечание*: Вызывается 1 раз в начале кода.

Если указать массив, который состоит из 5 ячеек {0,1,1,1,0}, то мастер не сможет изменить значения 0 и 4 элементов массива доступного по шине I2C, который был объявлен в функции begin(). Но все элементы массива будут доступны для чтения.

Если указать массив, который состоит из 5 ячеек, в то время как массив доступный по шине I2C содержит 10 ячеек, то мастер не сможет изменить значения последних 5 ячеек, а значения первых сможет изменить только если соответствующие ячейки маскировочного массива равны 1 или true.

Если не объявлять маскировочный массив (не вызывать функцию writeMask), то все элементы массива, доступного по шине I2C, который был объявлен в функции begin(), будут доступны мастеру для записи и чтения.

Для ведущего:

Функция readByte. Назначение: Чтение одного байта данных

из устройства на шине I2C. Синтаксис: readByte (адрес устройства, адрес регистра). Параметры: адрес устройства – данные которого требуется получить (число от 0x00 до 0x7F). Адрес регистра – данные которого требуется получить (число от 0x00 до 0xFF) // адрес регистра соответствует номеру элемента массива, объявленного ведомым устройством функцией begin. Возвращаемые значения: uint8_t байт – данные, считанные из указанного адреса, указанного устройства.

Функция writeByte. Назначение: Запись одного байта данных в устройство на шине I2C. Синтаксис: writeByte (адрес устройства, адрес регистра, байт данных). Параметры: адрес устройства – которому предназначены отправляемые данные (число от 0x00 до 0x7F).

Адрес регистра – которому предназначены отправляемые данные (число от 0x00 до 0xFF) // адрес регистра соответствует номеру элемента массива, объявленного ведомым устройством функцией begin.

Байт данных – который требуется записать в указанный регистр, указанного устройства

Возвращаемые значения: uint8_t байт – результат операции записи, соответствует значению, возвращаемому функцией Wire.endTransmission():

– 0 – успех;

– 1 – переполнение буфера передачи данных;

– 2 – получен NACK на переданный адрес устройства;

– 3 – получен NACK на переданный адрес регистра;

– 4 – неизвестная ошибка.

При создании некоторых проектов, требуется разделить выполняемые задачи между несколькими микропроцессорами.

В этой лабораторной мы научимся соединять две Arduino по аппаратной шине I2C.

Преимущества:

– реализуется возможность подключения до 126 устройств. Не рекомендуется присваивать устройствам адреса 0x00 и 0x7F;

– не требуются дополнительные модули;

– все устройства одинаково подключаются к шине;

– каждое ведомое устройство имеет свой уникальный адрес на шине.

Недостатки:

– Код программы немного сложнее чем для шины UART.

Настройка параметров шины I2C. Максимальная, аппаратно-реализуемая частота передачи данных, может достигать 1/16 от тактовой частоты.

Библиотека Wire позволяет устанавливать скорость передачи данных через функцию setClock(), которую требуется вызвать до функции begin().

Библиотека Wire позволяет аппаратно подключить Arduino к шине I2C с указанием роли Arduino на шине: ведущий или ведомый.

– *Wire.setClock(400000);* // скорость передачи данных 400 кБит/с.

– *Wire.begin();* // подключение к шине I2C в роли ведущего.

– *Wire.begin(адрес);* // подключение к шине I2C в роли ведомого, с указанием адреса.

Функции библиотеки iarduino_I2C_connect. В библиотеке *iarduino_I2C_connect* реализованы 4 функции: 2 для ведущего и 2 для ведомого.

На ведомом устройстве достаточно вызвать функцию begin() с указанием массива, данные которого требуется сделать доступными для мастера. Далее можно работать с этим массивом, «забыв» про шину I2C. Мастер, обращаясь к ведомому сможет получать и изменять данные элементов указанного массива (обращаясь к номеру элемента массива как к номеру регистра).

Если требуется запретить мастеру менять значения некоторых ячеек массива, то достаточно вызвать «необязательную» функцию writeMask() с указанием маскировочного массива, каждый элемент которого является флагом, разрешающим запись в соответствующий элемент массива доступного по шине I2C.

На ведущем устройстве доступны функции readByte() и writeByte(). Указывая в качестве параметров функций, адрес ведомого устройства и номер регистра, можно побайтно читать и записывать данные.

Функции для ведомого:

Begin():

– назначение: указание массива, элементы которого будут

доступны для чтения/записи по шине I2C;

– синтаксис: begin(массив); // тип данных массива – byte;

– возвращаемые значения: Нет;

– примечание: Вызывается 1 раз в коде функции Setup().

writeMask():

– назначение: указание маскировочного массива, каждый элемент которого является флагом разрешения записи по шине I2C;

– синтаксис: writeMask(маскировочный_массив); // тип данных массива – bool;

– возвращаемые значения: Нет;

– примечание: Вызывается 1 раз в коде функции Setup();

Пример:

```
#include <Wire.h>
#include <iarduino_I2C_connect>
iarduino_I2C_connect ccc;
byte aaa [5]; // объявляем 5 элементов массива, к которому разрешим
//доступ.
bool bbb [5] = {0,1,1,1,0}; // объявляем 5 элементов маскировочного
//массива.
Wire.begin(0x33); // подключаемся к шине I2C в роли ведомого с адресом
//0x33.
ccc.begin(aaa); // разрешаем доступ к массиву aaa по шине I2C.
ccc.wtiteMask(bbb); // маскируем разрешение на запись в массив aaa.
```

– в соответствии со значениями массива bbb, мастер сможет записывать данные в ячейки 1,2,3 массива aaa, а попытка записи в 0 и 4 элементы будет проигнорирована;

– если массив aaa имеет больше элементов чем массив bbb, то попытка мастера записать данные в «лишние» элементы будет проигнорирована;

– если не объявлять маскировочный массив (не вызывать функцию writeMask () вообще), то все элементы массива aaa будут доступны для записи мастером.

Функции для ведущего:

readByte():

– назначение: Чтение одного байта данных из устройства на шине I2C;

– синтаксис: readByte (адрес устройства, адрес регистра);

– возвращаемые значения: uint8_t байт – данные, считанные из указанного адреса, указанного устройства;

– примечание: если производится чтение по шине I2C из массива, объявленного функцией begin(), то адрес регистра соответствует номеру элемента массива.

writeByte():

– назначение: Запись одного байта данных в устройство на шине I2C;

– синтаксис: writeByte (адрес устройства, адрес регистра, байт данных);

– возвращаемые значения: uint8_t байт – результат операции записи, соответствует значению, возвращаемому функцией Wire.endTransmission().

– 0 – успех;

– 1 – переполнение буфера передачи данных;

– 2 – получен NACK на переданный адрес устройства;

– 3 – получен NACK на переданный адрес регистра;

– 4 – неизвестная ошибка.

– примечание: если производится запись по шине I2C в массив, объявленный функцией begin(), то адрес регистра соответствует номеру элемента массива.

Порядок выполнения лабораторной работы

Для выполнения лабораторной работы нам понадобится установить библиотеки:

– библиотека *LiquidCrystal* (для подключения дисплея LCD1602);

– библиотека *iarduino_I2C_connect* (для удобства соединения нескольких микроконтроллеров Atmega328P по шине I2C).

Схема подключения:

На шине I2C находятся 2 устройства: 2-е отладочные платы. Все устройства шины I2C соединены через аппаратные выводы шины I2C.

Отладочная плата master – к аналоговому выводу A1 и A2 подключены кнопки, а к выводу A0 светодиод. Также на отладочной плате находится двухстрочный ЖК-дисплей.

Отладочная плата slave 0x02 – к цифровому выводу D3 подключен температурный датчик DALAS 18B20, а также к выводу D7-красный светодиод, к выводу D8 - зеленый светодиод. Также на отладочной плате находится модуль считывания RFID меток (выводы D9, D10, D11, D12, D13) и панель считывателя ключа ibutton (вывод D2).

На шине I2C используются дополнительные подтягивающие резисторы (для линий SDA и SCL).

Для установления обмена данными между платами необходимо их запрограммировать. Код для отладочных плат приведен ниже в примерах.

Код программы. Отладочная плата master.

Пример:

```
// Подключаем библиотеки:
#include <EEPROM.h>
#include <Wire.h>   // подключаем библиотеку для работы с шиной I2C
#include <LiquidCrystal.h> // подключаем библиотеку для работы с LCD дисплеем
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd (rs, en, d4, d5, d6, d7);
#include <iarduino_I2C_connect.h> // подключаем библиотеку для соединения arduino по шине I2C
// Объявляем переменные и константы:
iarduino_I2C_connect I2C2; // объявляем переменную для работы с библиотекой iarduino_I2C_connect
// объявляем константу с указанием номера цифрового вывода, к которому подключена кнопка
const byte PIN_Button_master = A2;
// объявляем переменную для чтения состояний собственной кнопки
bool VAR_Button_master = 0;
// объявляем переменную для чтения состояний кнопки ведомого
bool VAR_Button_slave = 0;
// объявляем переменную для чтения значения с температурного датчика ведомого по адресу 0x02
int VAR_Potentiometer0 = 0;
int VAR_Potentiometer1 = 0;
int VAR_Potentiometer2 = 0;
int VAR_Potentiometer3 = 0;
float VAR_Potentiometer = 0;
void setup () {
```

Продолжение примера

```
Serial.begin(9600);
lcd.begin(16, 2);
Wire.begin(); // инициируем подключение к шине I2C в качестве ведущего (master) устройства
pinMode (PIN_Button_master, INPUT); // Устанавливаем режим работы вывода собственной кнопки, как вход
pinMode (A0, OUTPUT);
//Выводим данные на LCD дисплей:
lcd.print ("Lab 10"); // выводим текст "Lab 10"
delay (1000); // ждём 1 секунду
lcd.setCursor(0,0); lcd.print("knopka = "); // выводим текст "knopka = "
lcd.setCursor(0,1); lcd.print("temper = "); // выводим текст "temper = "
}
void loop (){
//Считываем данные:
VAR_Potentiometer0 = I2C2.readByte(0x02,0);
//выводим в монитор порта 0-й байт значения температуры
Serial.print(I2C2.readByte(0x02,0)); Serial.print("     ");
//записываем память полученное значение
EEPROM.write(0, VAR_Potentiometer0);
VAR_Potentiometer1 = I2C2.readByte(0x02,1);
//выводим в монитор порта 1-й байт значения температуры
Serial.print(I2C2.readByte(0x02,1)); Serial.print("     ");
//записываем память полученное значение
EEPROM.write(1, VAR_Potentiometer1);
VAR_Potentiometer2 = I2C2.readByte(0x02,2);
//выводим в монитор порта 2-й байт значения температуры
Serial.print(I2C2.readByte(0x02,2)); Serial.print("     ");
//записываем память полученное значение
EEPROM.write(2, VAR_Potentiometer2);
VAR_Potentiometer3 = I2C2.readByte(0x02,3);
//выводим в монитор порта 3-й байт значения температуры
Serial.print(I2C2.readByte(0x02,3)); Serial.print("     ");
//записываем память полученное значение
EEPROM.write(3, VAR_Potentiometer3);
//считываем значение температуры с памяти
VAR_Potentiometer = EEPROM_float_read (0);
//выводим в монитор порта значения температуры
Serial.print(I2C2.readByte(0x02,5)); Serial.print("     ");
//выводим в монитор порта состояние кнопки ведомого устройства
Serial.println(VAR_Potentiometer,5);
delay (50); // задержка между считываниями
```

Продолжение примера

```
VAR_Button_master = digitalRead(PIN_Button_master); // Считываем состояние собственной кнопки
// Считываем состояние кнопки ведомого (адрес ведомого 0x02, номер регистра 4)
VAR_Button_slave   = I2C2.readByte(0x02,4);
//Отправляем данные:
// Отправляем состояние собственной кнопки ведомому (адрес ведомого 0x02, номер регистра 4, состояние кнопки)
I2C2.writeByte(0x02,4, VAR_Button_master);
//в случае нажатия кнопки на ведущем устройстве включаем светодиод на ведомом устройстве
digitalWrite (A0, VAR_Button_master);
//Выводим данные на LCD дисплей:
lcd.setCursor(11, 0); lcd.print(" ");  // стираем предыдущее состояние кнопки ведомого 0x02
lcd.setCursor(11, 0); lcd.print(VAR_Button_slave);  // выводим состояние кнопки ведомого 0x02
lcd.setCursor(11, 1); lcd.print(" "); // стираем предыдущее значение потенциометра ведомого 0x02
lcd.setCursor(11, 1); lcd.print(VAR_Potentiometer); // выводим значение потенциометра ведомого 0x02
delay (50); // ждём 0.05 секунд, иначе данные на LCD дисплее будут заметно мерцать
}
//функция записи и считывания данных в память
void EEPROM_float_write (int addr, float val) // запись в ЕЕПРОМ
{
byte *x = (byte *) &val;
for (byte i = 0; i < 4; i++) EEPROM.write(i+addr, x[i]);
}
float EEPROM_float_read (int addr) // чтение из ЕЕПРОМ
{
byte x [4];
for (byte i = 0; i < 4; i++) x[i] = EEPROM.read(i+addr);
float *y = (float *) &x;
return y [0];
}
```

Arduino slave 0x01.

Пример:

```
// Подключаем библиотеки:
#include <EEPROM.h>
#include <Wire.h> // подключаем библиотеку для работы с шиной I2C
#include <iarduino_I2C_connect.h>// подключаем библиотеку для соединения arduino по шине I2C
// Объявляем переменные и константы:
#include <OneWire.h>//Подключаем библиотеку общения по одному проводу
#define SLAVE_ADDRESS 0x02
#define FLOATS_SENT 6
OneWire ds (3); // Вывод 3 отладочной платы
float celsius; //Переменная в которой сохраняется значение считанной температуры
//Обьявляем переменные в которых будут хранится:
float data[FLOATS_SENT];
// объявляем переменную для работы с библиотекой iarduino_I2C_connect
iarduino_I2C_connect I2C2;
// объявляем константу с указанием номера аналогового вывода, к которому подключён потенциометр
const byte PIN_Potentiometer = 0;
// объявляем константу с указанием номера цифрового вывода, к которому подключен светодиод
const byte PIN_LED = 8;
// объявляем переменную для чтения значения с потенциометра
float VAR_Potentiometer = 0;
// объявляем массив, данные которого будут доступны для чтения/записи по шине I2C
byte REG_Massive [6];
void setup () {
Serial.begin(9600); // //Разрешаем обмен данными по порту UART
pinMode (PIN_LED, OUTPUT);
// инициируем подключение к шине I2C в качестве ведомого (slave) устройства, с указанием своего адреса на шине.
Wire.begin(0x02);
I2C2.begin(REG_Massive);
Wire.begin(SLAVE_ADDRESS);
}
void loop () {
byte i; //Текущий байт массива данных
```

Продолжение примера

```
byte data [12]; //Значение данных считанных с температурного датчика
byte addr [8]; //Значение адреса, считанного с температурного датчика
if (! ds.search(addr)) { //Поиск адресов температурных датчиков
ds.reset_search(); //Если адреса не найдены то поиск останавливается
// delay (250); // на четверть секунды
return;
}
ds.reset(); //Выполняется сброс температурного датчика
ds.select(addr); // Выбираем считанный адрес
ds.write(0x44, 1); // Начинаем преобразование
delay (200); // Время необходимое для преобразования зависит
// от разрешения считываемой величины
ds.reset(); //Выполняется сброс температурного датчика
ds.select(addr);
ds.write(0xBE); // Считываем 9 байт данных с температурного датчика
for (i = 0; i < 9; i++) {
data[i] = ds.read();
}
//Считываем старший и младший байт значения температуры
int16_t raw = (data [1] << 8) | data [0];
raw = raw & ~7; //9, битное разрешение время считывания 93.75 мс
//raw = raw & ~3; //10, битное разрешение время считывания 187.5 мс
//raw = raw & ~1; //11, битное разрешение время считывания 375   мс
//raw = raw & ~0; //12, битное разрешение время считывания 750   мс
celsius = (float)raw / 16.0; // определяем температуру в градусах цельсия
//delay (1000);
// int sensorValue = analogRead(A0);
//VAR_Potentiometer = celsius; // Считываем значение потенциометра
//EEPROM_float_write (0, VAR_Potentiometer);
EEPROM_float_write (0, celsius);
delay (20);
// Сохраняем старший байт значения потенциометра в 0 ячейку массива REG_Massive
REG_Massive [0] = EEPROM.read(0);
REG_Massive [1] = EEPROM.read(1);
// Сохраняем старший байт значения потенциометра в 2 ячейку массива REG_Massive
REG_Massive [2] = EEPROM.read(2);
REG_Massive [3] = EEPROM.read(3);
```

Продолжение примера

```
REG_Massive [5] = 1;
Serial.println(celsius); //вывод температуры с датчика в монитор порта
//включение светодиода в зависимости от нажатия кнопки на ведомом устройстве
digitalWrite (PIN_LED, REG_Massive [4]);
}
//функция записи и считывания данных в память
void EEPROM_float_write (int addr, float val) // запись в ЕЕПРОМ
{
byte *x = (byte *) &val;
for (byte i = 0; i < 4; i++) EEPROM.write(i+addr, x[i]);
}
float EEPROM_float_read (int addr) // чтение из ЕЕПРОМ
{
byte x [4];
for (byte i = 0; i < 4; i++) x[i] = EEPROM.read(i+addr);
float *y = (float *) &x;
return y [0];
}
```

Алгоритм работы:

– Отладочная плата *master* проверяет состояние собственной кнопки.

– Отладочная плата *master* опрашивает отладочную плату *Slave*.

– Отладочная плата Slave 0x02 возвращает значение температуры с цифрового датчика.

– Отладочная плата *master* мастер отправляет состояние своей кнопки в отладочную плату Slave 0x02.

– Отладочная плата Slave 0x02 включает или выключает светодиод, в соответствии с полученными данными.

– Отладочная плата *master* выводит данные о состояниях кнопки отладочной платы *Slave* 0x02 и температуры с датчика на LCD дисплей и в монитор порта на компьютер.

Задание для самостоятельной работы

1. Запрограммируйте отладочные платы *master* и *Slave* 0x02, так чтобы при нажатии кнопок на ведущей плате (выводы A1 и A2) загорались красный и зеленый светодиод ведомой платы (выводы D7 и D8).

2. Запрограммируйте отладочные платы *master* и *Slave* 0x02, так чтобы при поднесении ключа ibutton к считывателю отладочной платы *Slave* 0x02 (вывод D2), на ЖК-дисплее, отладочной платы *master*, отображался код ключа ibutton, а также состояние нажатия кнопок в виде текста («Разрешено», «Запрещено»).

Контрольные вопросы

1. Понятие интерфейса.
2. Синхронный последовательный интерфейс I2C. Сигналы интерфейса и их уровни.
3. Конфигурация I2C – шины.
4. Протокол связи I2C.
5. Адресация на шине I2C. Основные типы передачи данных.

ЗАКЛЮЧЕНИЕ

Программное обеспечение является одним из необходимых условий функционирования любой, вычислительной (или как сейчас говорят – цифровой) системы. И не так важно, какое оно – встроенное в оборудование, загружаемое ли с внешнего носителя, в любом случае только при его правильной работе компьютерная система будет выполнять те действия, которые от нее требуются.

Сегодня процесс автоматизации требует дополнительного применения датчиков (сенсоров), устройств ввода, управляющих устройств (контроллеров), исполнительных устройств, устройств вывода, использующих электронную технику и методы вычислений, иногда копирующие нервные и мыслительные функции человека.

Процесс разработки любого устройства управления обязательно включает разработку алгоритма. Алгоритм – точный набор инструкций, описывающих порядок действий для решения задачи за конечное время. Особенностью устройств управления на базе микропроцессорного устройства является взаимодействие аппаратных узлов и программного обеспечения, что усложняет разбиение алгоритма на элементы.

Подобные микропроцессорные устройства, реализованные в одной микросхеме (в одном кристалле) называются микроконтроллерами, они хорошо приспособлены для построения узлов или законченных систем управления, имеют небольшую цену, габариты и вес. Наибольшее распространение среди них получили 8-разрядные микроконтроллеры. Это объясняется тем, что большинство задач управления не требует значительной арифметической производительности, большую часть задач составляют задачи логической обработки, задачи ввода/вывода и четкая координация выполнения задач во времени.

В данном пособии проявляется достаточно близкое знакомство с семейством 8-разрядных микроконтроллеров AVR фирмы Atmel. Выбор данного семейства обусловлен хорошим соотношением цена/качество, доступностью на отечественном рынке, богатым инструментарием по программированию и отладке. Основной упор в изучении микроконтроллеров сделан на освоение принципов

программного управления, механизмов прерывания, использованию таймеров/счетчиков в задачах формирования и измерения временных интервалов, знакомству с задачами аналогового ввода/вывода и задачами организации обмена данными по последовательному интерфейсу. Лабораторный практикум ориентирован на использование программ-симуляторов, имитирующих поведение микроконтроллера под управлением программы.

Каждая тема сопровождается многочисленными готовыми примерами программ с подробным описанием алгоритма и методики его тестирования в том или ином симуляторе. Для закрепления осваиваемого материала по каждой теме предлагаются наборы заданий разного уровня сложности.

Контрольные вопросы по итогам лабораторного практикума помогают проверить запоминание терминов, определений и понимание пройденного материала.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Водовозов А.М. Элементы систем автоматики / А.М. Водовозов. – М. : Академия, 2008. – 224 с.

2. Ревич Ю.В. Практическое программирование микроконтроллеров AtmelAVR на языке ассемблера / Ю.В. Ревич. – Санкт-Петербург : БХВПетербург, 2011. – 352 с.

3. Белов А.В. Самоучитель разработчика устройств на микроконтроллерах AVR / А.В.Белов. – Санкт-Петербург : Наука и Техника, 2008. – 544 с.

4. Хартов В.Я. Микроконтроллеры AVR / В. Я. Хартов. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2012. – 280 с.

5. Голик С.Е. Микроконтроллеры: архитектура и программирование / С.Е. Голик. – СПб. : Изд-во СПбГЭТУ, 2006. – 451 с.

6. Микропроцессорные устройства: Методические указания к практическим занятиям / Сост.: Д. Н. Бондаренко. СПб. : Изд-во СПбГЭТУ «ЛЭТИ», 2007. – 32 с.

7. PIC-микроконтроллеры. Полное руководство / С. Катцен; пер. с англ. Евстифеева А.В. – М. : ДОДЭКА-XXI, 2010. – 650 с.

8. Решение интерфейсных задач с использованием однокристальных микроконтроллеров: метод. указания к лаб. работам; сост.: А.В. Кекконен, А.А.Тимофеев. – СПб. : Изд-во СПбГЭТУ "ЛЭТИ", 2009. – 39 с.

9. Бальзамов А.Ю. Микроконтроллеры AVR: архитектура и программирование / А.Ю. Бальзамов; Мордов. гос. ун-т. – Саранск, 2014. – 266 с.

10. Торгаев С.Н. Практическое руководство по программированию STM-микроконтроллеров / С.Н. Торгаев, М.В. Тригуб, И.С.Мусоров, Д.С. Чертихина ; Томский политехнический университет. – Томск : Изд-во Томского политехнического университета, 2015. – 111 с.

11. Водовозов А.М. Микроконтроллеры для систем автоматики / А.М. Водовозов ; Мин-во обр. и науки РФ ; Вологод. гос. ун-т. – Изд. 2-е доп. и перераб. – Вологда : ВоГУ, 2015. – 164 с.