Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Компьютерные системы и сети (ИУ6)

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине «Микропроцессорные системы» на тему:

Регистратор температуры

Студент ИУ6-71 _____ **А.Ю. Маранин**

(группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ **В.Я. Хартов**

(Подпись, дата) (И.О.Фамилия)

Москва, 2017

Реферат

Записка 39 с., 24 рис., 3 табл., 18 источников, 3 прил.

ATmega8, КОНТРОЛЛЕР, ПРОЕКТИРОВАНИЕ, МИКРОКОНТРОЛЛЕР, МК-СИСТЕМА, ЭЛЕКТРОПРИБОР, ДАТЧИК ТЕМПЕРАТУРЫ, DS1621, ПРЕОБРАЗОВАТЕЛЬ USB-USART, FT232RL.

Объектом разработки данной курсовой работы является проект контроллера, являющегося регистратором температуры от семи цифровых датчиков типа DS1621. Показания датчиков температуры, превышающие пороговое значение будут поочередно выведены на дисплей из семисегментных индикаторов с указанием номера датчика, от которого получили температуру, а также передана на ПЭВМ по последовательному каналу. Предусмотрена возможность изменения пороговой температуры.

Содержание

Введение

С развитием технологий разработки микросхем, сложные вычислительные задачи могут быть решены с помощью микроконтроллеров, имеющих очень компактные размеры. Форм-фактор и исполнение датчиков и прочих дополнительных элементов также оптимизируются с каждым годом. Это позволяет создавать сложные микроконтроллерные системы в пределах одноплатного модуля, из чего можно заключить, что область применения микроконтроллеров практически безгранична. Подобная разработка может быть актуальна как для домашнего применения, так и, преимущественно, для офисных, лабораторных и других рабочих помещений. Тем не менее, проектирование представленного контроллера регистратора температуры не востребовано и не инновационно с точки зрения какого-либо предприятия или научной новизны, а является таким для конкретного отдельно взятого студента.

Преимущество использования микроконтроллеров семейства AVR в том, что они имеют широкое распространение, простоту использования и невысокую стоимость. Также, их легко программировать, так как они имеют гибкую систему команд и подробную документацию. Полученная система регистрации температуры основана на микроконтроллере ATmega8-16PU семейства ATmega в корпусе DIP-28 (вставляемое в отверстия).

В качестве среды проектирования функциональной схемы и схем алгоритмов был выбран программный пакет Microsoft Visio, для отладки схемотехнических решений использовался Proteus. Также для оформления чертежей использовался САПР «Компас 3D». Для разработки принципиальной схемы использовался программный комплекс Kicad. Разработка программного кода производилась в CodeVisionAVR C Compiler, отладка кода в AVRStudio 4. Все продукты либо бесплатны, либо предоставляли студенческую версию. «Компас 3D» использовался в качестве пробной тридцатидневной версии.

1 Анализ требований и уточнение спецификаций

Согласно техническому заданию, микроконтроллер ATmega8 должен работать как регистратор температуры от семи датчиков типа DS1621. Показатели сравниваются с пороговым значением и при его превышении выводятся на дисплей из семисегментных индикаторов с указанием номера датчика, а также пересылаются на компьютер по последовательному канал.

1.1 Выводная индикация

Для отображения температуры с датчиков индикации использовались один семисегментный дисплей на 4 элемента CC56-12SRWA с общим катодом (красный) и один семисегментный дисплей на 1 элемент SC56-11 с общим катодом (зеленый).

Первый индикатор показывает номер датчика, остальные 4 — температуру.

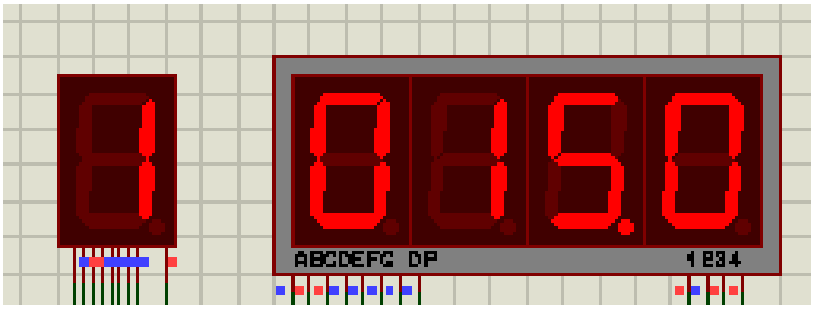


Рисунок 1 – Выводная индикация в симуляторе Proteus

Рисунок 2 – Выводная индикация на макетной плате

2 Проектирование пользовательского интерфейса

2.1 Способ ввода информации

Единственные данные, которые можно ввести в систему — это пороговое значение температуры, в зависимости от которого происходит отображение информации от того или иного датчика. Для ввода порогового значения в систему был выбран способ передачи значения через последовательный канал с ПЭВМ. Он состоит из интерфейса передачи USART микроконтроллера, подключенного к микросхеме-переходнику FT232rl, которая преобразует команды, и информацию из TTL логики в последовательный (Serial)сигнал и наоборот.Далее информация принимается ПЭВМ через любую программу, позволяющую отправлять и принимать данные от COM портов, например PuTTY. Переданная на последовательный канал (USART) информация будет принята микроконтроллером и обработана. Допускается ввод только одного числа диапазоном от - 80 до 128 (ограничение выбрано из-за спецификации датчика DS1621).

2.2 Способ вывода информации

Информация о температуре с измеряемого датчика выводятся на блок семисегментных индикаторов, представленных выше, а также информация передается в последовательный канал (на макете — через переходник FT232rl на компьютер в терминальную программу PuTTY).

3 Разработка технологического макета

3.1 Выбор элементной базы

Из микроконтроллеров был выбран ATmega8, соответствующий техническому заданию.

Датчики были выбраны типа DS1621 из-за широко распространенного и перспективного канала передачи I2C.

Для индикации использовалась 1 семисегментный дисплей на 4 элемента CC56-12SRWA с общим катодом красный и 1 семисегментный дисплей на 1 элемент SC56-11 с общим катодом зеленый.

Для питания использовался стабилизатор напряжения L7805, преобразующего напряжения от 20В до 5В, и соответствующие конденсаторы согласно его технической документации.

Для передачи данных по последовательному каналу с микроконтроллера на ПЭВМ использовалась микросхема-переходник FT232rl, организующая и согласующая связь между USART и USB.

3.2 Выбор средств разработки

Поскольку в требованиях к курсовой работе указаны многие виды документации, необходимо грамотно выбрать средства разработки для этих видов документации.

По предыдущему опыту разработок было решено воспользоваться пакетом MicrosoftOffice для создания некоторых схем и текстовых документов.

Так, например, для создания структурно-функциональной схемы и схемы алгоритмов работы программы устройства было решено использовать приложение Visio из вышеупомянутого пакета. Оно позволяет создавать простейшие схематичные диаграммы из векторных графических примитивов, для оформления чертежей использовался САПР «Компас 3D».

Для создания текстовых и табличных документов для печати в формате А4 было использовано приложение Word из вышеупомянутого программного

пакета. В нём были созданы задание на курсовую работу, лист спецификации и эта расчётно-пояснительная записка.

Для разработки принципиальной электрической схемы устройства был выбран программный пакет Kicad, а для отладки был использован ISIS Proteus.

Разработка программного кода производилась в CodeVisionAVR C Compiler, отладка кода в AVRStudio 4.

3.3 Проектирование принципиальной схемы

Для проектирования принципиальной электрической схемы был выбран программный пакет Kicad.

Поскольку в Kicad отсутствует поддержка ГОСТов и ЕСКД, но есть поддержка пользовательских библиотек компонентов, было решено создать свои собственные элементы, в частности, касающиеся графического представления таблиц-разъёмов.

Использование подобных пользовательских библиотек позволяет применять программы, не поддерживающие отечественные стандарты вместе с ними, то есть добавлять некоторую поддержку ГОСТов, пусть и не идеально.

Тем не менее, поскольку часть стандартной системной графики этой программы неизменяема, окончательная доводка принципиальной схемы до соответствующих стандартов была произведена при помощи программы для редактирования векторной графики Microsoft Visio.

Штампы для всех чертежей использовались из САПР «Компас 3D».

3.4 Реализация алгоритмов и программы

Схема алгоритма работы программы весьма линейна: сначала следует инициализация ресурсов МК, после чего программа уходит в бесконечный цикл по выводу данных на светодиоды и сбору ввода пользователя, изредка перемежающийся просчетом игровой логики.

Схема алгоритма работы программы весьма линейна: сначала следует инициализация ресурсов МК: задание констант, задание начальной пороговой

температуры, установка портов на вводы и выводы, задание регистров, отвечающих за скорость работы I2C и USART, настройки прерываний.

Частота работы I2C = 100КГц, что соответствует оптимальной частотой работы датчика DS1621.

Частота работы последовательного канала USART = 250 килобод.

Исходя из этого была выбрана частота микроконтроллера F = 8 Мгц.

Датчики DS1621 сконфигурированы на постоянный анализ температуры.

После всех предварительных настроек, программа уходит в бесконечный цикл, опрашивает датчики поочередно. Если температура, полученная от датчика выше пороговой номер и показания отобразятся на семисегментном дисплее, а также будут переданы по каналу USART, а далее через преобразователь FT232rl на компьютер, организуется задержка в 3с. Если температура не выше пороговой датчик пропускается.

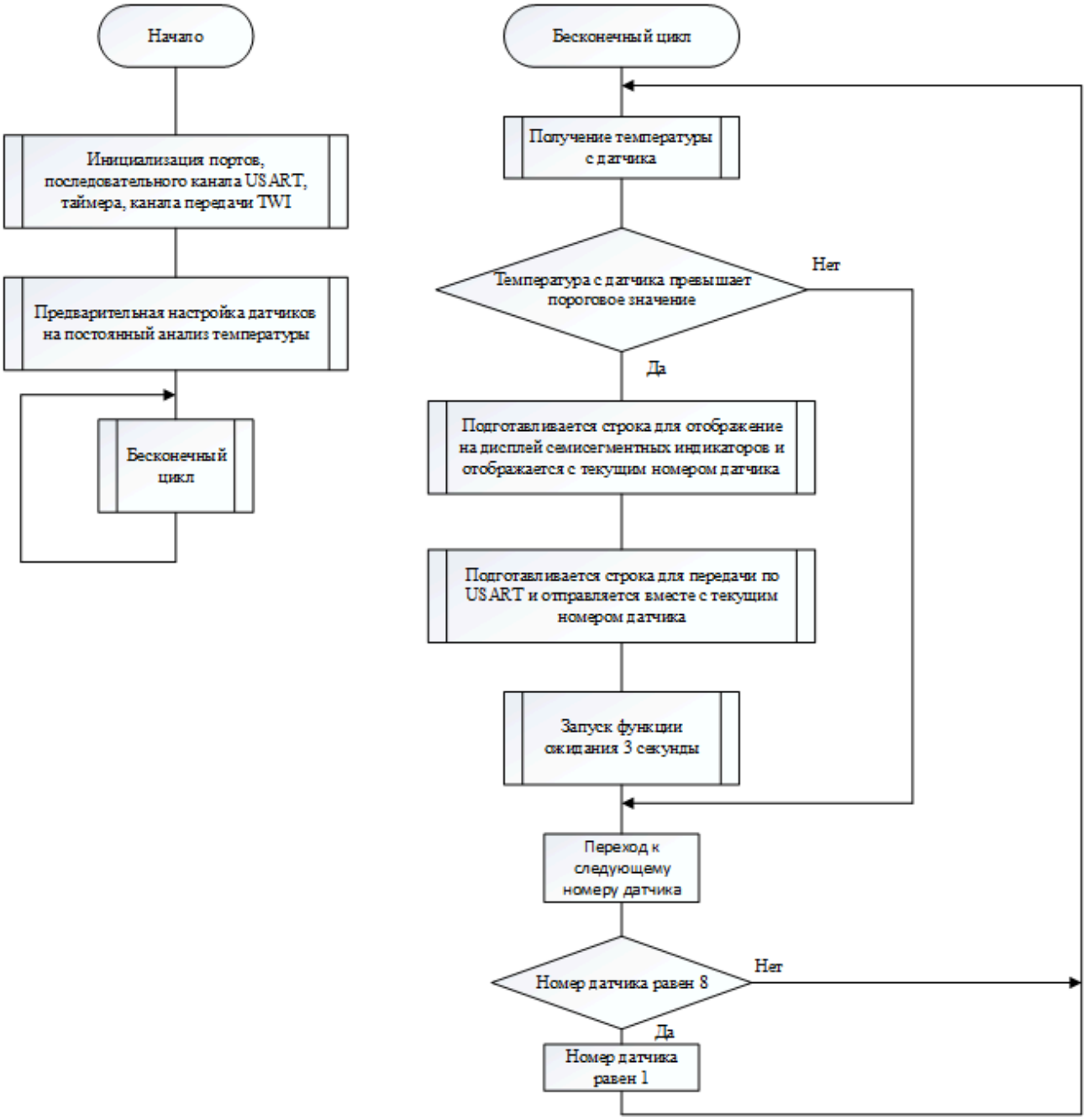


Рисунок 3 – Упрощенная схема алгоритма работы программы

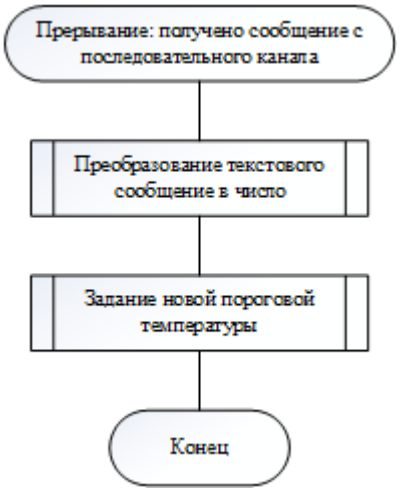


Рисунок 4 – Упрощенная схема алгоритма получение сообщения по последовательному каналу

Предусмотрена возможность изменения пороговой температуры с помощью компьютера по последовательному каналу USART. С последовательного канала принимается четыре символа, если один из них — клавиша «Ввод», то до этого набранные символы распознаются и если получается число — заменяют пороговую температуру.

3.5 Реализация последовательного канала USART

Микроконтроллеры ATmega8 имеют в своём составе модуль полнодуплексного универсального синхронно/асинхронного приемопередатчика USART. Через него осуществляется прием и передача информации, представленной последовательным кодом, поэтому модуль USART часто называют также последовательным портом. С помощью этого модуля микроконтроллер может обмениваться данными с различными внешними устройствами.

Поток данных, передаваемых по каналу UART, представляет собой совокупность посылок или кадров. Каждый кадр содержит стартовый бит, восемь или девять битов данных и стоповый бит. Стартовый бит имеет уровень логического 0, стоповый – уровень логической 1. Скорость передачи данных может варьироваться в широких пределах, причем высокие скорости

передачи могут быть достигнуты даже при относительно низкой тактовой частоте микроконтроллера.

Для взаимодействия с программой в модуле предусмотрены прерывания при наступлении следующих событий: «прием завершен» с адресом вектора $009 в таблице векторов прерываний, «регистр данных передатчика пуст» с адресом вектора $00A, «передача завершена» с адресом вектора $00B.

Принимаемые и передаваемые данные (8 разрядов) хранятся в регистре UDR. Физически регистр UDR состоит из двух отдельных регистров, один из которых используется для передачи данных, другой – для приема. При чтении регистра UDR выполняется обращение к регистру приёмника, при записи – к регистру передатчика.

3.6 Реализация последовательной шины I2C

В таких микроконтроллерах как ATmega8 есть встроенный контроллер этого двухпроводного последовательного интерфейса — TWI.

Протокол TWI позволяет проектировщику системы внешне связать до 128 различных устройств через одну двухпроводную двунаправленную шину, где одна линия - линия синхронизации SCL и одна - линия данных SDA. В качестве внешних аппаратных компонентов, которые требуются для реализации шины, необходимы только подтягивающий к плюсу питания резистор на каждой линии шины. Все устройства, которые подключены к шине, имеют свой индивидуальный адрес, а механизм определения содержимого шины поддерживается протоколом TWI.

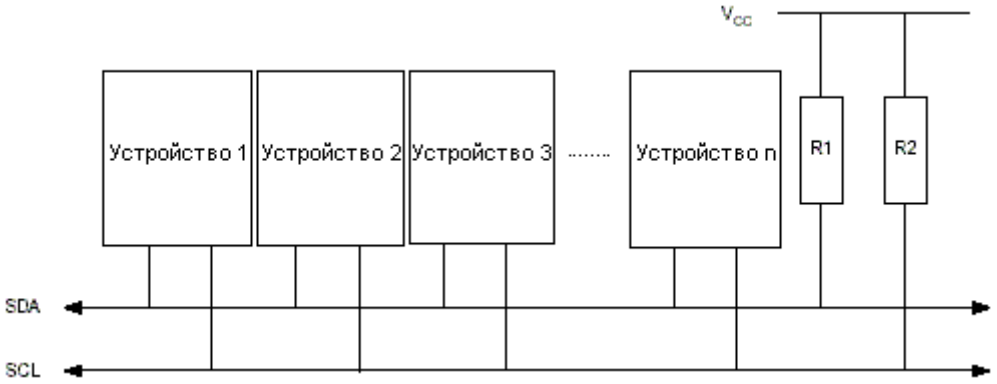


Рисунок 5 – Внешнее подключение к шине TWI

Обе линии шины подключены к положительной шине питания через подтягивающие резисторы. Среди всех совместимых с TWI устройствами в качестве драйверов шины используются транзистор или с открытым стоком или с открытым коллектором. Этим реализована функция монтажного И, которая очень важна для двунаправленной работы интерфейса. Низкий логический уровень на линии шины TWI генерируется, если одно или более из TWI-устройств выводит лог. 0. Высокий уровень на линии присутствует, если все TWI-устройства перешли в третье высокоимпедансное состояние, позволяя подтягивающим резисторам задать уровень лог. 1.

Обычно шина TWI работает на скоростях 100кГц или 400кГц.

При работе шины всегда есть ведущее и ведомые устройства. Ведущее устройство инициирует и заканчивает передачу данных. Передача инициируется, когда ведущий формирует условие СТАРТа на шине, и прекращается, когда ведущий формирует на шине условие ОСТАНОВа. Между условиями СТАРТа и ОСТАНОВа шина считается занятой и в этом случае ни какой другой мастер не может осуществлять управляющие воздействия на шине. Существуют особые случаи, когда новое условие СТАРТа возникает между условиями СТАРТа и ОСТАНОВа. Данный случай именуется как условие "Повторного старта" и используется при необходимости инициировать мастером новый сеанс связи, не теряя при этом управление шиной. После "Повторного старта" шина считается занятой до следующего ОСТАНОВа. Это идентично поведению после СТАРТа, следовательно, при описании ссылка на

условие СТАРТа распространяется и на "Повторный старт", если, конечно же, нет специального примечания.

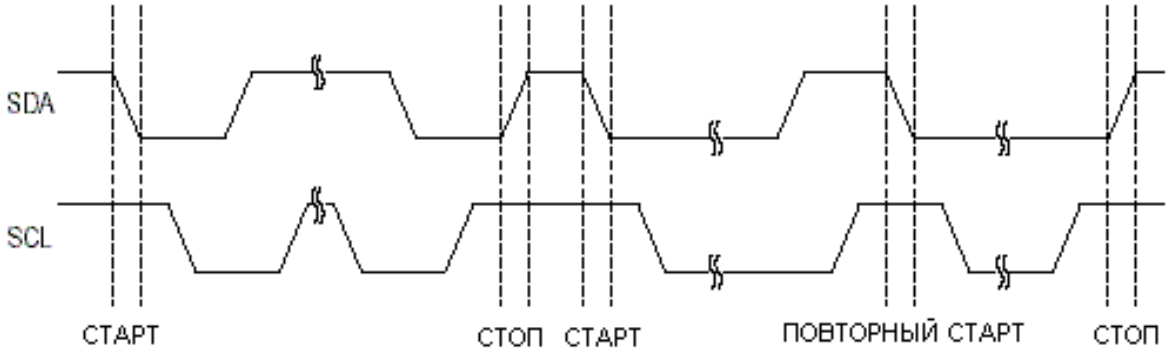


Рисунок 6 – Условия СТАРТа, ПОВТОРНОГО СТАРТА и ОСТАНОВа

Все передаваемые адресные пакеты по шине TWI состоят из 9 бит, в т.ч. 7 бит адреса, один бит управления для задания типа операции ЧТЕНИЕ/ЗАПИСЬ и один бит подтверждения. Если бит ЧТЕНИЕ/ЗАПИСЬ = 1, то будет выполнена операция чтения, иначе - запись. Если подчиненный распознает, что к нему происходит адресация, то он должен сформировать низкий уровень на линии SDA на 9-ом цикле SCL (формирование бита подтверждения). Если адресуемое подчиненное устройство занято или по каким-либо другим причинам не может обслужить ведущее устройство, то на линии SDA необходимо оставить высокий уровень во время цикла подтверждения.

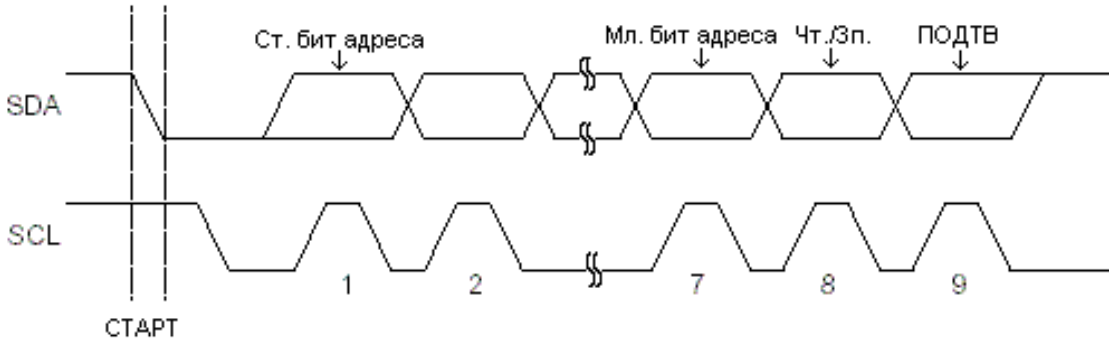


Рисунок 7 – Формат адресного пакета

Сеанс связи обычно состоит из условия СТАРТа, ПОДЧИН_АДР+ЧТЕНИЕ/ЗАПИСЬ, одного или более пакетов данных и условия ОСТАНОВа.

Для расчеты скорости работы можно использовать формулу:

$$f_{SCL} = \frac{f_{ЦПУ}}{16 + 2 \cdot (TWBR) \cdot 4^{TWPS}},$$

TWBR - значение регистра скорости TWI;

TWPS - значение бит предделителя в регистре состояния TWI.

TWI может работать в одном из 4-х режимов работы. Они называются: ведущий передатчик (MT), ведущий приемник (MR), подчиненный передатчик (ST) и подчиненный приемник (SR). Некоторые из этих режимов могут использоваться в рамках одного и того же приложения.

В данной работе используется 2 основных состояния TWI микроконтроллера: ведущий передатчик и ведущий приемник.

Таблица 1 – Коды состояния в режиме ведущего передатчика

Код состояния (TWSR), биты предделителя равны 0	Состояние двухпроводной последовательной шины и схемы двухпроводного последовательного интерфейса	Программные действия					Следующее действие, выполняемое схемой TWI
		В/из TWDR	В TWCR				
			STA	STO	TWINT	TWEA	
$08	Передано условие СТАРТ	Загрузка ПОДЧИН_АДР+ЗАПИСЬ	0	0	1	x	Передается ПОДЧИН_АДР + ЗАПИСЬ Принимается ПОДТВ. или НЕТ ПОДТВ.
$10	Передано условие ПОВТОРНЫЙ СТАРТ	Загрузка ПОДЧИН_АДР+ЗАПИСЬ	0	0	1	X	Передается ПОДЧИН_АДР + ЗАПИСЬ; Принимается ПОДТВ или НЕТ ПОДТВ
		или ПОДЧИН_АДР+ЧТЕНИЕ	0	0	1	X	Передается ПОДЧИН_АДР + ЧТЕНИЕ; Переход на режим ведущего приемника
$18	Передано ПОДЧИН_АДР+ЗАПИСЬ и принято ПОДТВерждение	Загрузка байта данных или	0	0	1	X	Передается байт данных, принимается или не принимается ПОДТВерждение
		действия без загрузки TWDR или	1	0	1	X	Передается ПОВТОРНЫЙ СТАРТ
		действия без загрузки TWDR или	0	1	1	X	Передается условие СТОП и сбрасывается флаг TWSTO
		действия без загрузки TWDR	1	1	1	X	Вслед за условием СТАРТ передается условие СТОП и сбрасывается флаг TWSTO
$20	Передано ПОДЧИН_АДР+ЗАПИСЬ и принято НЕТ ПОДТВ	Загрузка байта данных или	0	0	1	X	Передается байт данных, принимается или не принимается ПОДТВерждение
		действия без загрузки TWDR или	1	0	1	X	Передается ПОВТОРНЫЙ СТАРТ
		действия без загрузки TWDR или	0	1	1	X	Передается условие СТОП и сбрасывается флаг TWSTO
		действия без загрузки TWDR	1	1	1	X	Вслед за условием СТАРТ передается условие СТОП и сбрасывается флаг TWSTO
$28	Передается байт данных; принимается ПОДТВерждение	Загрузка байта данных или	0	0	1	X	Передается байт данных, принимается или не принимается ПОДТВерждение
		действия без загрузки TWDR или	1	0	1	X	Передается ПОВТОРНЫЙ СТАРТ
		действия без загрузки TWDR или	0	1	1	X	Передается условие СТОП и сбрасывается флаг TWSTO
		действия без загрузки TWDR	1	1	1	X	Вслед за условием СТАРТ передается условие СТОП и сбрасывается флаг TWSTO
$30	Передается байт данных; принимается НЕТ ПОДТВерждения	Загрузка байта данных или	0	0	1	X	Передается байт данных, принимается или не принимается ПОДТВерждение
		действия без загрузки TWDR или	1	0	1	X	Передается ПОВТОРНЫЙ СТАРТ
		действия без загрузки TWDR или	0	1	1	X	Передается условие СТОП и сбрасывается флаг TWSTO
		действия без загрузки TWDR	1	1	1	X	Вслед за условием СТАРТ передается условие СТОП и сбрасывается флаг TWSTO

Таблица 2 – Коды состояния в режиме ведущего приемника

Код состояния (TWSR), биты предделителя равны 0	Состояние двухпроводной последовательной шины и схемы двухпроводного последовательного интерфейса	Программные действия					Следующее действие, выполняемое схемой TWI
		В/из TWDR	В TWCR				
			STA	STO	TWINT	TWEA	
$08	Передано условие СТАРТ	Загрузка ПОДЧИН_АДР+ЧТЕНИЕ	0	0	1	x	Передается ПОДЧИН_АДР + ЧТЕНИЕ Принимается ПОДТВ. или НЕТ ПОДТВ.
$10	Передано условие ПОВТОРНЫЙ СТАРТ	Загрузка ПОДЧИН_АДР+ЧТЕНИЕ	0	0	1	X	Передается ПОДЧИН_АДР+ЧТЕНИЕ, принимается ПОДТВ. или НЕТ ПОДТВ.
		или ПОДЧИН_АДР+ЗАПИСЬ	0	0	1	X	Передается ПОДЧИН_АДР+ЗАПИСЬ, переключение на режим «Ведущий передатчик»
$38	Потеря арбитрации во время передачи бит ПОДЧИН_АДР + ЧТЕНИЕ или бита НЕТ ПОДТВ.	Действия без загрузки TWDR или	0	0	1	X	Шина освобождается и вводится безадресный подчиненный режим
		действия без загрузки TWDR	1	0	1	X	Условие СТАРТ передается после освобождения шины
$40	Передано ПОДЧИН_АДР+ЧТЕНИЕ и принято ПОДТВ	Действия без загрузки TWDR или	0	0	1	0	Принимается байт данных, возвращается бит НЕТ_ПОДТВ.
		действия без загрузки TWDR	0	0	1	1	Принимается байт данных, возвращается бит ПОДТВ.
$48	Передано ПОДЧИН_АДР+ЧТЕНИЕ и принято НЕТ ПОДТВ	Действия без загрузки TWDR или	1	0	1	X	Передается ПОВТОРНЫЙ СТАРТ
		действия без загрузки TWDR или	0	1	1	X	Передается условие СТОП и сбрасывается флаг TWSTO
		действия без загрузки TWDR	1	1	1	X	Вслед за условием СТАРТ передается условие СТОП и сбрасывается флаг TWSTO
$50	Принят байт данных и возвращается ПОДТВерждение	Чтение байта данных или	0	0	1	0	Принимается байт данных и возвращается НЕТ ПОДТВ
		чтение байта данных	0	0	1	1	Принимается байт данных и возвращается ПОДТВ
$58	Принят байт данных и возвращается НЕТ ПОДТВерждения	Чтение байта данных или	1	0	1	X	Передается ПОВТОРНЫЙ СТАРТ
		чтение байта данных или	0	1	1	X	Передается СТОП и сбрасывается флаг TWSTO
		чтение байта данных	1	1	1	X	Вслед за условием СТАРТ передается условие СТОП и сбрасывается флаг TWSTO

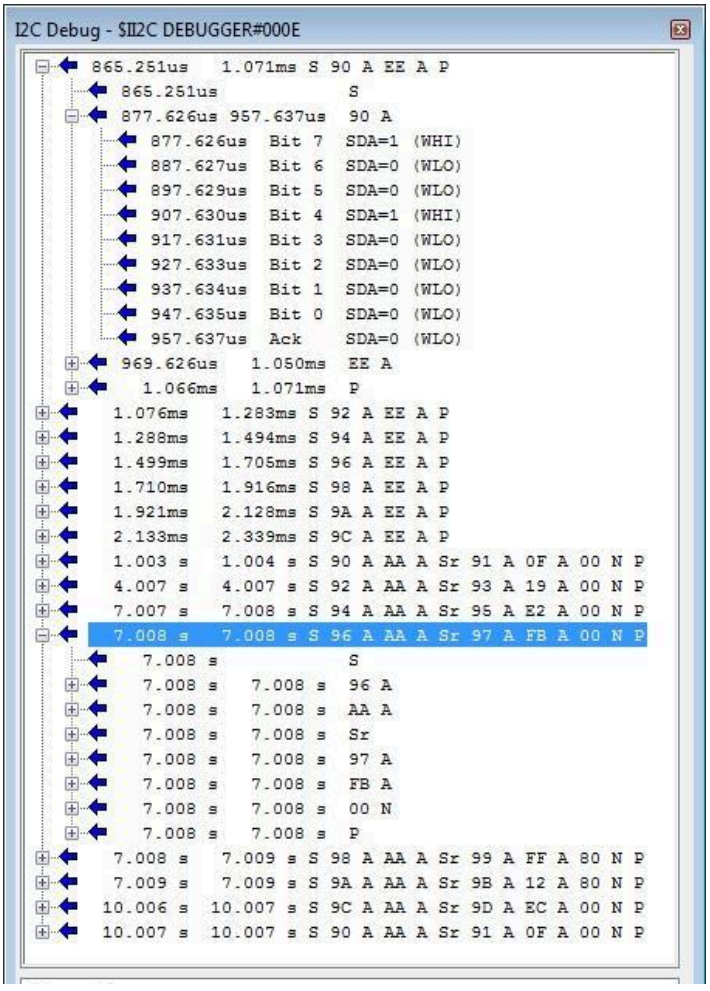


Рисунок 8 – Передача данных по шине TWI

4 Расчет потребляемой мощности

Потребляемая мощность состоит из нескольких частей: свечения светодиодов семисегментного индикатора, мощности, выделяемой на резисторах и проводящих элементах, мощности, потребляемой микроконтроллером и датчиками температур, переходником FT232rl.

Максимальная мощность семисегментного индикатора:

40диодов * 20 мА * 2.5В = 2Вт

Мощность резисторов:

8 резисторов * 20мА * 2,5В + 2 резистора * 5мкА * 5В = 0.4Вт 0.005 Вт = 0.4005Вт

Мощность датчиков:

7 Датчиков * 5В *1.4мА = 0,049Вт

Мощность преобразователя

0.015А*5В = 0.075Вт

Мощность микроконтроллера:

5В * 0.012А = 0.06В.

С поправкой на различного рода потери в 10%, потребляемая макетом мощность такова:

2,58 Вт.

Данные о потребляемые токах и напряжениях, мощностях взяты из спецификации (datasheet) на эти радиоэлементы.

Фактическая потребляемая мощность макета составляет приблизительно 2.58 Вт в режиме пользования.

5 Отладка программы микроконтроллера и тестирование

Начальные значения температур:

Датчик 1: 15.0

Датчик 2: 25.0

Датчик 3: -30.0

Датчик 4: -5.0

Датчик 5: -0.5

Датчик 6: 18.5

Датчик 7: -20.0

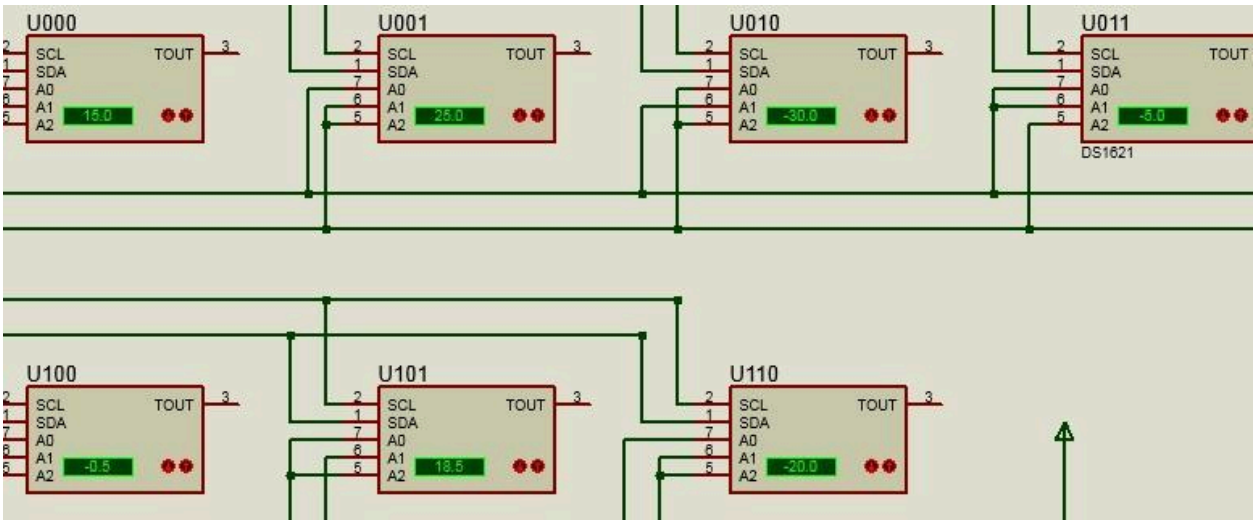


Рисунок 9 – Температура датчиков

Начальное значение пороговой температуры при включении устройства -20 градусов.

Отображение информации на семисегментном дисплее:

Рисунок 10 – Отображение температуры датчика 1 на дисплее

Рисунок 11 – Отображение температуры датчика 2 на дисплее

Рисунок 12 – Отображение температуры датчика 6 на дисплее

Рисунок 13 – Отображение информации, переданной по последовательному каналу

Так как пороговая температура = -20, показания датчиков №3,4,5,7 не отображаются. Уменьшим пороговую температуру для отображение информации со всех датчиков.

Рисунок 14 – Отображение температуры датчика 1 на дисплее

Рисунок 15 – Отображение температуры датчика 2 на дисплее

Рисунок 16 – Отображение температуры датчика 3 на дисплее

Рисунок 17 – Отображение температуры датчика 4 на дисплее

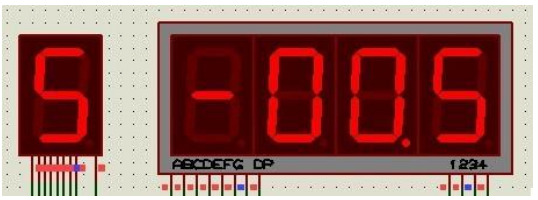


Рисунок 18 – Отображение температуры датчика 5 на дисплее

Рисунок 19 – Отображение температуры датчика 6 на дисплее

Рисунок 20 – Отображение температуры датчика 7 на дисплее

Рисунок 21 – Отображение информации, переданной по последовательному каналу

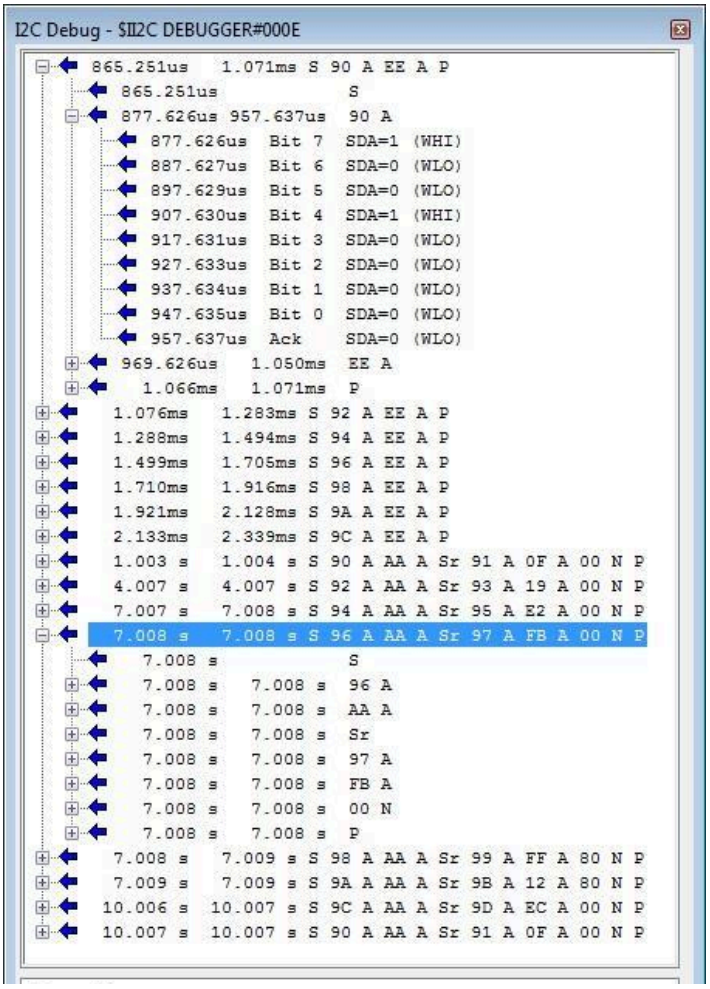


Рисунок 22 – Передача данных по шине TWI

В первую секунду идет настройка всех семи датчиков:

Поочередно ведущий микроконтроллер подает на шину команду СТАРТ, адрес датчика (например 90h или 10010000b), причем последний бит — 0, что соответствует запись в датчик. Передается команда EEh, означающая непрерывный анализ температуры. После ответа с протоколом «ACK», ведущий передает СТОП.

Таким образом происходит конфигурация всех семи датчиков.

После первой секунды ведущий микроконтроллер опрашивает все датчики по очереди.

Подает команду СТАРТ, указывает адрес датчика в режиме записи (96h), передает команду AAh, означающую передачу результатов. Воспроизводит ПОВТОРНЫЙ СТАРТ, указывает адрес датчика в режиме чтения (97h) с последним битом = 1 (чтение), получает первый байт информации, передает

сигнал «ACK», получает второй байт информации о дробной части температуры, передает «NAK», для досрочного завершения передачи. Далее ведущий передает СТОП.

Таким образом опрашиваются все 7 датчиков.

6 Программирование микроконтроллера

При программировании микроконтроллера полученный в результате компиляции программы машинный код загружается в память программ, а требуемые данные заносятся в ЭСППЗУ (EEPROM память). Подавляющее большинство микроконтроллеров семейства поддерживает 2 режима программирования:

1. режим параллельного программирования при высоком напряжении;
2. режим программирования по последовательному каналу.

Под «высоким» напряжением здесь понимается управляющее напряжение (12В), подаваемое на вывод RESET микроконтроллера для перевода последнего в режим программирования. При этом независимо от режима программирование FLASH - и EEPROM памяти осуществляется всегда побайтно.

В процессе программирования могут выполняться следующие операции:

- стирание кристалла (Chip erase);
- чтение/запись Flash-памяти программ;
- чтение/запись EEPROM памяти данных;
- чтение/запись конфигурационных ячеек;
- чтение/запись ячеек защиты;
- чтение ячеек идентификатора.

Все модели микроконтроллеров поставляются со стертой памятью программ и памятью данных (во всех ячейках находится число $FF) и пригодны к немедленному программированию.

Микроконтроллерами AVR поддерживаются два режима последовательного программирования. При этом первый — режим последовательного программирования при высоком напряжении — поддерживается только моделями AT90S/LS2323 и AT90S/LS2343 и является аналогом режима параллельного программирования остальных моделей.

Второй же режим, называемый также режимом программирования по последовательному каналу, поддерживается всеми моделями семейства.

Рассмотрим более подробно режим программирования по последовательному каналу.

В этом режиме программирование памяти программ и данных осуществляется через последовательный интерфейс SPI. Данный режим используется, как правило, для программирования (перепрограммирования) микроконтроллера непосредственно в системе (ISP, In System Programming).

Наличие этого режима является одним из важнейших достоинств микроконтроллеров AVR, т.к. он позволяет значительно упростить и удешевить модернизацию программного обеспечения.

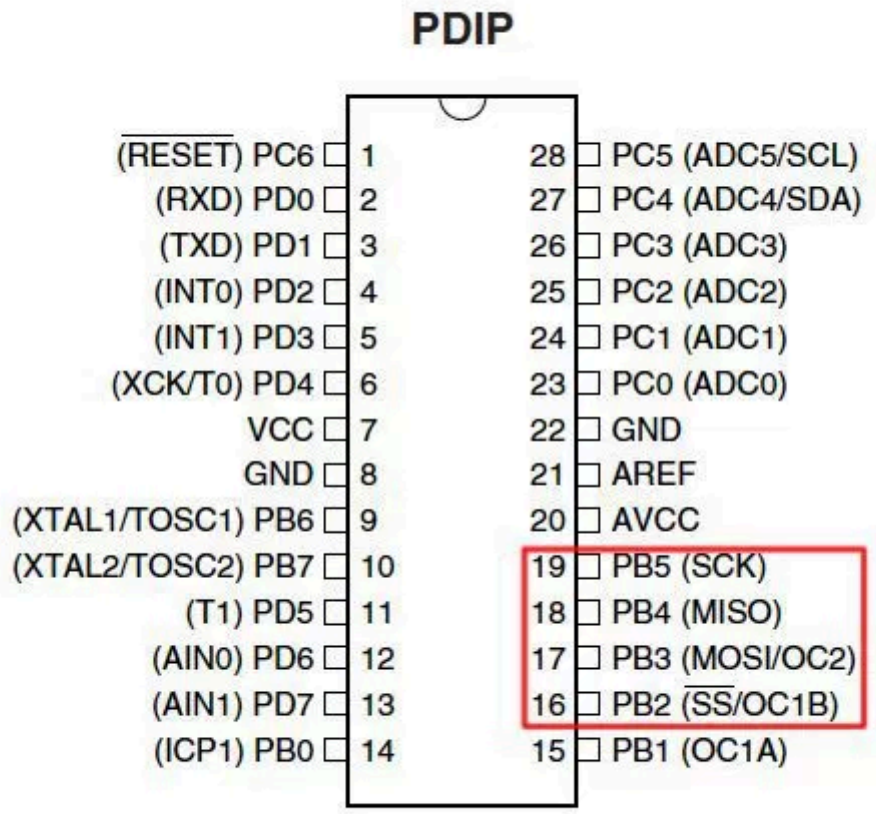


Рисунок 23 – Выводы, необходимые для SPI программирования

Также необходимо соединить питание и землю программатора.

Программирование осуществляется путем посылки 4-байтовых команд на вывод MOSI микроконтроллера. Результат выполнения команд чтения снимается с вывода MISO микроконтроллера. Передача команд и вывод

результатов их выполнения осуществляется от старшего разряда к младшему. При этом «защелкивание» входных данных выполняется по нарастающему фронту сигнала SCK, а «защелкивание» выходных данных — по спадающему.

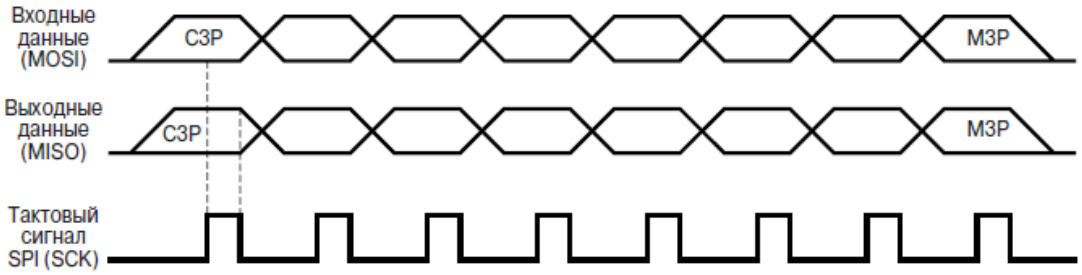


Рисунок 24 – Временные диаграммы сигналов при программировании по последовательному каналу

При программировании по последовательному каналу микроконтроллеру требуется источник тактового сигнала. Им может быть внешняя схема, выход которой подключен к XTAL1 микроконтроллера, кварцевый резонатор или внутренний RC-генератор (используется в данной работе).

Таблица 3 – Команды режима программирования по последовательному каналу

Команда		Формат команды				Описание команды
		1-й байт	2-й байт	3-й байт	4-й байт	
Разрешение программирования		1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Разрешает программирование микроконтроллера, пока на выводе RESET присутствует сигнал НИЗКОГО уровня
Стирание кристалла		1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Очистка содержимого Flash- и EEPROM-памяти. После посылки команды необходимо: 1. Выдержать паузу длительностью t_{WD_ERASE}. 2. Подать на вывод RESET положительный импульс. 3. Выждать не менее 20 мс. 4. Послать команду Разрешение программирования.
Чтение FLASH-памяти		0010 **H**000	xxxx **aaaa**	**bbbb bbbb**	**oooo oooo**	Чтение младшего (**H** = 0) или старшего (**H** = 1) байта памяти программ (**o**), расположенного по адресу **a:b**
Запись FLASH-памяти		0100 **H**000	xxxx **aaaa**	**bbbb bbbb**	**iiii iiii**	Запись младшего (**H** = 0) или старшего (**H** = 1) байта (**i**) в память программ по адресу **a:b**
Чтение EEPROM-памяти		1010 0000	xxxx xxx**a**	**bbbb bbbb**	**oooo oooo**	Чтение содержимого ячейки (o) EEPROM-памяти по адресу a:b
Запись EEPROM-памяти		1100 0000	xxxx xxx**a**	**bbbb bbbb**	**iiii iiii**	Запись значения (i) в ячейку EEPROM-памяти по адресу a:b
Чтение конфигурац. ячеек и ячеек защиты	AT90S/LS2323	0101 1000	xxxx xxxx	xxxx xxxx	***12S***x xxx**F**	Чтение состояния ячеек защиты LB1, LB2 и конфигурационных ячеек
	AT90S/LS2343	0101 1000	xxxx xxxx	xxxx xxxx	***12S***x xxx**R**	
Чтение конфигурац. ячеек (AT90S/LS2333, AT90S/LS4433)		0101 0000	xxxx xxxx	xxxx xxxx	xx***87*** **6543**	Чтение состояния конфигурационных ячеек (см. примечание к таблице)
Запись ячеек защиты		1010 1100	1111 1***21***	xxxx xxxx	xxxx xxxx	Запись ячеек защиты LB1 и LB2. Для программирования ячейки соответствующий разряд 2-го байта должен быть сброшен

Команда		Формат команды				Описание команды
		1-й байт	2-й байт	3-й байт	4-й байт	
Запись конфигурационных ячеек	AT90S/LS2323	1010 1100	1011 111**F**	xxxx xxxx	xxxx xxxx	Запись конфигурационных ячеек. Соответствие разрядов 2-го байта команды конкретным ячейкам указано в примечании к таблице
	AT90S/LS4434					
	AT90S/LS8535					
	AT90S/LS2343	1010 1100	1011 111**R**	xxxx xxxx	xxxx xxxx	
	AT90S/LS2333	1010 1100	101**7** **6543**	xxxx xxxx	xxxx xxxx	
	AT90S/LS4433					
Чтение идентификатора		0011 0000	xxxx xxxx	xxxx xx**bb**	**oooo oooo**	Чтение ячейки идентификатора (**o**) с адресом **b** (только в режимах защиты №1 и №2, см. **Табл. 11.1**)

Расшифровка условных обозначений, используемых в таблице:

a — разряды старшего байта адреса;

b — разряды младшего байта адреса;

i — посылаемые в микроконтроллер данные;

o — считываемые из микроконтроллера данные;

x — состояние разряда безразлично;

1, ***2*** — ячейки защиты LB1 и LB2 соответственно;

F — конфигурационная ячейка FSTRT;

R — конфигурационная ячейка RCEN;

S — конфигурационная ячейка SPIEN;

3 — конфигурационная ячейка CKSEL0;

4 — конфигурационная ячейка CKSEL1;

5 — конфигурационная ячейка CKSEL2;

6 — конфигурационная ячейка BODEN;

7 — конфигурационная ячейка BODLEVEL.

Для перевода микроконтроллера в режим программирования по последовательному каналу необходимо выполнить следующие действия:

1 Подать на микроконтроллер напряжение питания, при этом на выводах SCK и RESET должен присутствовать сигнал низгого уровня. В некоторых случаях (если программатор не гарантирует установку сигнала SCK в «0» при подаче питания) после установки сигнала SCK в «0» необходимо подать на вывод RESET положительный импульс длительностью не менее двух периодов тактового сигнала микроконтроллера.

2 Если тактирование микроконтроллера осуществляется от внешней схемы, подать тактовый сигнал на вывод XTAL1.

3 Выждать не менее 20 мс.

4 Послать на вывод MOSI команду «Разрешение программирования» (Programming enable).

Для контроля прохождения команды при посылке 3-го байта возвращается значение 2-го байта ($53). Если возвращаемое значение отлично от указанного, необходимо подать на вывод SCK положительный импульс и снова послать команду «Разрешение программирования» (Programming enable). Причем необходимо передавать все 4 байта команды. Отсутствие возврата числа $53 после 32 попыток указывает на отсутствие связи между программатором и микросхемой либо на неисправность микросхемы.

После завершения программирования на вывод можно подать напряжение высокого уровня для перевода микроконтроллера в рабочий режим.

Заключение

В результате выполнения курсовой работы был разработан контроллер, измеряющий температуру при помощи семи датчиков DS1621. Температура с датчиков, превышающих пороговую, отображается на семисегментном индикаторе, предусмотрена возможность изменение порогов температуры.

Произведены проектирование функционального устройства, построенного с использованием микроконтроллера, и разработка необходимой документации на объект разработки. В процессе выполнения курсового проекта были решены следующие задачи: анализ объекта разработки на функциональном уровне, разработка функциональной схемы, выбор элементной базы для реализации объекта и поиск схемотехнических решений, синтез принципиальной электрической схемы, тестирование в системе автоматизированного проектирования Proteus, сборка устройства и тестирование на макетной плате.

Список использованных источников

1. В.Я. Хартов Микроконтроллеры AVR. Практикум для начинающих. 2-е издание, Издательство МГТУ им. Баумана, 2012г.
2. В.Я. Хартов Микропроцессорные системы. 2-е издание, Академия, М., 2014 г.
3. В.Я. Хартов Проектирование и отладка программ для микроконтроллеров AVR фирмы ATMEL. Изд-во МГТУ им. Н.Э.Баумана, 2004 г.
4. Б.Ю. Семенов Шина I2C в радиотехнических конструкциях., "СОЛОН-Р", 2002г.
5. Баранов В.Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы. М., “Додэка – XX1”, 2004г.
6. М.С. Голубцов Микроконтроллеры AVR: от простого к сложному. М., “СОЛОН-Пресс”, 2003г.
7. Маслов В. Методические указания по оформлению РПЗ.
8. ГОСТ 2.743-91 Обозначения условные в графических схемах. Элементы цифровой техники.
9. ГОСТ 2.701-84 Правила выполнения схем.
10. ГОСТ 2.702-75 Правила выполнения электрических схем.
11. ГОСТ 2.702-2011 Правила выполнения электрических схем.
12. Подавление дребезга контактов [Электронный ресурс]. – URL: http://www.labfor.ru/guidance/digital-leso2/debounce (дата обращения 10.11.2016).
13. ГОСТ 2.751-73 Обозначения условные графические в схемах
14. Описание, спецификация datasheet микроконтроллера ATMEL ATmega8.
15. Описание, спецификация datasheet переходника FTDI Chip FT232rl.
16. Описание, спецификация datasheet четырехразрядного семисегментного индикатора Kingbright CC56-12SRWA.
17. Описание, спецификация datasheet одноразрядного семисегментного индикатора Kingbright SC56-11.
18. Описание, спецификация datasheet стабилизатора напряжения L7805.

Приложение 1 –

Текст исходной программы

```
#include <mega8.h>
#include <delay.h>
#include <stdlib.h>

//биты USART
#define RXCIE 7
#define TXCIE 6
#define RXEN 4
#define TXEN 3
#define TXC 6 // флаг регистра UCSRA, устанавливающийся в 1 при завершении передачи
#define UDRE 5 // флаг регистра UCSRA, устанавливающийся в 1, когда регистр данных
пуст

//биты TWI (I2C)
#define TWINT 7
#define TWEA 6
#define TWSTA 5
#define TWSTO 4
#define TWEN 2

//биты таймера1
#define OCIE1A 4 //бит для разрешения прерывания по совпадению
#define WGM12 3  //бит для сброса счетного регистра при совпадении
#define CS10 0   //два бита настройки делителя
#define CS12 2


char porog_temp = 5; //пороговая температура.
unsigned digit[11] = {0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110,
0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01101111, 0b01000000}; //массив с
числами для семисегментного индикатора, 11 элемент - знак "-"
unsigned razr[5] = {0b11111000, 0b11110100, 0b11101100, 0b11011100, 0b10111100};
//массив с номерами разрядов семисегментного индикатора


unsigned dat_adr[7] = {0b10010000, 0b10010010, 0b10010100, 0b10010110, 0b10011000,
0b10011010, 0b10011100}; //адреса датчиков для i2c интерфейса с сдвинутым влево на 1
разряд (т.к. в i2c нулевой бит - режим чтенияя или записи)
char dat_temp1; //переменная, в которой хранится целая часить температуры текущего
датчика
char dat_temp2; //переменная, в которой хранится дробная часить температуры текущего
датчика
unsigned char message[5]; //массив для печати на семисегментный дисплей
unsigned char fl;//флаг для выхода из режима печати и переключение датчика

void decode(); //функция преобразования двух шестнадцатиричных чисел в строку из пяти
элементов, содержащих биты цифр десятичной системы для семисегментного дислпея
unsigned char dat_num = 1; //номертекущегодатчика

void reset() //функция броса всех настроек к стандартным и подготовки к началу работы
{
      dat_num = 7;

}
```

```
unsigned char usart_message[8]; //сообщение, которое будет отправлено по USART на
ПЭВМ
void temp_transmit()  //функция для передачи сообщение по usart на ПЭВМ
{
	unsigned char sh = 0;
	while (sh < 8)
	{
		UDR = usart_message[sh];
		while (!(UCSRA & (1 << UDRE)))
		{
			#asm("nop");
		}
		sh++;
	}
}

//функции для запуска вычислений температуры в датчиках
void dat_init();
void dat_conf(unsigned char adr);

//функция получения результатов датчика
void get_temp(unsigned char adr);


unsigned char re_mes[4];   //сообщение, которое будет принято по USART от ПЭВМ
unsigned char us_s = 0;

interrupt [USART_RXC] void u_rec()   //прерывание по приему байта с интерфейса usart
{
	//каждый байт записывается в строку re_mes, длинной 4 символа
	//если символ = Enter данные воодятся как пороговая температура и отчет датчиков
начинается сначала
	re_mes[us_s] = UDR;
	if (re_mes[us_s] == 0x0D)
	{
		porog_temp = atoi(re_mes);
		us_s = 0;
		reset();
	}
	else
	{
		us_s++;
		if (us_s > 3)
			us_s = 0;
	}
}

interrupt [TIM1_COMPA] void timer_int()
{
	fl = 0; //при прерывании таймера T1 сбрасывается флаг, и переключается номер
отображаемого датчика.
}

void main(void)
{

	DDRB = 0xFF; //порт B на выход для работы с 8ми сегментным индикатором
	PORTB = 0x00; //начальное значение - погашен.

	DDRD = 0b11111100; //порт C на выход для работы с катодами 8ми сегментных
```

```
индикаторов
	PORTD = 0b11111100; //начальнео значение - ни один из разрядов не работает

	//настройка USART
	UBRRL = 1; //скорость 250к бод
UCSRB = (1 << RXCIE) | (1 << RXEN) | (1 << TXEN); // прерывание после завершения
приема, прием и передача разрешены.

	//настройка TWI (I2C)
	//предделители устанавливаются таким образом, чтобы частота синхросигнала
	//SCL = 100 КГц (требуемая частота работы датчика температуры DS1621 по
спецификации.)
	//F_SCL = F_CPU / ( 16 + 2 * TWBR * 4^TWPS ) - формула расчета.
	TWBR = 0x20;	//установка делителя частоты работы i2c.
	TWSR = 0x00;	//установка предделителя частоты работы i2c, начальное
состояние шины - нулевое.

	//настройка таймера T1, который будет работать в режиме "сравнения".
	OCR1AH = 0x5B; //запись значений в регистр совпадения, сначала старший байт,
потом младший
	OCR1AL = 0x8D; //значение выбиралось исходя из задержки = 3с, делителя на 1024
и частоты МК = 8МГц.
	TIMSK = (1 << OCIE1A); //включено прерывание по совпадению счетного регистра
таймера T1 канала A с регистром сравнения
	#asm("sei");
	TCCR1A = 0;
	TCCR1B = (1 << WGM12); //сброс счетного счетчика при совпадении
	//TCCR1B |= (0 << CS10) | (0 << CS12); //установка делителя = 1024  выкл




	dat_init(); //вычисление температуры датчиками
	delay_ms(1000); // время на вычисление температуры датчиками

	while (1)
	{
		unsigned char i; //счетчик перебора разрядов семисегментного дисплея
		unsigned char f2 = 0; //флаг проверки: превышает ли показания датчика
пороговое значение.
		//если f2 = 1, то значения печатаются и передаются по USART, если 0 -
переход к след.датчику
		fl = 0xFF;

		get_temp(dat_adr[dat_num-1]);



		 if ( ((dat_temp1 < 0x80) && (porog_temp < 0x80)) || ((dat_temp1 >=
0x80) && (porog_temp >= 0x80)) )
		if (dat_temp1 >= porog_temp)
			f2 = 1;
		else
			f2 = 0;
		 else
{
		if ((dat_temp1 >= 0x80) && (porog_temp < 0x80))
			f2 = 0;
			 if ((dat_temp1 < 0x80) && (porog_temp >= 0x80))
			f2 = 1;
```

```
              }


            if (f2) //если температура с датчика превышает пороговую
            {
                  decode(); //подготавливается строка для отображение на блоке из
семисегментных индикаторов
                  temp_transmit();
                  TCCR1B |= (1 << CS10) | (1 << CS12); //установка делителя = 1024
вкл
                                                      //запускается таймер,
отсчитывающий 3 секунды
                                                      //на отоображение результатов этого
датчика

                  while (f1)                                //печать в блок
семисегментного индикатора ПОРАЗРЯДНО
                  {
                        PORTB = message[i];
                        PORTD = razr[4 - i];
                        delay_ms(3);
                        PORTB = 0;
                        i++;
                        if (i > 4)
                              i = 0;
                  }
            }

            dat_num++;
            if (dat_num == 8)
                  dat_num = 1;
      }


}


void dat_conf(unsigned char adr)
{
      //start
      TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
      while (!(TWCR & (1 << TWINT) ));

      //вводим адресс датчика, режим записи и отправляем
      TWDR = adr;
      TWCR = (1 << TWINT) | (1 << TWEN);
      while (!(TWCR & (1 << TWINT) ));

      //передаем команду датчику на вычисление результатов
      TWDR = 0xEE;
      TWCR = (1 << TWINT) | (1 << TWEN);
      while (!(TWCR & (1 << TWINT) ));

      //stop
      TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
}

void dat_init()
{
      unsigned char q = 0;
      while (q < 7)
```

```
	{
		dat_conf(dat_adr[q]);
		q++;
	}
}

void get_temp(unsigned char adr)
{
	//start
	TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
	while (!(TWCR & (1 << TWINT) ));

	//вводим адресс датчика, режим записи и отправляем
	TWDR = adr;
	TWCR = (1 << TWINT) | (1 << TWEN);
	while (!(TWCR & (1 << TWINT) ));

	//передаем команду датчику на вывод результатов
	TWDR = 0xAA;
	TWCR = (1 << TWINT) | (1 << TWEN);
	while (!(TWCR & (1 << TWINT) ));

	//повторный старт
	TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
	while (!(TWCR & (1 << TWINT) ));

	//вводим адресс датчика, режим чтения и отправляем
	TWDR = (adr | 1);
	TWCR = (1 << TWINT) | (1 << TWEN);
	while (!(TWCR & (1 << TWINT) ));

	//включаем TWI, получаем первый байт температуры, посылаем ответ ACK
	TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
	while (!(TWCR & (1 << TWINT) ));
dat_temp1 = TWDR;

	//включаем TWI, получаем второй байт температуры, посылаем ответ NAK
TWCR = (1 << TWINT) | (1 << TWEN) | (0 << TWEA);
	while (!(TWCR & (1 << TWINT) ));
	dat_temp2 = TWDR;

	//stop
	TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
}

void decode()
{
	unsigned char r1, r2, r3;

	message[0] = digit[dat_num];
	usart_message[0] = dat_num + 0x30;
	usart_message[1] = ''; // пробел
	if (dat_temp1 & 0b10000000)
	{
		dat_temp1 = (dat_temp1 ^ 0b11111111) + 1;
		if (dat_temp2 == 0x80) dat_temp1--;
		message[1] = digit[10];
		usart_message[2] = '-';
	}
	else
	{
```

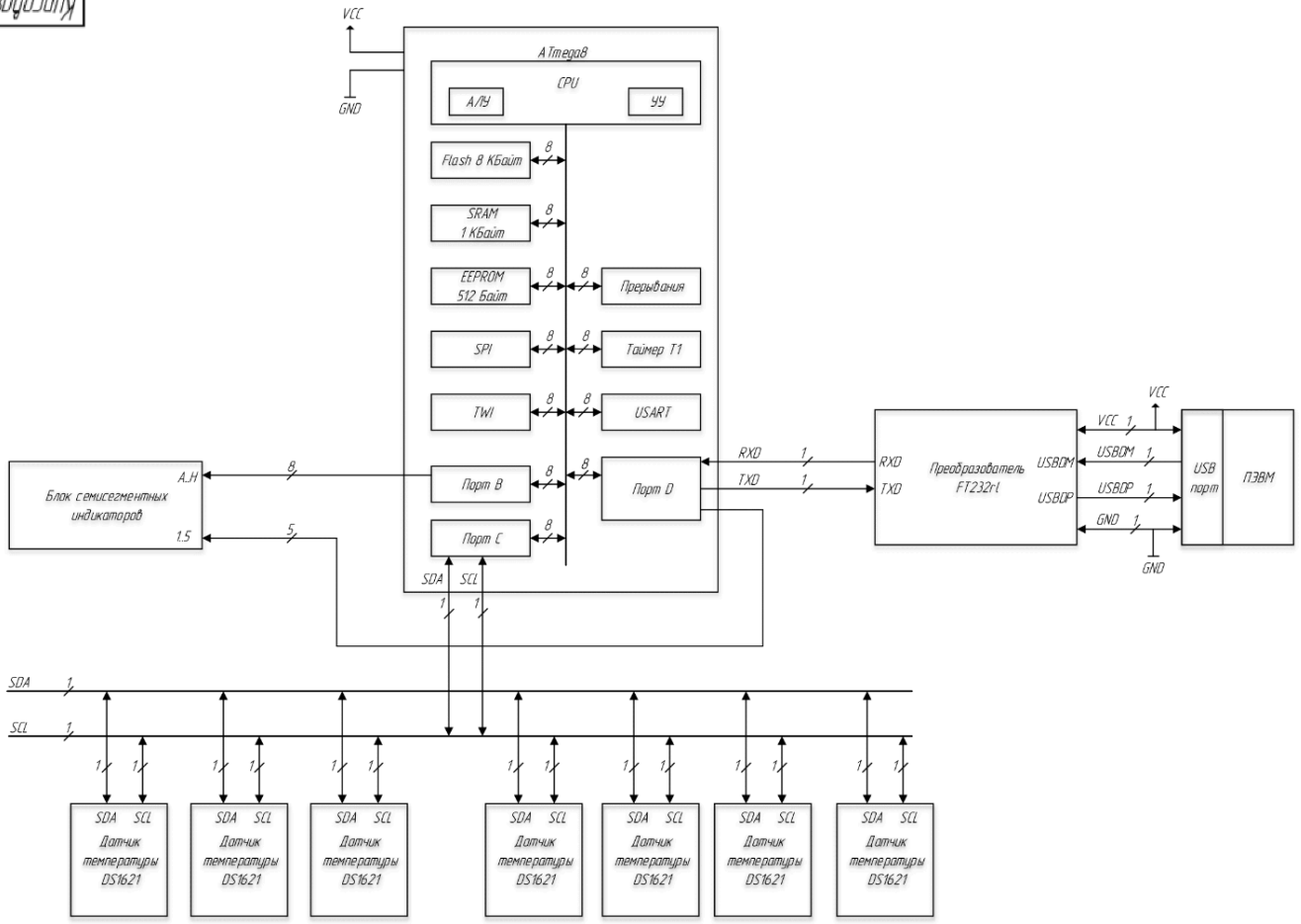
```
            r3 = dat_temp1 / 100;
message[1] = digit[r3];
      usart_message[2] = r3 + 0x30;
      }
      dat_temp1 %= 100;
      r2 = dat_temp1 / 10;
      message[2] = digit[r2];
      usart_message[3] = r2 + 0x30;

      dat_temp1 %= 10;
      r1 = dat_temp1;
      message[3] = (digit[r1] | 0b10000000);
      usart_message[4] = r1 + 0x30;
      usart_message[5] = '.';
      if (dat_temp2 == 0x80)
      {
            message[4] = digit[5];
            usart_message[6] = '5';
      }
      else
      {
            message[4] = digit[0];
            usart_message[6] = '0';
      }
      usart_message[7] = 0x0D; //переход на следующую строку
}
```

Приложение 2 –

Функциональная электрическая схема

Курсовая работа по дисциплине «Микропроцессорные системы»



					Курсовая работа по дисциплине «Микропроцессорные системы»	
Изм.	Лист	№ докум.	Подп.	Дата	Схема электрическая функциональная	Лит. / Масса / Масштаб: 1:1
Разраб.		Маронин А.Ю.				Лист 1 / Листов 1
Пров.		Хартов В.Я.				МГТУ им. Н.Э. Баумана Группа ИУ6-71
Т.контр.						
Н.контр.						
Утв.						

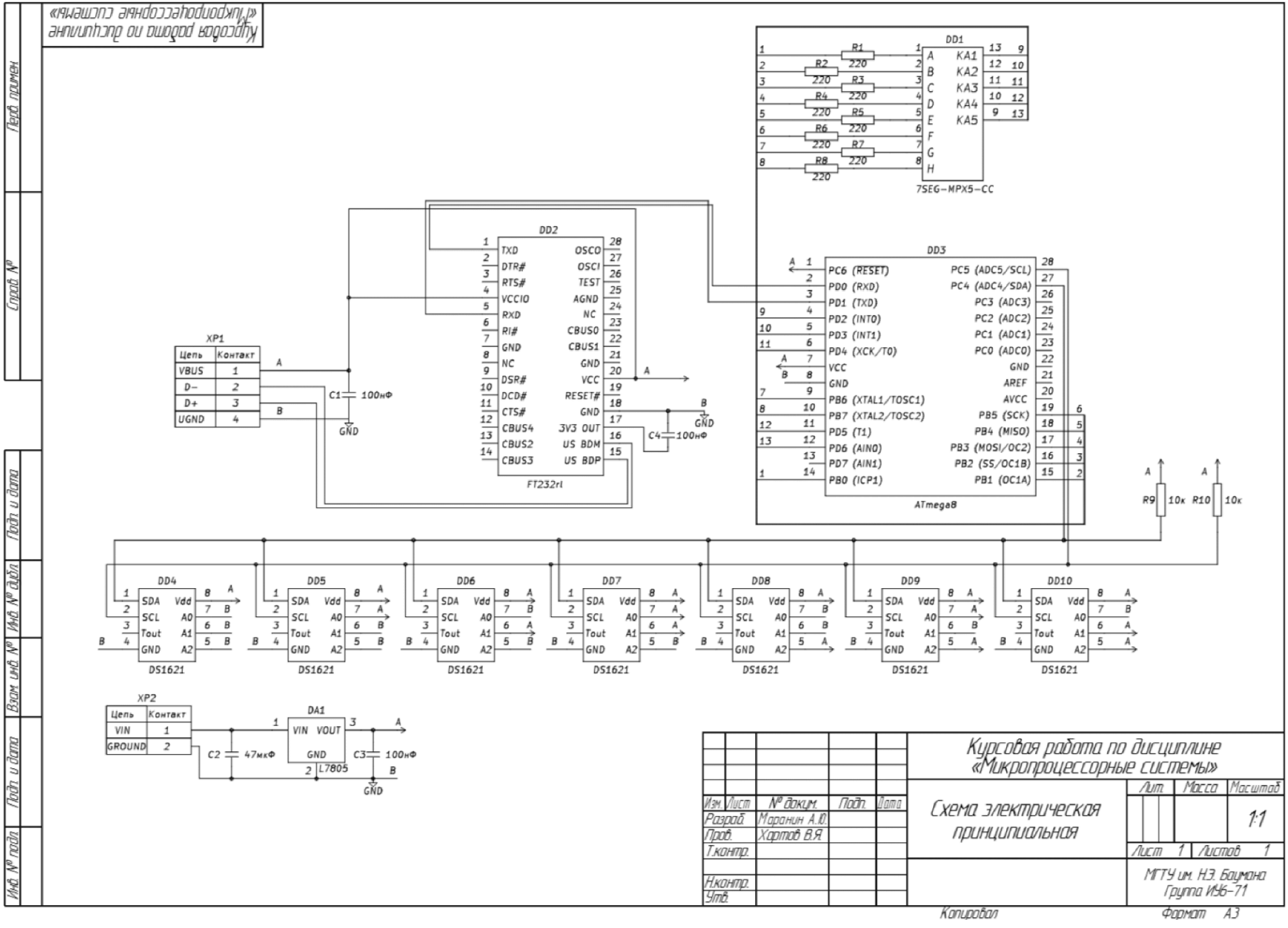
Копировал

Формат А3

Перв. примен. | Справ. № | Подп. и дата | Инв. № дубл. | Взам. инв. № | Подп. и дата | Инв. № подл.

Приложение 3 –

Принципиальная электрическая схема


Курсовая работа по дисциплине «Микропроцессорные системы»
DD1
7SEG-MPX5-CC
DD2
FT232rl
DD3
ATmega8
XP1
XP2
DA1
L7805
DD4
DD5
DD6
DD7
DD8
DD9
DD10
DS1621
C1 100нФ
C2 47мкФ
C3 100нФ
C4 100нФ
R1 220
R2 220
R3 220
R4 220
R5 220
R6 220
R7 220
R8 220
R9 10к
R10 10к
GND
Курсовая работа по дисциплине «Микропроцессорные системы»
Схема электрическая принципиальная
Маранин А.Ю.
Хартов В.Я.
МГТУ им. Н.Э. Баумана
Группа ИУ6-71
Копировал
Формат А3