Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет
им. Г.И. Носова»
Многопрофильный колледж

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ
ПРАКТИЧЕСКИХ И ЛАБОРАТОРНЫХ РАБОТ

по ПМ.02Применение микропроцессорных систем, установка и настройка периферийного оборудования
МДК.02.01 Микропроцессорные системы
для студентов специальности
09.02.01. Компьютерные системы и комплексы
базовой подготовки
Часть 1

Магнитогорск, 2020

ОДОБРЕНО:
Предметно-цикловой комиссией
«Информатики и вычислительной техники»
Председатель И.Г. Зорина
Протокол № 7 от «17» февраля 2020 г.

Методической комиссией МпК
Протокол №3 от «26» февраля 2020г

Составитель:
преподаватель МпК ФГБОУ ВО «МГТУ им. Г.И. Носова Татьяна Борисовна Ремез

Методические указания по выполнению практических и лабораторных работ разработаны на основе рабочей программы ПМ.02 «Применение микропроцессорных систем, установка и настройка периферийного оборудования»
Содержание практических и лабораторных работ ориентировано на формирование общих и профессиональных компетенций по основной профессиональной образовательной программе по специальности 09.02.01. «Компьютерные системы и комплексы» базовой подготовки: МДК.02.01 Микропроцессорные системы

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ

Важную часть теоретической и профессиональной практической подготовки студентов составляют практические и лабораторные занятия.

Состав и содержание практических и лабораторных работ направлены на реализацию действующего федерального государственного образовательного стандарта среднего профессионального образования.

Ведущей дидактической целью практических занятий является формирование практических умений - профессиональных (умений выполнять определенные действия, операции, необходимые в последующем в профессиональной деятельности), необходимых в последующей учебной деятельности по профессиональным модулям.

Ведущей дидактической целью лабораторных работ является экспериментальное подтверждение и проверка существенных теоретических положений (законов, зависимостей).

В соответствии с рабочей ПМ.02 «Применение микропроцессорных систем, установка и настройка периферийного оборудования», МДК.02.01 «Микропроцессорные системы» предусмотрено проведение лабораторных работ и практических занятий.

В результате их выполнения, обучающийся должен:

уметь:

- составлять программы на языке ассемблера для микропроцессорных систем;
- производить тестирование и отладку МПС;
- выбирать микроконтроллер/микропроцессор для конкретной системы управления

Содержание лабораторных работ ориентировано на формирование общих компетенций по профессиональному модулю основной профессиональной образовательной программы по специальности:

ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес

ОК 2. Организовывать собственную деятельность, определять методы и способы выполнения профессиональных задач, оценивать их эффективность и качество

ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность

ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития

ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности

ОК 6. Работать в коллективе и команде, эффективно общаться с коллегами, руководством, потребителями

ОК 7. Брать на себя ответственность за работу членов команды (подчиненных), результат выполнения заданий

ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации

ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности

И овладению профессиональными компетенциями:

ПК 2.1. Создавать программы на языке ассемблерадля микропроцессорных систем.

ПК 2.2. Производить тестирование, определение параметров и отладку микропроцессорных систем.

Выполнение студентами лабораторных работ по ПМ.02 «Применение микропроцессорных систем, установка и настройка периферийного оборудования», МДК.02.01 «Микропроцессорные системы»направлено на:

- обобщение, систематизацию, углубление, закрепление, развитие и детализацию полученных теоретических знаний по конкретным темам междисциплинарных курсов;
- формирование умений применять полученные знания на практике, реализацию единства интеллектуальной и практической деятельности;
- приобретение навыков работы с различными приборами, аппаратурой, установками и другими техническими средствами для проведения опытов;

- развитие интеллектуальных умений у будущих специалистов: аналитических, проектировочных и др.;

- выработку при решении поставленных задач профессионально значимых качеств, таких как самостоятельность, ответственность, точность, творческая инициатива.

Практические и лабораторные занятия проводится после соответствующей темы, которая обеспечивает наличие знаний, необходимых для ее выполнения.

Продолжительность выполнения практической или лабораторной работы составляет не менее двух академических часов (от 2 до 6) и проводится после соответствующего занятия, которое обеспечивает наличие знаний, необходимых для ее выполнения.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно и полностью.

Оценка «хорошо» ставится, если допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Тема 1.3 Микропроцессорные системы (МПС)
Практическая работа № 1
Изучение схемы типовой МПС

Формируемые компетенции:

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы: изучить принципы построения и функционирования МПС на примере МПС серии КР580

Выполнив работу, Вы будете:

уметь:

- выбирать микроконтроллер/микропроцессор для конкретной системы управления

Материальное обеспечение:

не требуется

Теоретические сведения

Комплект микросхем серии КР580, выполненных по n-МДП- и ТТЛШ - технологии, характеризуется архитектурным единством, которое обеспечивается автономностью и функциональной законченностью отдельных микросхем, унификацией их интерфейса, программируемостью микросхем, их логической и электрической совместимостью. Восьмиразрядная организация, фиксированный набор команд, большой выбор периферийных микросхем различного назначения, относительно высокое быстродействие, умеренное потребление мощности обеспечивают МПК широкое применение при создании средств вычислительной техники, устройств локальной автоматики, контроллеров измерительных приборов и периферийных устройств. микро-ЭВМ для управления технологическими процессами и измерительными системами и др. Состав МПК серии КР580 приведен в табл. 1.

Таблица 1 - Состав серии КР580

Тип микросхем	Функциональное назначение
КР580ВМ80А	Однокристальный 8-разрядный микропроцессор
КР580ВВ51А	Программируемый последовательный интерфейс
КР580ВИ53	Программируемый таймер
КР580ВВ55А	Программируемый параллельный интерфейс
КР580ВТ57	Контроллер прямого доступа к памяти
КР580ВН59	Контроллер прерываний
КР580ВВ79	Интерфейс клавиатуры
КР580ВГ75	Контроллер ЭЛТ
КР580ВК91А	Интерфейс МП-канал общего пользования
КР580ВА93	Приемопередатчик МП-канал общего пользования
КР580ГФ24	Генератор тактовых сигналов
КР580ВК28, КР580ВК38	Системный контроллер и шинный формирователь
КР580ИР82, КР580ИР83	Буферный регистр/регистр с инверсией
КР580ВА86, КР580ВА87	Шинный формирователь/формирователь с инверсией

Типовая электрическая принципиальная схема микропроцессорной системы на базе микросхем серии КР580 приведена на рис. 1. Число и состав микросхем в системе определяются требованиями, предъявляемыми потребителем. Необходимыми компонентами в любой системе являются:

- микропроцессор КР580ВМ80А,
- генератор КР580ГФ24,
- системный контроллер КР580ВК28 (КР580ВК38),

- буферная схема адреса (построена на двух микросхемах КР580ВА86 (КР580ВА87) для обеспечения нагрузочной способности по шине адреса).

Объем памяти ЗУ и использование одной или нескольких периферийных микросхем КР580ВВ51А, КР580ВИ53, КР580ВВ55А, КР580ВТ57, КР580ВН59, КР580ВВ79 или КР580ВГ75 определяет пользователь.

Микропроцессорная система имеет системную шину, образуемую из трех шин:

- адреса A15—A0,
- данных D7—D0,
- управления.

Системная шина позволяет строить микропроцессорную систему по модульному принципу: модуль центрального процессора, модуль ЗУ, модуль УВВ и т. д. Каждый модуль может содержать собственные буферные схемы адреса и данных. Двунаправленные выводы данных периферийных микросхем рекомендуется подключать к системной шине через шинные формирователи (КР580ВА86, КР580ВА87 или КР589АП16, К589АП26). Магистральная структура микропроцессорной системы позволяет подключать микросхемы ЗУ общей емкостью до 64К байт и микросхемы УВВ до 256 каналов ввода и до 256 каналов вывода.

Для помехоустойчивости системы низкочастотные помехи по цепи питания необходимо блокировать конденсатором суммарной емкостью из расчета 0,1 мкФ на каждую микросхему, включенным между шинами +5 В и GND непосредственно в начале шины -5 В.

Высокочастотные помехи необходимо блокировать конденсатором емкостью 0,015— 0,022 мкФ, включенным между каждым выводом +5 В микросхемы и шиной GND в непосредственной близости от микросхем (не далее 5 мм).

Для увеличения быстродействия системы трехстабильные линии шины адреса и данных рекомендуется подключать к шинам +5 В через резисторы сопротивлением 2,2 кОм.

Микросхемы серии КР580 по входам и выходам совместимы с микросхемами ТТЛ серий К133 и К155.

Основные стыковочные параметры ИМС серии КР580 даны в табл. 2.

Таблица 2 – Параметры ИМС

Параметр	Обозначение	Значения параметров [макс (мин.)]
Напряжение питания, В	U_{CC}	5,25(4,75)
Входное напряжение низкого уровня, В	U_{IL}	0,8
	U_{IH}	(2,0)
Входное напряжение высокого уровня, В	U_{OL}	0,45
	U_{OH}	(2,4)
Выходное напряжение низкого уровня, В	I_{OL}	2,2
	I_{OH}	-0,4
Выходное напряжение высокого уровни, В	I_{LI}	±10
	I_{OZ}	±10
Выходной ток низкого уровня, мА	C_L	100
Выходной ток высокого уровня, мА	C_I	10
Ток утечки на входах, мкА	C_O	20
Ток утечки на входах/выходах, мкА		
Емкость нагрузки, пФ		
Емкость на входах, пФ		
Емкость на входах/выходах, пФ		

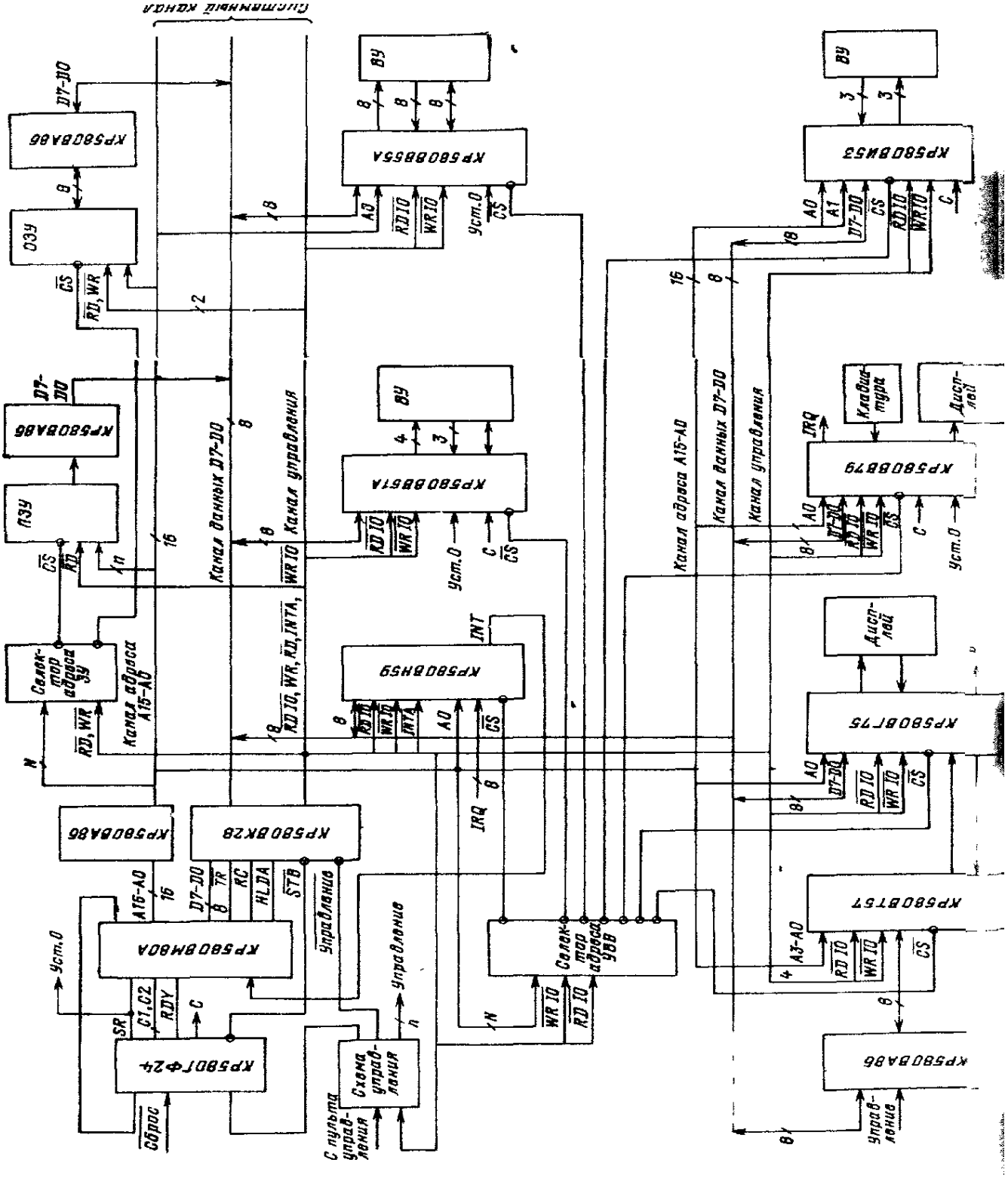


Рис.1 – Типовая электрическая принципиальная схема МПС на основе МП К580ВМ80А

Микросхема КР580ГФ24 — генератор тактовых сигналов фаз Cl, С2, предназначен для синхронизации работы микропроцессора КР580ВМ80А.

Генератор формирует:

- две фазы Cl,C2 с положительными импульсами, сдвинутыми во времени, амплитудой 12 В и частотой 0,5—3,0 МГц;
- тактовые сигналы опорной частоты амплитудой напряжения уровня ТТЛ;
- стробирующий сигнал состояния STB длительностью не менее ($T_{ОП}$/9—15 нс), где $T_{ОП}$— период тактовых сигналов опорной частоты;
- тактовые сигналы С, синхронные с фазой С2, амплитудой напряжения уровня ТТЛ.

Генератор синхронизирует сигналы RDYIN и RESIN с фазой С2.Стробирующий сигнал состояния STB формируется при наличии на входе SYN напряжения высокого уровня, поступающего с выхода микропроцессора КР580ВМ80А в начале каждого машинного цикла. Сигнал STB используют для занесения информации состояния микропроцессора в микросхему КР580ВК28 или КР580ВК38 для формирования управляющих сигналов.

Для согласования работы микропроцессора КР580ВМ80А с другими устройствами сигнал RDYIN синхронизируется по фазе С2 на выходе RDY генератора.

Выходной сигнал SR пользуют для установки в исходное состояние микропроцессора и других микросхем в системе.

Для автоматической установки микропроцессора КР580ВМ80А в исходное состояние при подаче напряжений питания ко входу RESINмикросхемы КР580ГФ24 подключают цепь, состоящую из элементов R,VD, С2.

Задание

Изучить принципиальную схему МПС (назначение устройств и принцип их соединения) и нарисовать обобщенную структурную схему МПС с указанием типовых устройств, входящих в состав МПС

Контрольные вопросы

1. Каково назначение ИМС КР580ВМ80А? Покажите на схеме.
2. Каково назначение ИМС КР580ВВ51А? Покажите на схеме.
3. Каково назначение ИМС КР580ВВ55? Покажите на схеме.
4. Каково назначение ИМС КР580ВИ53А? Покажите на схеме.
5. Каково назначение ИМС КР580ВА86? Покажите на схеме.
6. Каково назначение ИМС КР580ГФ24? Покажите на схеме.
7. Какие шины входят в состав системной шины в данной МПС?
8. Каким схемным способом выполняется защита от помех в данной МПС?
9. Как осуществляется установка в исходное состояние МП и других ИМС в системе?
10. Какую функцию выполняет ИМС КР580ВК28? Покажите на схеме.

Форма представления результата:

Отчет должен содержать:

1. Цель работы.
2. Перечень обязательных компонентов МПС и их назначение.
3. Структурная схема МПС
4. Ответы на контрольные вопросы

Тема 1.5. Организация ввода/выводаданных МПС
Практическая работа №2
Изучение устройства параллельных портов МК ADuC842

Формируемые компетенции:

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы:изучить особенности работы параллельных портов микроконтроллера, а именно работу со светодиодными индикаторами и способами управления ими.

Выполнив работу, Вы будете:

уметь:

- выбирать микроконтроллер/микропроцессор для конкретной системы управления

Материальное обеспечение:

не требуется.

Теоретические сведения

1. Устройство параллельных портов микроконтроллера

Параллельны порты предназначены для обмена многоразрядной двоичной информацией между микроконтроллером и внешними устройствами, при этом в качестве внешнего устройства может использоваться другой микроконтроллер. Каждый из портов содержит восьмиразрядный регистр, имеющий байтовую и битовую адресацию для установки (запись —1) или сброса (запись

—0‖) разрядов этого регистра с помощью программного обеспечения. Выходы этих регистров соединены с внешними ножками микросхемы. С точки зрения внешнего устройства порт представляет собой обычный источник или приемник информации со стандартными цифровыми логическими уровнями (обычно ТТЛ), а с точки зрения микропроцессора — это ячейка памяти, в которую можно записывать данные или в которой сама собой появляется информация.

В качестве внешнего устройства может служить любой объект управления или источник информации (различные кнопки, датчики, микросхемы, дополнительная память, исполнительные механизмы, двигатели, реле и так далее).

В зависимости от направления передачи данных параллельные порты называются портами ввода, вывода или портами ввода вывода.

В качестве простейшего порта вывода может быть использован параллельный регистр, так как он позволяет запоминать данные, выводимые микропроцессором и хранить их до тех пор, пока подается питание. Все это время сигналы с выхода этого регистра поступают на внешнее устройство. Упрощенная функциональная схема порта вывода показана на рисунке 2. Двоичные слова с системной шины данных микропроцессора записываются в регистр по сигналу —WR‖. Выходы "Q" регистра могут быть использованы как источники логических уровней для управления внешними устройствами. Этот регистр называется регистром данных порта вывода.

Для отображения этого регистра только в одну ячейку памяти адресного пространства микропроцессорного устройства в составе порта ввода-вывода всегда присутствует дешифратор адреса. Этот регистр называется регистром данных порта вывода. На выходе дешифратора адреса устанавливается единица лишь тогда, когда двоичное слово на его входе имеет заданное строго определенное значение, соответствующее адресу ячейки памяти, во всех остальных случаях на выходе дешифратора адреса удерживается логический ноль.

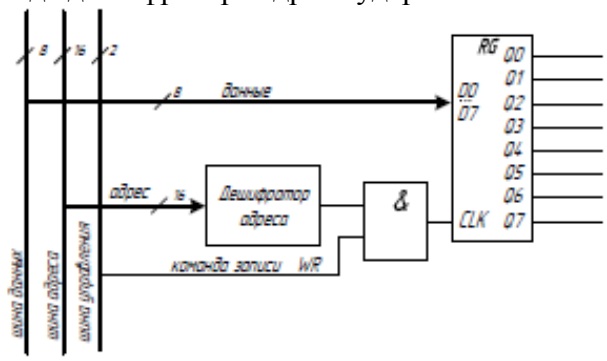


Рис. 2 – Функциональная схема порта вывода

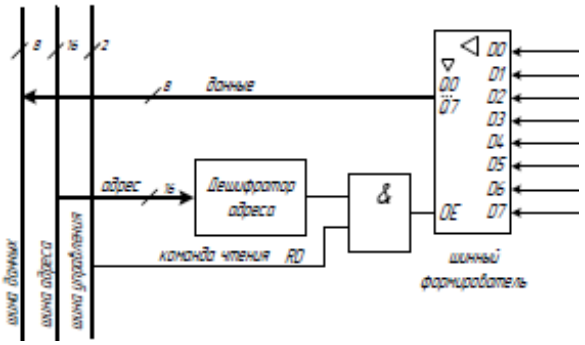


Рис. 3 – Функциональная схема порта ввода

В качестве порта ввода может быть использован шинный формирователь. Для построения порта ввода выходы шинного формирователя (Q0-Q7) подключены к внутренней шине данных микропроцессора, а на его вход подключаются сигналы, которые нужно ввести в микропроцессорную систему. Упрощенная функциональная схема порта ввода приведена на рисунке 3.

Шинный формирователь работает следующим образом: данные с входов D0-D7 поступают на выходы Q0-Q7 лишь тогда когда на входе OE (Output Enable — разрешение выхода) установлен высокий логический уровень (1), когда же на OE логический ноль выходы переходят в третье состояние и никак не влияют на шину данных. Значение сигнала с внешнего вывода порта передается на шину данных (считывается) по управляющему сигналу RD.

Для отображения шинного формирователя порта ввода в один адрес пространства адресов микроконтроллера используется дешифратор адреса. Выделяемый дешифратором адрес называют адресом регистра данных порта ввода. Из порта ввода возможно только чтение информации.

Так как из порта ввода возможно только чтение информации (команд RD), а в порт вывода только запись (команда WR), то для портов ввода и вывода можно отвести один и тот же адрес в адресном пространстве микропроцессора. Упрощенная схема одного разряда параллельного порта ввода-вывода приведена на рисунке 4. Подобное схемотехническое решение применяется в микроконтроллерах архитектуры MCS-51, такой порт называется квазидвунаправленным. Для микроконтроллеров других архитектур схема может отличаться.

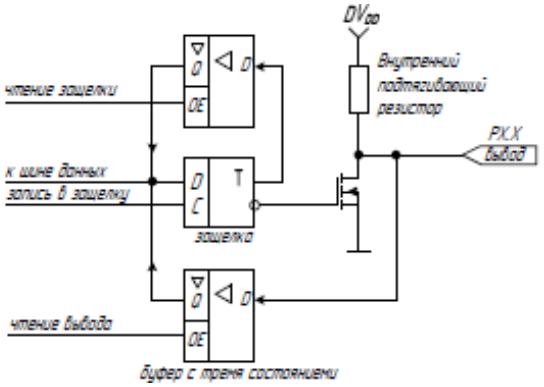


Рисунок 4 – Упрощенная схема одного бита параллельного квазидвунаправленного порта

Один разряд регистра порта представляет собой D-триггер, запись входных данных в который происходит по высокому уровню синхросигнала (вход триггера —C‖). На рисунке такой сигнал назван «запись в защелку». Мощность сигнала с инвертирующего выхода триггера усиливается при помощи МОП (металл — окись — полупроводник) транзистора, а за тем поступает на внешний вывод микросхемы. Внутренний подтягивающий резистор служит для обеспечения выходного тока порта при установленном высоком логическом уровне. Как правило, вместо резистора в портах микроконтроллера используется управляемый генератор стабильного тока (ГСТ), собранный на МОП транзисторах. Следует обратить внимание, что у некоторых портов вывода, внутреннего генератора тока может и не быть, пример тому P0.

2. Подключение внешних устройств к порту параллельному порту микроконтроллера

Внешними устройствами называются любые устройства, которыми управляет, от которых получает или которым передает информацию микроконтроллер. В качестве внешних устройств могут выступать устройства ввода-вывода информации — светодиодный индикатор, дисплей, датчик, кнопка, клавиатура, другой микропроцессор.

Внутренняя схема порта построена таким образом, чтобы максимально упростить подключение внешних устройств к микроконтроллеру. Присутствие в схеме порта выходного транзистора позволяет подключить к выводам порта светодиодные индикаторы непосредственно, без усилителя мощности, как это показано на рисунке 5. При этом необходимо следить за максимальной допустимой мощностью, рассеиваемой на микросхеме и отдельных выводах порта. Светодиод в такой схеме загорается при низком потенциале на выводе порта, для этого в порт должен быть записан логический ноль. Резистор R1 ограничивает ток через светодиод, если этот резистор не поставить, то ток по цепи индикатора может достигнуть недопустимой величины и светодиод или внутренний транзистор выйдут из строя.

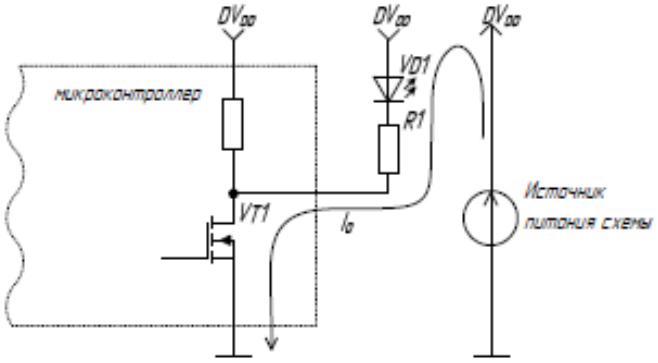


Рисунок 5 – Эквивалентная схема подключения светодиодного индикатора к параллельному порту

Если же требуется обеспечить больший ток нагрузки, то для усиления тока порта используют внешний транзистор, как показано на рисунке 6.

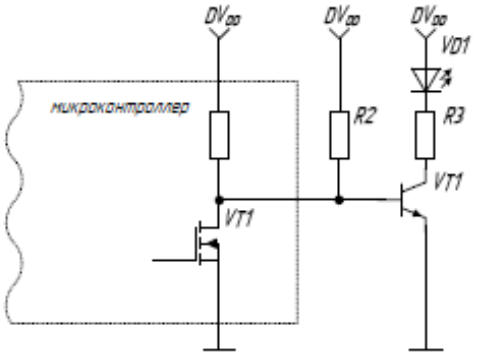


Рисунок 6 – Схема подключения светодиодного индикатора
через транзисторный ключ

В приведенной схеме база транзистора подключена непосредственно к выводу порта. Если выходного порта контроллера достаточно для открывания транзистора, то резистор R2 может быть исключен из схемы. Этот резистор нужен для увеличения тока базы транзисторного ключа. Резистор R3 рассчитывается исходя из допустимого тока светодиода. Для зажигания светодиода необходимо в соответствующий разряд параллельного порта записать логическую единицу, а для его гашения — логический ноль.

Двунаправленный порт позволяет получать цифровую информацию от различных источников — кнопок, датчиков, микросхем. Согласование микросхем между собой не представляет трудностей, так как практически все современные цифровые микросхемы по входу и выходу согласованны между собой и имеют строго определенные значения логических уровней. С датчиками и кнопками дело обстоит несколько сложнее. Все датчики выполняются так, что они с точки зрения электрической схемы представляют собой контакты, работающие на замыкание-размыкание. Поэтому схема подключения датчиков и кнопок принципиально не отличаются. Со стороны микроконтроллера замыкание–размыкание должно интерпретироваться как подача на вход логических уровней. Простейшая схема, преобразующая замыкание контактов в логические уровни показана на рисунке 7.

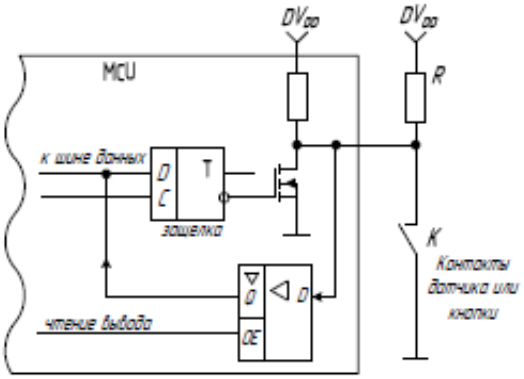


Рисунок 7 – Подключение источника цифровой информации

Когда контакт K разомкнут то на вход микроконтроллера через резистор R подается напряжение питания, интерпретируемое контроллером как логическая единица. Если контакт замкнут, то на вход подается потенциал общего провода, что соответствует логическому нулю.

Для того что бы ввести информацию с параллельного порта в него должна быть записаны логические единицы. Дело в том, что если в разряд записать логическую единицу, то выходной транзистор закрывается и на выводе микросхемы за счет внутреннего подтягивающего резистора устанавливается высокий уровень. Этот уровень может быть изменен на нулевой потенциал замыканием вывода микросхемы на общий провод. И информация может быть правильно интерпретирована микропроцессором. В случае же когда в разряд порта записан логический ноль, внутренний транзистор открыт, на выходе появляется низкий потенциал, изменить который извне невозможно. Поэтом, перед тем как осуществить ввод информации через параллельный порт, соответствующий разряд необходимо настроить на ввод — записать в него логическую единицу.

3. Особенность параллельных портов микроконтроллера ADuC842

В микроконтроллере ADuC842, фирмы Analog Devices, для обмена информации с внешними устройствами используется четыре параллельных порта ввода-вывода. Дополнительно к своему основному назначению некоторые порты могут использоваться для подключения внешней памяти.

Port0 (P0.0 – P0.7) — Двунаправленный 8-ми разрядный параллельный порт ввода-вывода с открытым стоком. При записи в бит регистра порта логической единицы соответствующая линия порта переходит в режим высокоимпедансного входа. Для работы в режиме порта ввода-вывода необходимо внешнее подтягивание каждого разряда порта к уровню логической единицы.

Port1 (P1.0 – P1.7) — Входной порт, по умолчанию настроен на ввод аналоговых сигналов (функция АЦП). Каждый ввод может быть переведен в режим цифрового входа, для этого в соответствующий бит порта должен быть записан логический ноль. Порт не имеет внутреннего усиливающего транзистора и потому при вводе дискретной информации через него не требуется записывать в разряды логическую единицу. В режиме порта цифрового ввода необходимо внешнее подтягивание каждого разряда порта к уровню логической единицы.

Port2(P2.0 – P2.7), Port3(P3.0 – P3.7) — Двунаправленные 8-ми разрядные параллельные порты ввода-вывода. При записи в бит регистра порта логической единицы, соответствующая линия порта переходит в режим высокоимпедансного входа со слабым подтягиванием сигнала к уровню логической единицы.

Формат и адреса портов P0-P3 приведены на рисунке 8. На этом же рисунке показаны адреса отдельных битов портов в битовом пространстве памяти.

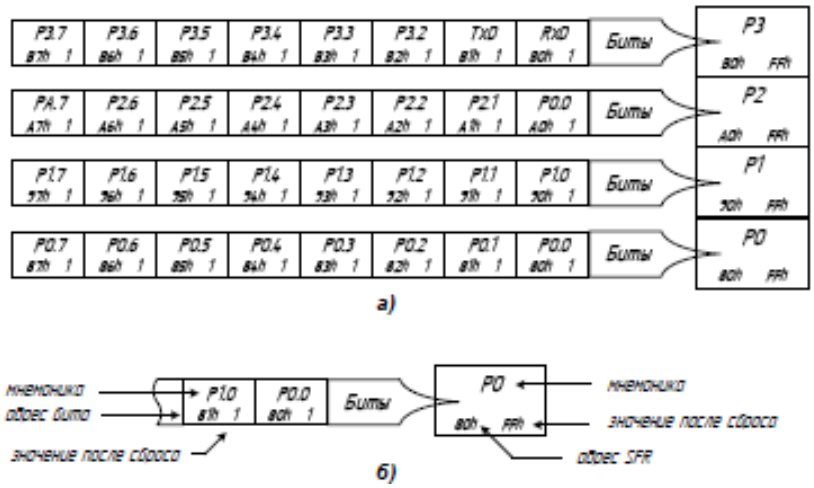


Рисунок 8 – Адреса SFR параллельных портов

Микроконтроллер ADuC842 в лабораторном практикуме исследуется в составе учебного лабораторного стенда LESO1, для того чтобы определить какое периферийное устройство подключено к какому порту следует изучить принципиальную схему учебного стенда.

Задание

Зарисовать в тетрадь функциональные схемы портов ввода-вывода, а также схемы подключения внешних устройств к параллельным портам (индикатор и кнопка), знать назначение всех компонентов.

Форма представления результата:

Отчет должен содержать:

1. Цель работы.
2. Эквивалентная схема подключения светодиода к параллельному порту.
3. Эквивалентная схема подключения источника цифрового сигнала к параллельному порту.

Практическая работа №3

Изучение схемы подключения матричной клавиатуры к МК ADuC842

Формируемые компетенции:

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы:изучить особенности работы параллельных портов микроконтроллера, а также изучить схемы подключения кнопок и матричной клавиатуры к микроконтроллеру.

Выполнив работу, Вы будете:

уметь:

- выбирать микроконтроллер/микропроцессор для конкретной системы управления

Материальное обеспечение:

не требуется.

Теоретические сведения

Применение матричной клавиатуры для ввода информации в микропроцессорную систему

Для реализации взаимодействия пользователя с микропроцессорной системой используют различные устройства ввода-вывода информации. В самом простом случае в роли устройства ввода может выступать кнопка, представляющая собой элементарный механизм, осуществляющий замыкание-размыкание контактов под действием внешней механической силы. Схема подключения кнопки к линии ввода параллельного порта ввода микроконтроллера показана на рисунке 9. Когда контакты кнопки S1 разомкнуты через резистор R1 на вход контроллера поступает высокий логический уровень ―1‖, когда же контакты замкнуты, то вход оказывается соединенным с общим проводом, что соответствует логическому уровню ―0‖. Если параллельный порт микроконтроллера имеет встроенный генератор тока, то в схеме можно обойтись без резистора R1.

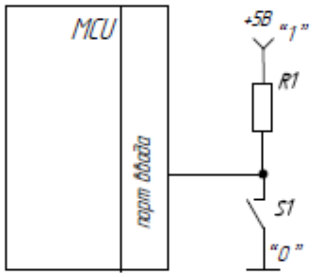


Рисунок 9 – Подключение одиночной кнопки к параллельному порту

Недостаток приведенной схемы заключается в том, что для подключения каждой кнопки требуется отдельная линия параллельного порта. Так как часто требуется вводить информацию с большого количества кнопок, то для уменьшения количества линий ввода-вывода используется клавиатура, представляющая собой двухмерную матрицу кнопок, организованных в ряды и столбцы (рисунок 10).

Подключение клавиатуры отличается от схемы подключения одиночной кнопки тем, что потенциал общего провода на опрашиваемые кнопки подается не непосредственно, а через порт вывода.

В каждый момент времени сигнал низкого уровня (логический ноль) подается только на один столбец кнопок, на остальные должна подаваться логическая единица. Это исключит неоднозначность определения номера нажатой кнопки. Двоичные сигналы, присутствующие при этом на строках клавиатуры, считываются через порт ввода микроконтроллера.

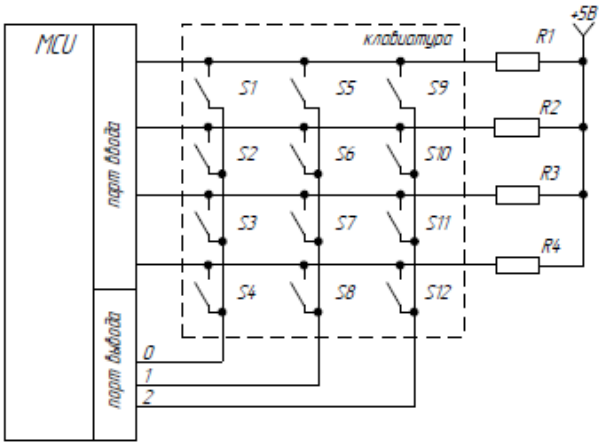


Рисунок 10 – Подключение матричной клавиатуры к параллельному порту

Временная диаграмма напряжений на портах вывода при выполнении программы опроса клавиатуры приведена на рисунке 11.

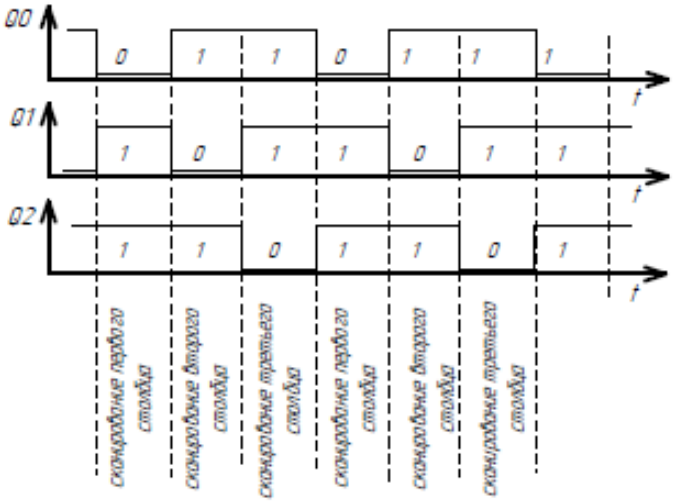


Рисунок 11 – Временные диаграммы работы порта вывода

В каждый момент времени производится чтения информации из порта ввода. Программа микроконтроллера по считанной комбинации должна определить номер нажатой кнопки клавиатуры.

Задание

Зарисовать в тетрадь схемы подключения одиночной кнопки и матричной клавиатуры к параллельным портам (индикатор и кнопка), знать назначение всех компонентов.

Форма представления результата:

Отчет должен содержать:

1. Цель работы.
2. Эквивалентная схема подключения кнопки к параллельному порту.
3. Эквивалентная схема подключения клавиатуры к параллельному порту.

Практическая работа №4
Изучение таймеров МК ADuC842

Формируемые компетенции:

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы: изучить особенности работы таймеров микроконтроллера, а также методику конфигурирования таймеров и формирования с помощью таймера временных интервалов.

Выполнив работу, Вы будете:

уметь:

- выбирать микроконтроллер/микропроцессор для конкретной системы управления

Материальное обеспечение:

не требуется.

Теоретические сведения

1. Общие сведения о таймерах

Таймеры предназначены для формирования временных интервалов, позволяя микропроцессорной системе работать в режиме реального времени.

Таймеры представляют собой цифровые счѐтчики, которые подсчитывают импульсы либо от высокостабильного генератора частоты, либо от внешнего источника сигнала, в этом случае таймер называют счетчиком внешних событий. К системной шине микропроцессора таймеры подключаются при помощи параллельных портов.

Как правило, в микропроцессорной системе в качестве генератора частоты выступает генератор внутренней синхронизации микроконтроллера. Частота генератора задает минимальный интервал времени, который может определять таймер. Интервалы времени, задаваемые таймером, могут устанавливаться только из дискретного набора допустимых времѐн. Разрядность цифрового счѐтчика, входящего в состав таймера, определяет максимальный интервал времени, который может определять таймер.

Обычно используются 16-тиразрядные таймеры, поэтому, для подключения такого таймера к 8-миразрядному процессору требуется два параллельных порта. Кроме того, таймером нужно управлять. Таймер нужно включать и выключать, определять, не возникало ли переполнение таймера. Факт переполнения запоминается в дополнительном триггере, подключенном к выходу переноса счетчика таймера. Этот триггер называется флагом переполнения таймера. Триггер (флаг) включения и выключения таймера и флаг переполнения таймера подключают к системной шине микропроцессора через дополнительный порт ввода-вывода. Структурная схема таймера в самом общем виде показана на рисунке 12. Каждый из портов ввода-вывода отображается во внутреннем адресном пространстве микропроцессора, и имеет свой отдельный адрес.

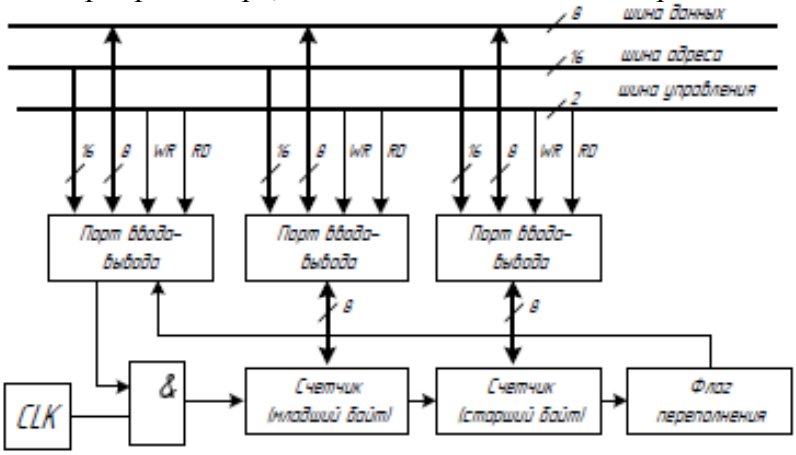


Рисунок 12 – Структурная схема таймера

Очевидно, максимальное число, которое может быть записано в 16- битный счетный регистр таймера равно $2^{16} - 1 = 65535$, что представляет собой логическую единицу в каждом разряде регистра. Таким образом, если перед запуском таймера в его счетчики были записаны нули, то переполнение таймера произойдет через 65536 машинных циклов. Зная частоту задающего генератора микропроцессорной системы, а как следствие и период сигнала генератора T_G, можно легко определить время переполнения таймера в секундах:

$$T_T = 65536\, T_G .$$

Если же требуется установить меньший интервал времени, то перед запуском таймера в его регистры можно записать начальный код, и тогда счет начнется не с нуля, а с записанного кода, и счетчику потребуется меньше времени для переполнения. В этом случае время работы таймера TT определяется по формуле:

$$T_T = (2^n - Code) \cdot T_G, \quad (1)$$

где: *Code* – код, записанный в таймер до его запуска, *TT* – Время работы таймера, *TG* – период колебаний задающего генератора, *n* – разрядность таймера.

Если же требуется сформировать интервал времени больший, чем максимальное время переполнения, то таймер можно запустить несколько раз в цикле. В этом случае временной интервал определяется как:

$$\Delta T = TT \cdot N, \quad (2)$$

где N – количество итераций цикла, TT– время срабатывания таймера.

2. Таймеры-счетчики микроконтроллера ADuC842

Микроконтроллер **ADuC842** имеет три 16-разрядных таймера-счетчика: Таймер 0, Таймер 1 и Таймер 2. Структура и режимы работы таймеров-счетчиков соответствуют общим принципам архитектуры MCS-51.

Каждый таймер-счетчик содержит по два 8-битных регистра THx и TLx (x = 0, 1, и 2). Каждый таймер-счетчик может быть запрограммирован на работу в качестве либо таймера (отсчет времени через подсчет внутренних импульсов синхронизации), либо счетчика (подсчет событий на внешнем входе). В обоих случаях переполнение счетного регистра приводит к формированию запроса прерывания и устанавливается специальный флаг переполнения.

В режиме таймера регистр TLx увеличивает свое значение на единицу каждый машинный цикл. Поскольку машинный цикл однотактового ядра состоит из одного тактового периода, то максимальная скорость счета равна тактовой частоте ядра.

В режиме счетчика, регистр TLx увеличивает свое значение на единицу при переходе уровня из высокого в низкий на соответствующем внешнем выводе микроконтроллера: T0, T1 или T2. Когда на внешнем выводе один машинный цикл держится высокий логический уровень, а уже в следующем цикле – низкий, тогда регистр таймера увеличивает свое значение на единицу. Таким образом, для распознавания перехода из 1 в 0 требуется два такта внутреннего генератора микроконтроллера, это значит, что максимальная скорость счета может составить половину частоты внутреннего тактового генератора.

Таймеры 0 и 1 обслуживаются регистром режима **TMOD** и регистром управления **TCON.**

TMOD – регистр конфигурации Таймера 1 и Таймера 0.

SFR адрес **0x89.**

Значение после подачи питания **0x00.**

Регистр не имеет битовой адресации.

TCON – регистр управления Таймера 1 и Таймера 0.

SFR адрес **0x88.**

Значение после подачи питания **0x00.**

Регистр имеет битовую адресацию.

Таблица 3 – Описание бит регистра **TMOD**

<table>
<tr><th>номер</th><th>мнемоника</th><th>описание</th></tr>
<tr><td>7</td><td>GATE</td><td>Бит управления таймером 1.
При GATE=1 для работы необходимо условие TR1=1 и INT1#=1.
При GATE=0 Таймер 1 работает всегда, когда TR1=1.</td></tr>
<tr><td>6</td><td>C/T#</td><td>Бит выбора типа событий для Таймера 1.
При C/T#=1 он работает как счетчик (вход с внешнего вывода T1 – P3.5), при C/T#=0 — как таймер (вход с внутреннего генератора).</td></tr>
<tr><td>5</td><td>M1</td><td rowspan="2">M1, M0 биты определяют режим работы таймера 1
<table><tr><th>M1</th><th>M0</th><th></th></tr><tr><td>0</td><td>0</td><td>TH1 работает как 8-битный таймер-счетчик, TL1 выступает в качестве делителя частоты на 32</td></tr><tr><td>0</td><td>1</td><td>16-битный таймер-счетчик, TH1 и TL1 включены последовательно.</td></tr><tr><td>1</td><td>0</td><td>8-битный таймер-счетчик с автоперезагрузкой, TH1 удерживает значение, которое загружается в TL1 всякий раз при переполнении TL1.</td></tr><tr><td>1</td><td>1</td><td>Таймер-счетчик 1 остановлен.</td></tr></table></td></tr>
<tr><td>4</td><td>M0</td></tr>
<tr><td>3</td><td>GATE</td><td>Бит управления таймером0.
При GATE=1 для работы необходимо условие TR0=1 и INT0#=1.
При GATE=0 Таймер 0 работает всегда, когда TR0=1.</td></tr>
<tr><td>2</td><td>C/T#</td><td>Бит выбора типа событий для Таймера 0.
При C/T#=1 он работает как счетчик (вход с внешнего вывода T0 – P3.4), при C/T#=0 — как таймер (вход с внутреннего генератора).</td></tr>
<tr><td>1</td><td>M1</td><td rowspan="2">M1, M0 биты определяют режим работы таймера 0
<table><tr><th>M1</th><th>M0</th><th></th></tr><tr><td>0</td><td>0</td><td>TH0 работает как 8-битный таймер-счетчик, TL0 выступает в качестве делителя частоты на 32</td></tr><tr><td>0</td><td>1</td><td>16-битный таймер-счетчик, TH0 и TL0 включены последовательно.</td></tr><tr><td>1</td><td>0</td><td>8-битный таймер-счетчик с автоперезагрузкой, TH0 содержит значение, которое загружается в TL0 всякий раз при переполнении TL0.</td></tr><tr><td>1</td><td>1</td><td>TL0 используется в качестве 8-битного таймера-счетчика со стандартными битами управления Таймера 0.
TH0 используется только в качестве 8-битного счетчика, управление происходит стандартными битами управления таймера 1.</td></tr></table></td></tr>
<tr><td>0</td><td>M0</td></tr>
</table>

Таблица 4– Описание бит регистра TCON

номер	мнемоника	описание
7	TF1	Флаг переполнения Таймера 1. Устанавливается аппаратно при переполнении счетного регистра таймера. Очищается аппаратно при передаче управления на процедуру обработки прерывания.
6	TR1	Бит запуска Таймера 1. При TR1=1 счет разрешен.
5	TF0	Флаг переполнения Таймера 0. Устанавливается аппаратно при переполнении счетного регистра таймера. Очищается аппаратно при передаче управления на процедуру обработки прерывания.
4	TR0	Бит запуска Таймера 0. При TR0=1 счет разрешен.
3	IE1	Флаг внешнего прерывания 1 (INT1#) Устанавливается аппаратно при падающем фронте или при нулевом уровне на внешней ножке INT1#, зависит от состояния бита IT1. Очищается аппаратно при передаче управления на процедуру обработки прерывания.
2	IT1	Бит выбора типа активного сигнала на входе INT1#. При IT1=1 активным является переход из высокого в низкий, при IT1=0 активным является низкий уровень сигнала.
1	IE0	Флаг внешнего прерывания 1 (INT0#) Устанавливается аппаратно при падающем фронте или при нулевом уровне на внешней ножке INT0#, зависит от состояния бита IT0. Очищается аппаратно при передаче управления на процедуру обработки прерывания.
0	IT0	Бит выбора типа активного сигнала на входе INT1#. При IT0=1 активным является переход из высокого в низкий, при IT0=0 активным является низкий уровень сигнала.

Каждый таймер содержит два 8-битных регистра, которые могут быть использованы как независимые регистры или скомбинированы в одиночные 16-битные регистры в зависимости от режима работы таймера.

TH0 и **TL0** – старший и младший байт Таймера 0.

SFR адрес – 0x8C и 0x8A, соответственно.

TH1 и **TL1** – старший и младший байт Таймера 1. SFR адрес – 0x8D и 0x8B, соответственно.

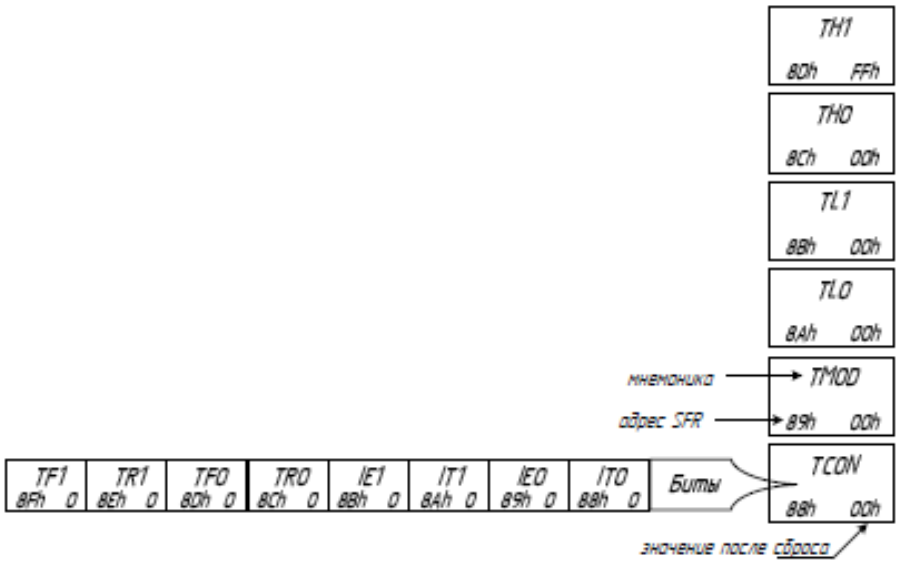


Рисунок 13 – Адреса регистров SFR таймеров

3. Режимы работы Таймера 1 и 0

Для выбора нулевого режима следует установить биты **M1 =0** и **M0 = 0** регистра **TMOD**. В этом режиме таймер-счетчик сконфигурирован как 13-битный счетный регистр. Этот счѐтчик состоит из 8 бит регистра **TH**x и младших 5 бит регистра **TLx,** где **x** в обозначении регистра заменяется на 0 или 1 в зависимости от того таймера, которым мы управляем. Старшие 3 бита регистров **TLx** не определены и игнорируются. Установка запускающего таймер флага **TR0** или **TR1** не очищает эти регистры. Работе таймера 0 или таймера 1 в режиме 0 соответствует схема:

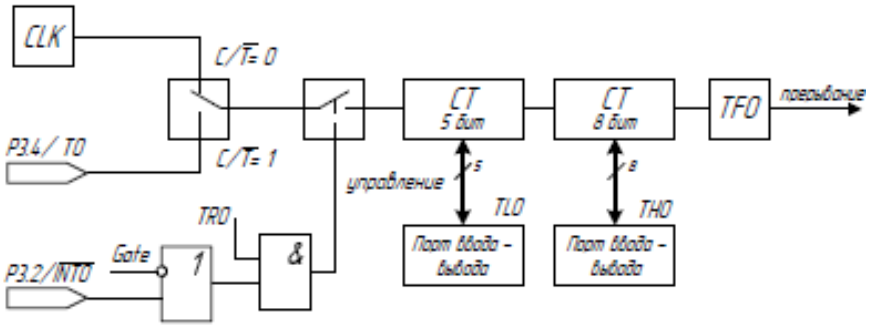


Рисунок 14 – Структурная схема таймера в режиме 0 (для Таймера 0)

Переполнение таймера происходит, когда единицы в каждом разряде меняются нулями, при этом флаг переполнения **TFx** устанавливается в единицу. Флаг **TF**x может быть использован для прерывания. К входу счетчика подключен источник сигналов, когда **TR**x = 1 и выполняется одно из условий **Gate** = 0 или **INT**x = 1. В этом случае удобно управлять таймером через флаг **TR**x. В случае, когда **Gate** = 1 таймер может управляться с внешнего вывода микроконтроллера **INT**x, такая возможность позволяет измерять длительность импульсов. Измеряемый импульс должен подаваться на вход **INT**x. Тип источника тактовых импульсов определяется флагом **C/T#,** когда флаг установлен (**C/T#** = 1), сигнал на счетчик подается с внешнего вывода микросхемы **Tx,** если флаг сброшен (**C/T#** = 0) – с внутреннего тактового генератора. Флаг **TRx** контролируется через регистр специальной функции **TCON**, флаги **Gate** и **C/T#** – через **TMOD.**

Этот режим был введѐн для совместимости с устаревшим семейством микроконтроллеров MCS-48 для облегчения переноса уже разработанных программ на новые процессоры, и поэтому в настоящее время не используется.

Для выбора первого режима следует установить биты **M1 = 0** и **M0 = 1** регистра **TMOD**. В первом режиме работы таймер работает как шестнадцатиразрядный счѐтчик. Режим 1 похож на режим 0, за исключением того, что в регистрах таймера используются все 16 бит. В этом режиме

регистры **TH**x и **TL**x также включены друг за другом. Работе таймера 0 или таймера 1 в режиме 1 соответствует схема:

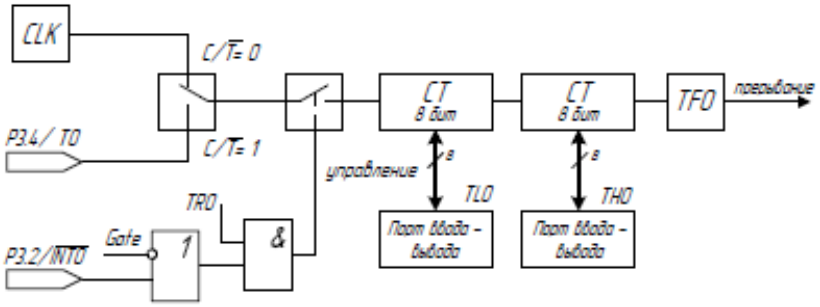


Рисунок 15 – Структурная схема таймера в режиме 1 (для Таймера 0)

Нулевой и первый режимы работы Таймеров 0 и 1 предназначены для формирования одиночного интервала времени. Если возникает необходимость формировать последовательность интервалов времени для периодических процессов, то загрузка регистров **THx** и **TLx** для задания нужного интервала времени производится программно, что для коротких интервалов времени может привести к значительным затратам процессорного времени.

Для формирования последовательности одинаковых интервалов времени используется режим работы таймера с перезагрузкой – режим 2.

Для выбора второго режима следует установить биты **M1 =1** и **M0 = 0** регистра **TMOD**. В режиме 2 регистр таймера **TL**x работает как 8-битный счетчик с автоматической перезагрузкой начального значения из регистра **TH**x в регистр **TL**x. Переполнение регистра **TL**x не только устанавливает флаг **TF**x, но и загружает регистр **TL**x содержимым регистра **TH**x, который предварительно инициализируется программно. Перезагрузка не изменяет содержимое регистра **TH**x. Работе таймера 0 или таймера 1 в режиме 2 соответствует схема:

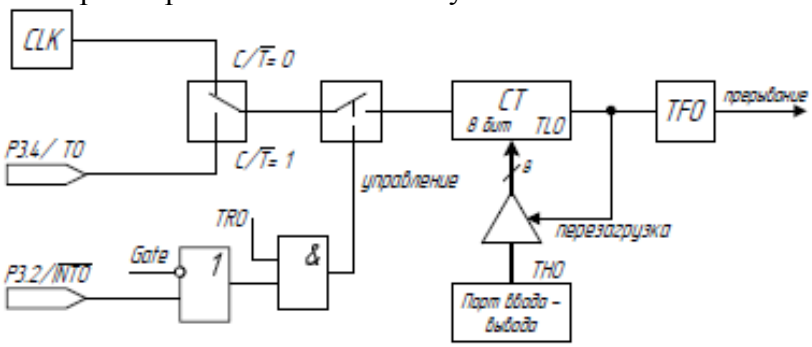


Рисунок 16 – Структурная схема таймера в режиме 2 (для Таймера 0).

Для таймеров 0 и 1 третий режим различается. Таймер 1 в режиме 3 просто хранит свое значение – такой же эффект как при TR1 = 0. Таймер 0 в режиме 3 работает как два независимых счетчика, это показано на рисунке 17. Регистр **TL0** использует биты управления таймера 0: **C/T#, GATE**, **TR0** и **TF0**. Регистр **TH0** работает только в режиме таймера, и использует биты **TR1** и **TF1** таймера 1.

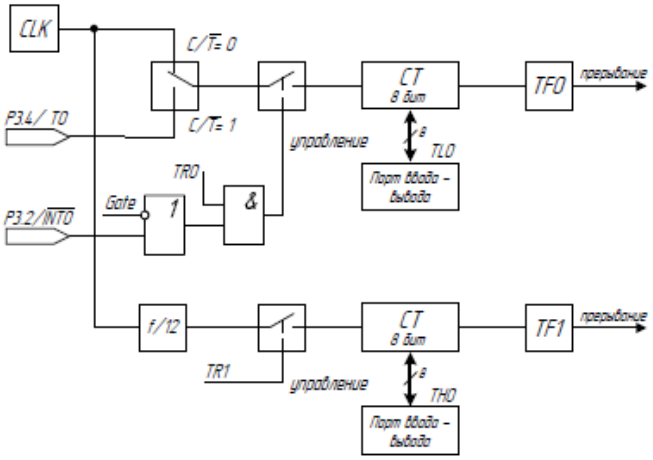


Рисунок17 – Структурная схема таймера в режиме 3.

Работа таймера TL0 разрешается, если бит **TR0** = 1, а таймера **TH0** – если бит **TR1** = 1. Таймер 1 при работе таймера 0 в режиме 3 постоянно включен.

Этот режим работы позволяет реализовать два независимых таймера, если таймер 1 используется для работы последовательного порта. В микроконтроллере ADuC842 для синхронизации последовательного порта принято использовать специально для этого выделенный четвертый таймер (Таймер 3), поэтому этот режим мало чем интересен.

4. Особенность работы тактового генератора ADuC842

Встроенный генератор микроконтроллера ADuC842 предназначен для работы с кварцевым резонатором 32.768 кГц. Для формирования тактов синхронизации процессора используется умножитель частоты: система фазовой автоподстройки частоты (ФАПЧ) и управляемый делитель. С помощью системы ФАПЧ частота тактового генератора умножается на 512, что соответствует 16,777216 МГц. Делитель же настраивается на фиксированные коэффициенты деления (1, 2, 4, 8, 16, 32, 64, 128). При включении микроконтроллера по умолчанию установлен делитель 8, что соответствует частоте 16,777216/8=2,097152(МГц). Изменить этот делитель можно через регистр PLLCON.

Задание

Зарисовать в тетрадь структурные схемы работы таймера в различных режимах работы, а также формулы для определения времени работы таймера и временного интервала, знать назначение битов, хранящихся в регистрах TMOD и TCON.

Форма представления результата:

Отчет должен содержать:

1. Цель работы.
2. Структурная схема таймера в режиме 0.
3. Структурная схема таймера в режиме 1.
4. Структурная схема таймера в режиме 2.
5. Структурная схема таймера в режиме 3.
6. Состав регистров TMOD и TCON.

Практическая работа №5
Изучение устройства последовательного порта МК ADuC842

Формируемые компетенции:

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы: изучить особенности работы последовательных портов (UART), освоить методику расчета скорости последовательного порта.

Выполнив работу, Вы будете:

уметь:

- выбирать микроконтроллер/микропроцессор для конкретной системы управления

Материальное обеспечение:

не требуется.

Теоретические сведения

1. Последовательный асинхронный интерфейс UART

Последовательный интерфейс использует одну сигнальную линию для передачи данных, по которой биты информации передаются друг за другом последовательно. При последовательной передаче сокращается количество сигнальных линий, что упрощает разводку проводников на печатной плате, уменьшает габариты устройства и позволяет делать более помехозащищенные интерфейсы. При последовательной передаче каждый информационный бит должен сопровождаться импульсом синхронизации — стробом. Если импульсы синхронизации передаются от одного устройства к другому по выделенной линии, то такой интерфейс называют синхронным, в этом случае генератор синхронизации располагается на стороне устройства инициирующего передачу. Если же приемник и передатчик содержат каждый свой генератор синхроимпульсов, работающий на одной частоте, то такой интерфейс называется асинхронным. Получается, что приемник информации сам вырабатывает синхроимпульсы.

Типичный представитель асинхронного последовательного интерфейса — **UART** (Universal Asynchronous Receiver-Transmitter — универсальный асинхронный приѐмопередатчик).

При передаче по интерфейсу UART каждому байту данных предшествует **СТАРТ-бит**, сигнализирующий приемнику о начале посылки, за **СТАРТ-битом** следуют биты данных. Завершает посылку **СТОП-бит**, гарантирующий паузу между посылками. **СТАРТ-бит** следующего байта посылается в любой момент после **СТОП-бита**, то есть между передачами возможны паузы произвольной длительности. **СТАРТ-бит**, обеспечивает простой механизм синхронизации приемника по сигналу от передатчика. Внутренний генератор синхроимпульсов приемника использует счетчик-делитель опорной частоты, обнуляемый в момент приема начала **СТАРТ-бита**. Этот счетчик генерирует внутренние стробы, по которым приемник фиксирует последующие принимаемые биты.

2. Особенности работы UART микроконтроллера ADuC842

В микроконтроллере ADuC842 последовательный порт UART является полнодуплексным — позволяет передавать и принимать данные одновременно. В рассматриваемом микроконтроллере прием через последовательный порт буферизирован: порт может начать принимать байт данных еще до того как предыдущий байт будет считан из буферного регистра приемника. Однако, если до конца приема предыдущий байт не будет считан из буфера, он будет потерян: новый принятый байт перезапишет старый.

Принимаются и передаются данные по разным линиям, передача происходит через вывод **TxD** микроконтроллера (Transmitter Data — передатчик данных), прием — через **RxD** (Receiver Data — приемник данных). Физически выводы **RxD** и **TxD** совмещены с выводами третьего параллельного порта P3.0 и P3.1 соответственно.

Программное взаимодействие с последовательным портом UART осуществляется через регистры специальных функций (SFR) **SBUF** и **SCON**.

Через **SBUF** (Serial buffer — последовательный буфер) осуществляется доступ к регистрам приемника и передатчика последовательного порта. Когда программно производится запись в **SBUF**, то данные загружаются в регистр передатчика, когда же программой происходит чтение **SBUF**, то осуществляется доступ к регистру приемника. Физически регистры приемника и передатчика разделены. **SFR** адрес — 0x99.

SCON — регистр конфигурации и управления последовательным портом микроконтроллера.

SFR адрес — **0x98**.

Значение после подачи питания **0x00**.

Регистр имеет побитовую адресацию.

Таблица 5 – Назначение битов регистра **SCON**

<table>
<tr><th>номер</th><th>мнемоника</th><th>описание</th></tr>
<tr><td>7</td><td>SM0</td><td rowspan="2">SM1, SM0 биты определяют режим работы последовательного порта.
<table>
<tr><th>SM1</th><th>SM0</th><th>Выбранный режим</th></tr>
<tr><td>0</td><td>0</td><td>режим 0: синхронный режим с фиксированной скоростью f_{core} /2.</td></tr>
<tr><td>0</td><td>1</td><td>режим 1: 8-битный асинхронный режим с настраиваемой скоростью передачи данных.</td></tr>
<tr><td>1</td><td>0</td><td>режим 2: 9-битный асинхронный режим с фиксированной скоростью f_{core} /32 или f_{core} /16.</td></tr>
<tr><td>1</td><td>1</td><td>режим 3: 9-битный асинхронный режим с настраиваемой скоростью передачи данных.</td></tr>
</table></td></tr>
<tr><td>6</td><td>SM1</td></tr>
<tr><td>5</td><td>SM2</td><td>Бит управления режимом приемопередатчика.
Устанавливается программно для запрета приема сообщения, в котором девятый бит имеет значение “0”. Такой режим используется при реализации сетевого протокола в многопроцессорной системе.</td></tr>
<tr><td>4</td><td>REN</td><td>Бит разрешения приема данных через последовательный порт.
Устанавливается программно, для разрешения приема.</td></tr>
<tr><td>3</td><td>TB8</td><td>9-ый бит передатчика последовательного порта.
Данные загруженные в TB8 передаются девятым битом. Это актуально для режима работы UART 2 и 3.</td></tr>
<tr><td>2</td><td>RB8</td><td>9-ый бит приемника последовательного порта.
В режиме работы 2 и 3 в этот бит загружается принятый девятый бит данных. В режиме 1 в этом бите хранится СТОП-бит.</td></tr>
<tr><td>1</td><td>TI</td><td>Флаг прерывания передатчика последовательного порта.
Устанавливается аппаратно при окончании передачи байта. Флаг должен сбрасываться (запись “0”) программно.</td></tr>
<tr><td>0</td><td>RI</td><td>Флаг прерывания приемника последовательного порта.
Устанавливается аппаратно при окончании приема байта. Флаг должен сбрасываться (запись “0”) программно.</td></tr>
</table>

2.1 Режим работы 0

Режим 8-битного сдвигового регистра. Для выбора этого режима работы следует в биты **SM0** и **SM1** записать логические нули (**SM1 = 0** и **SM0 = 0**). В этом режиме данные последовательно передаются и принимаются через вывод **RxD**. Вывод **TxD** используется для подачи тактовых импульсов. Передача инициируется любой инструкцией, записывающей данные в регистр **SBUF**. Передача байта данных начинается с младшего значащего бита. Прием начинается, когда бит разрешения приема установлен в единицу (**REN = 1**), а флаг прерывания приемника сброшен в ноль (**RI = 0**). Таким образом, в данном случае последовательный порт работает в синхронном режиме, поэтому у некоторых семейств микроконтроллеров такой последовательный порт называется не UART, а USART (Universal Synchronous-Asynchronous Receiver-Transmitter — универсальный синхронный - асинхронный приемопередатчик).

2.2 Режим работы 1

Асинхронный 8-битный режим с настраиваемой скоростью. Для выбора первого режима следует в бит **SM0** записать единицу, а в бит **SM1** — ноль (**SM1 = 0** и **SM0 = 1**). Вывод микросхемы **TxD** используется для передачи информации, вывод **RxD** — для приема.

Передача байта информации начинается с посылки СТАРТ-бита, за ним идут восемь информационных бит, заканчивается передача СТОП- битом. Таким образом, для передачи каждого байта информации используется 10 бит. Формирование **СТАРТ-бита** и **СТОП-бита** происходит автоматически.

Скорость передачи может устанавливаться Таймером 1 или Таймером 2, или комбинацией их обоих: один — для передачи, другой — для приема. Использование таймеров 1 и 2 характерно для всего семейства MCS-51/52, правда Таймер 2 есть только в старших моделях, но в микроконтроллере ADuC842 возможна синхронизация приемопередатчика UART от специального Таймера 3, что позволяет освободить таймеры общего назначения для выполнения других функций.

Передача данных начинается, когда в регистр **SBUF** программно записывается передаваемое число. Данные передаются до тех пор, пока на **TxD** не поступит **СТОП-бит**, после чего флаг прерывания передатчика (**TI**) установится в единицу, как это показано на рисунке ниже:

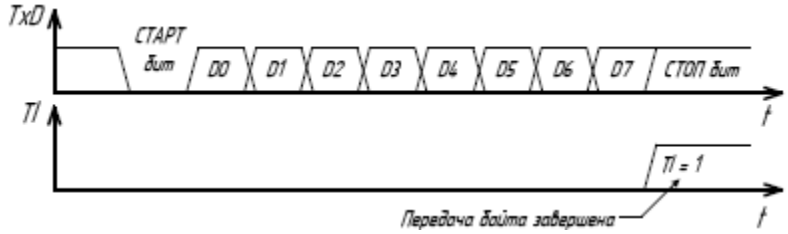


Рисунок 18 – Диаграмма передачи байта

Для разрешения приема данных через последовательный порт в бит **REN** следует записать логическую единицу. Прием байта данных начинается с приходом на линию **RxD СТАРТ-бита:** переход линии из высокого логического уровня в низкий. По окончанию приема всего байта флаг прерывания приемника **RI** устанавливается в единицу, и принятый байт может быть программно считан из буферного регистра **SBUF.**

2.3 Режим работы 2

Асинхронный 9-битный режим с фиксированной скоростью. Скорость передачи данных по умолчанию fcore/32, но если записать в бит **SMOD** регистра **PCON** логическую единицу, то скорость передачи будет удвоена: fcore/16. Для выбора второго режима следует в бит **SM0** записать логический ноль, а в бит **SM1** — единицу (**SM1 = 1** и **SM0 = 0**).

В этом режиме каждая передаваемая или принимаемая посылка состоит из одиннадцати бит: **СТАРТ-бит**, восемь информационных бит, девятый программируемый бит и **СТОП-бит**. Девятый бит часто используют в качестве бита паритета при контроле четности, хотя он может быть использован для любых других целей.

Прием и передача во втором режиме осуществляется аналогично первому режиму. Девятый бит при передаче программно записывается в бит **TB8** регистра **SCON**, при приеме девятый бит находится в **RB8** регистра **SCON.**

2.4 Режим работы 3

Асинхронный 9-битный режим с настраиваемой скоростью. Для выбора третьего режима следует в биты **SM0** и **SM1** записать логические единицы (**SM1 = 1** и **SM0 = 1**). В этом режиме последовательный порт UART работает также как в режиме 2, только скорость задается таймерами 1, 2 или 3, так же, как и в режиме 1.

Во всех четырех режимах передача данных начинается любой инструкцией, записывающей в регистр SBUF число. В режиме 0 прием начинается при условии **RI = 0** и **REN = 1**. Во всех остальных режимах прием начинается с приходом СТАРТ-бита при условии, что в бит **REN** записана логическая единица (**REN = 1**).

3. Расчет параметров синхронизации UART

По умолчанию последовательный порт микроконтроллера ADuC842 настроен на синхронизацию от Таймера 1. Скорость передачи UART определяется временем переполнения таймера:

$$BR = 132 \cdot TT, \qquad (1)$$

где TT – Время срабатывания таймера.

Таймер должен быть сконфигурирован для работы в режиме с автоперезагрузкой (режим 2). Для установки такого режима в старшие 4 бита регистра **TMOD** следует записать бинарную комбинацию 0010. В этом случае скорость передачи данных будет определяться по формуле:

$$BR = fcore32 \cdot 256 - TH1, \quad (2)$$

где: $fcore$– частота ядра микропроцессора, TH1 – содержимое регистра данных **TH1.**

Из формулы 2 легко найти значение регистра **TH1**, обеспечивающего требуемую скорость:

$$TH1 = 256 - fcore32 \cdot BR, \quad (3)$$

Результат вычисления должен быть округлен до ближайшего целого.

Используя Таймер 1 для синхронизации UART не всегда возможно получить требуемую частоту с достаточной точностью. Например, пусть при тактовой частоте ядра микропроцессора 2097кГц (значение для ADuC842 по умолчанию), требуется получить скорость передачи 19,2 кбит/с. По формуле 3 найдем значение TH1:

$$TH1 = 256 - 209732 \cdot 19.2 = 252{,}59 \approx 252.$$

Используя полученное значение TH1, рассчитаем реальную скорость передачи UART:

$$BR = 209732 \cdot 256 - 252 = 16.38 \text{ кбит/с}.$$

Реальная скорость на 14% меньше требуемой, а это значит, что передача данных невозможна.

Проблема была решена добавлением специального таймера 3, специализированного для высокоточной синхронизации UART в широком диапазоне частот. Кроме того, использование специализированного таймера высвобождает таймеры общего назначения для решения других различных задач.

Таймер 3, по сути, представляет собой набор настраиваемых делителей тактовой частоты ядра, структурная схема таймера изображена на рисунке ниже:

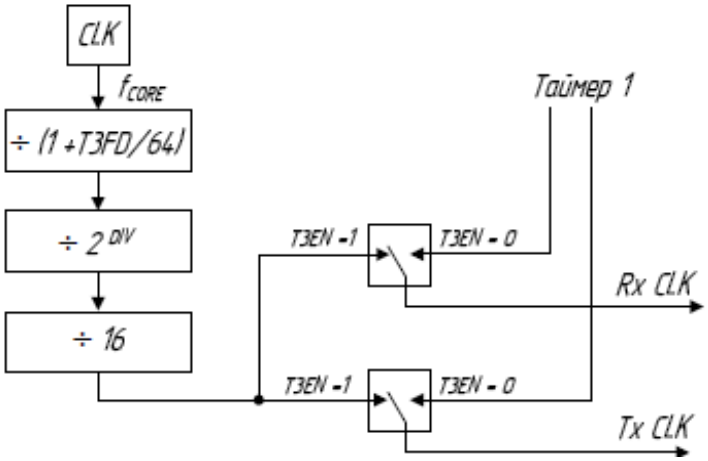


Рисунок 19 – Структурная схема Таймера 3

Для управления Таймером 3 предназначены два регистра специальных функций — **T3CON** и **T3FD**. Регистр **T3CON** содержит бит **T3EN**, при записи в него логической единицы синхронизация UART будет происходить от Таймера 3, в противном случае — от Таймера 1. Младшие три бита регистра **T3CON** определяют двоичный делитель **DIV**. Дробный коэффициент деления настраивается регистром **T3FD.**

T3CON – регистр конфигурации Таймером 3.

SFR адрес — **0x9E.**

Значение после подачи питания **0x00.**

Регистр не имеет побитовой адресации.

T3FD – регистр Таймера 3.

SFR адрес — **0x9D.**

Значение после подачи питания **0x00.**

Регистр не имеет побитовой адресации.

Таблица 6 – Назначение битов регистра T3CON

<table>
<tr><th>номер</th><th>мнемоника</th><th>описание</th></tr>
<tr><td>7</td><td>T3EN</td><td>Разрешение Таймера 3.
Когда бит установлен (T3EN = 1) синхронизация приемника и передатчика последовательного порта происходит от Таймера 3.
Когда бит сброшен (T3EN = 0) — синхронизация от Таймера 1.</td></tr>
<tr><td>6</td><td></td><td rowspan="3">Не используются</td></tr>
<tr><td>5</td><td></td></tr>
<tr><td>4</td><td></td></tr>
<tr><td>3</td><td></td><td></td></tr>
<tr><td>2</td><td>DIV2</td><td rowspan="3">Биты целочисленного делителя DIV.
<table><tr><th>DIV2</th><th>DIV1</th><th>DIV0</th><th>DIV</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>2</td></tr><tr><td>0</td><td>1</td><td>1</td><td>3</td></tr><tr><td>1</td><td>0</td><td>0</td><td>4</td></tr><tr><td>1</td><td>0</td><td>1</td><td>5</td></tr><tr><td>1</td><td>1</td><td>0</td><td>6</td></tr><tr><td>1</td><td>1</td><td>1</td><td>7</td></tr></table></td></tr>
<tr><td>1</td><td>DIV1</td></tr>
<tr><td>0</td><td>DIV0</td></tr>
</table>

Используя структурную схему Таймера 3 легко записать аналитическое выражение для расчета результирующей скорости последовательного порта:

$$BR = \frac{2f_{core}}{22^{DIV}-(T3FD+64)} \qquad (4)$$

где: $fcore$ — частота ядра микроконтроллера.

Значение делителя DIV можно определить по формуле 5, полученное значение следует округлить до целого вниз.

$$DIV = log^2 \frac{2f_{core}}{16BR} (5)$$

Дробный делитель T3FD можно найти по формуле 6, полученное значение следует округлить до ближайшего целого.

$$T3FD = \frac{2f_{core}}{2^{DIV-1}BR} - 64 \qquad (6)$$

Рассчитаем параметры конфигурации Таймера 3, для предыдущего примера: при тактовой частоте ядра микропроцессора 2097кГц, требуется получить скорость передачи 19,2 кбит/с.

$$DIV = log_2 \frac{2097}{16 \cdot 19{,}2} = 2{,}771 \approx 2$$

$$T3FD = \frac{2 \cdot 2097}{2^{2-1} \cdot 19{,}2} - 64 = 45{,}219 \approx 45$$

$$BR = \frac{2 \cdot 2097}{2^{2-1} \cdot (45+64)} = 19{,}239 \ \frac{\text{кбит}}{\text{с}}$$

Таким образом, ошибка установления скорости составляет всего 0.2%.

4. Особенности представления текстовой информации

В различных операционных системах для представления текстовой информации используют специальные наборы символов. Как правило, такой набор представляют в виде таблице, где

каждому символу соответствует бинарная последовательность длиною в один или несколько байт. В литературе подобную таблицу символов часто называют «кодировкой». На сегодняшний день наиболее распространенным является код ASCII (American Standard Code for Information Interchange — американский стандартный код для обмена информацией), который используется для внутреннего представления символьной информации в операционной системе MS DOS, в Блокноте операционной системы Windows, а также для кодирования текстовых файлов в Интернет.

Поскольку первоначально ASCII предназначался для обмена информации, в нѐм, кроме информационных символов, используются символы-команды для управления связью. В приведенной таблице такие символы показаны многоточием.

Таблица 7 – Таблица ASCII

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
1.	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
2.		!	"	#	$	%	&	'	(	)	*	+	,	–	.	/
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	…

Таблица кодов содержит 8 столбцов и 16 строк, каждая строка и столбец пронумерованы в шестнадцатеричной системе счисления. Шестнадцатеричное представление ASCII-кода складывается из номера столбца и номера строки, в которых располагается символ, при этом номер строки образуют первую цифру (старшие четыре бита), а номер столбца вторую цифру (младшие 4 бита). Так, например, ASCII-код символа ―E есть число 0x45, а символа “\“ — 0x5C.

Легко заметить, что в приведенной таблице представлено 128 символов, притом, что один символ кодируется байтом — восьмью битами. Дело в том, что верхние значения (128—255) могут занимать различные дополнительные символы, например, набор русского алфавита, это зависит от конкретного типа кодировки.

Задание

Записать в тетрадь особенности работы таймера в различных режимах, а также формулы для определения скорости последовательного порта, знать назначение битов, хранящихся в регистрах SCON и T3CON.

Форма представления результата:

Отчет должен содержать:

1. Цель работы.
2. Структурная схема таймера 3.
3. Состав регистров SCON и T3CON.

Практическая работа №6
Изучение схемы подключения ЖКИ к МК ADuC842

Формируемые компетенции:

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы: изучить особенности организации работы с ЖКИ.

Выполнив работу, Вы будете:

уметь:

- выбирать микроконтроллер/микропроцессор для конкретной системы управления

Материальное обеспечение:

не требуется.

Теоретические сведения

1. Устройство и принцип работы символьного жидкокристаллического индикатора

В настоящее время в микропроцессорных системах для отображения широко используют жидкокристаллические индикаторы (ЖКИ). Условно все ЖКИ можно разделить на две категории: символьные, или знакосинтезирующие, и графические. Графические индикаторы представляют собой матрицу из **m** строк и **n** столбцов, на пересечении которых находятся пиксели. Пиксель представляет собой неделимый объект прямоугольной или круглой формы, обладающий определѐнным цветом; пиксель – наименьшая единица растрового изображения. Если на определенный столбец и строку подать электрический сигнал, то пиксель на их пересечении изменит свой цвет. Подавая группу сигналов на столбцы и строки можно формировать по точкам произвольное графическое изображение. Так работает графический ЖКИ. В символьном же ЖКИ матрица пикселей разбита на подматрицы, каждая подматрица предназначена для формирования одного символа: цифры, буквы или знака препинания. Как правило, для формирования одного символа используют матрицу из восьмистрок и пяти столбцов. Символьные индикаторы бывают одно-, двух- и четырехстрочными.

Для упрощения взаимодействия микропроцессорной системы и ЖКИ используют специализированную микросхему – контроллер (драйвер) ЖКИ. Он управляет пикселями жидкокристаллического дисплея и интерфейсной частью индикатора. Обычно такой контроллер входит в состав индикатора. В целом жидкокристаллический индикатор представляет собой печатную плату, на которой смонтирован сам дисплей, контроллер и необходимые дополнительные электронные компоненты. Внешний вид ЖКИ показан на рисунке ниже.

Рисунок 20 – Внешний вид жидкокристаллического индикатора

В учебном стенде LESO1 использован двухстрочный восьмисимвольный ЖКИ. Структурная схема показана на рисунке 21.

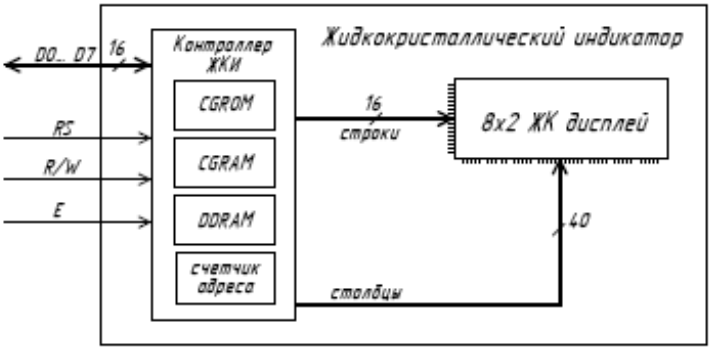


Рисунок 21 – Структурная схема ЖКИ

В состав контроллера ЖКИ входят три вида памяти: **CGROM**, **CGRAM**, **DDRAM**. Когда микроконтроллер передает в контроллер ЖКИ ASCII-коды символов, то они записываются в **DDRAM** (Display data RAM – ОЗУ ASCII-кодов отображаемых символов), такую память называют **видеопамятью** или **видеобуфером.** Видеобуфер в символьных индикаторахобычно содержит 80 ячеек памяти – больше, чем число знакомест дисплея. У двухстрочных индикаторов ячейки с адресами от 0x00 и до 0x27 отображаются на верхней строке дисплея, а ячейки с адресами 0x40 … 0x67 – на нижней строке. Смещая видимое окно дисплея относительно DDRAM, можно отображать на дисплее различные области видеопамяти. Сдвиг окна индикатора относительно видеобуфера для верхней и нижней строк происходит синхронно, как это показано на рисунке 22. Курсор будет виден на индикаторе только в том случае, если он попал в зону видимости дисплея (и если предварительно была подана команда отображать курсор).

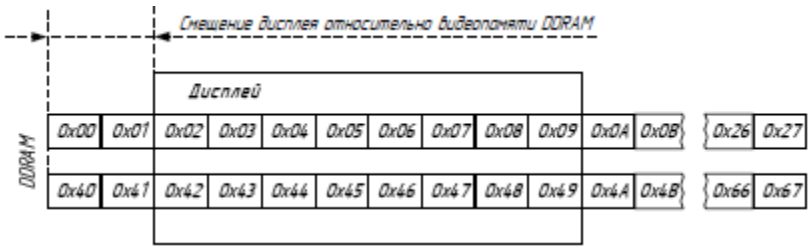


Рисунок 22 – Отображение символов из видеобуфера

Матрицы начертания символов хранятся в памяти знакогенератора. Память знакогенератора включает в себя **CGROM** (Character generator ROM – ПЗУ знакогенератора), в которую на заводе-изготовителе загружены начертания символов таблицы ASCII. Содержимое CGROM изменить нельзя. Для того, чтобы пользователь смог самостоятельно задать начертание нужных ему символов, в знакогенераторе имеется специальное ОЗУ – **CGRAM** (Character generator RAM). Под ячейки CGRAM отведены первые (младшие) 16 адресов таблицы кодов.

Схема подключения ЖКИ к микроконтроллеру ADuC842 показана ниже на рисунке:

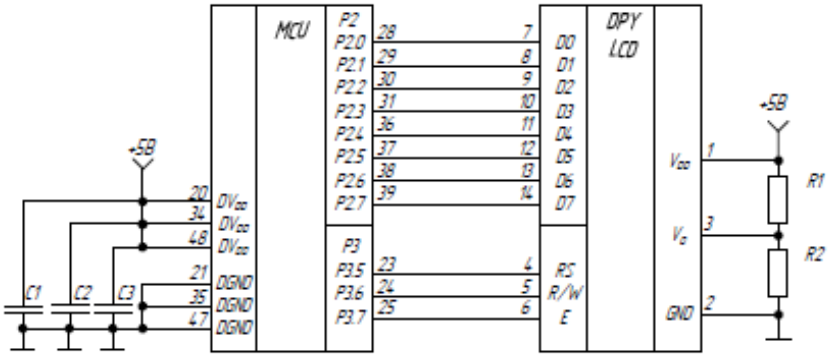


Рисунок 23 – Схема подключения ЖКИ к микроконтроллеру

Интерфейс подключения – параллельный. Для соединения индикатора с микроконтроллером используется 11 линий — восемь для передачи данных (**D0 - D7**) и три линии управления. Линия **RS** служит для сообщения контроллеру индикатора о том, что именно передается по шине: команда или данные (**RS = 1** — данные, **RS = 0** — команда). По линии **E** передается строб- сигнал, сопровождающий запись или чтение данных: по переходу сигнала на линии **E** из 1 в 0 осуществляется запись данных во входной буфер микроконтроллера индикатора. Запись информации в ЖКИ происходит по спаду этого сигнала. Потенциал на управляющем выводе **R/W** (**Read/Write**) задает направление передачи информации, при **R/W = 0** осуществляется запись в память индикатора, при **R/W = 1** – чтение из нее. Еще три линии предназначены для подачи питающего напряжения (**VDD, GND**) и напряжения смещения, которое управляет контрастностью дисплея.

Диаграммы передачи данных от управляющего микроконтроллера к контроллеру ЖКИ и от контроллера ЖКИ в управляющий микроконтроллер показаны на рисунках 24 и 25 соответственно.

После приема информации контроллеру ЖКИ требуется некоторое время на выполнение команд, в это время управляющий контроллер не должен давать следующую команду или пересылать данные.

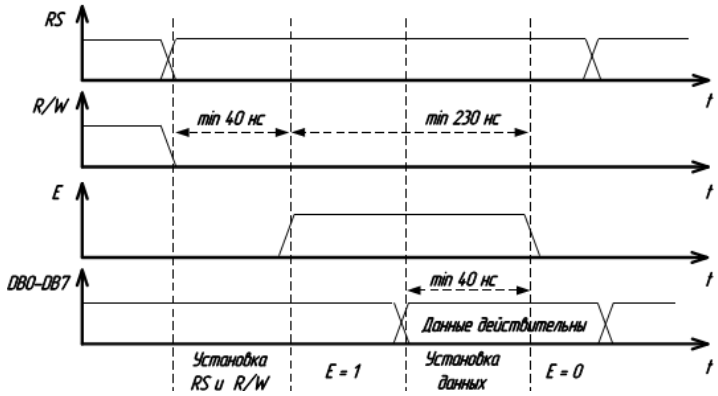


Рисунок 24 – Диаграмма передачи информации контроллеру ЖКИ

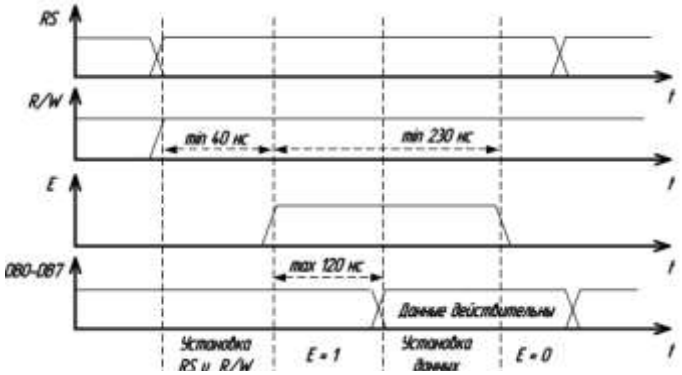


Рисунок 25 – Диаграмма чтения информации из контроллера ЖКИ

В таблице 8 приведены команды контроллера ЖКИ и время, необходимое для выполнения этих команд. Для того чтобы можно было определить, когда ЖКИ закончит свои внутренние операции, контроллер ЖКИ содержит специальный флаг занятости – **BUSY-флаг** (**BF**). Если контроллер занят выполнением внутренних операций, то **BF** установлен (**BF = 1**), если же контроллер готов принять следующую команду, то **BF** сброшен (**BF = 0**). Более простой способ организации обмена заключается в том, что управляющий микроконтроллер, зная, сколько времени требуется ЖКИ на обработку той или иной команды, после каждой передачи информации ждет соответствующее время.

Таблица 8 – Команды контроллера ЖКИ

Команда	Код										Описание	Время исполнения команды
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0		
Очистка дисплея	0	0	0	0	0	0	0	0	0	1	Записывает код 0x20 (пробел) во все ячейки DDRAM, устанавливает счетчик адреса DDRAM в 0x00.	1,5 мс
Возврат в начальную позицию	0	0	0	0	0	0	0	0	1	x	Устанавливает счетчик адреса DDRAM в 0x00 и возвращает курсор в начальную позицию. Содержимое DDRAM не изменяется.	1,5 мс
Установка режима ввода	0	0	0	0	0	0	0	1	I/D	SH	Задает направление перемещения курсора (I/D) и разрешает сдвиг сразу всех символов (SH).	38 мкс
Включение-выключение дисплея	0	0	0	0	0	0	1	D	C	B	Устанавливает/ отключает биты, отвечающие за включения дисплея (D), отображение курсора (C), мерцание курсора (B).	38 мкс
Сдвиг курсора или видимой области дисплея	0	0	0	0	0	1	D/C	R/L	x	x	Бит D/C определяет то, что будет перемещаться – видимая область дисплея или курсор (при **D/C = 1** перемещается видимая область, при **D/C = 0** – курсор), R/L задает направление перемещения. DDRAM не изменяется	38 мкс
Начальные установки	0	0	0	0	1	DL	N	F	x	x	Определяет разрядность шины интерфейса (**DL = 1** –8-бит, **DL = 0** 4-бита), количества строк на дисплее (**N = 1** – две строки, **N = 0** – одна строка) и размера символов (**F =1** – 5×11 точек, **F =0** 5×8 точек).	38 мкс
Установка адреса CGRAM	0	0	0	1	A5	A4	A3	A2	A1	A0	Установка счетчика адреса CGRAM	38 мкс
Установка адреса DDRAM	0	0	1	A6	A5	A4	A3	A2	A1	A0	Установка счетчика адреса DDRAM	38 мкс
Чтение BF и счетчика адреса	0	1	BF	A6	A5	A4	A3	A2	A1	A0	Если **BF = 1** то контроллер ЖКИ выполняет внутреннюю операцию. A6-A0 – текущее значение адреса DDRAM.	0
Запись данных в RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Запись данных в ОЗУ (DDRAM или CGRAM)	38 мкс
Чтение данных из RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Чтение данных из ОЗУ (DDRAM или CGRAM)	38 мкс

Задание

Зарисовать в тетрадь структурную схему ЖКИ и схему подключения ЖКИ к МК, временные диаграммы обмена информацией между ЖКИ и МК.

Форма представления результата:

Отчет должен содержать:

1. Цель работы.
2. Структурная схема ЖКИ.
3. Схема подключения ЖКИ к МК
4. Временные диаграммы работы с ЖКИ

Лабораторнаяработа № 1
Работа в среде программирования и отладки Keil-C

Формируемые компетенции:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем.

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы: изучить интегрированную среду программирования Keil-C.

Выполнив работу, Вы будете:

уметь:

- составлять программы на языке ассемблера для микропроцессорных систем;
- производить тестирование и отладку МПС;

Материальное обеспечение:

1. Компьютер с лицензионным программным обеспечением.
2. Мультимедиа проектор.
3. Лабораторные стенды «LESO1»

Задание:Изучить следующие вопросы:

1. Построение файловой системы персонального компьютера.
2. Изучить правила пользования текстовым редактором.
3. Создание и сохранение текстовых файлов.
4. Изучить порядок создания программного проекта в интегрированной среде программирования Keil-C.
5. Изучить настройку свойств программного проекта в интегрированной среде программирования Keil-C.
6. Изучить использование окна управления программным проектом в интегрированной среде программирования Keil-C.
7. Изучить методы трансляции отдельных файлов в интегрированной среде программирования Keil-C.
8. Изучить методы трансляции программного проекта в интегрированной среде программирования Keil-C.
9. Изучить работу отладчика программ в интегрированной среде программирования Keil-C.

Краткие теоретические сведения:

1 Написание программы

В настоящее время программа пишется на одном из языков программирования в виде текстового файла. Это означает, что для написания программы можно воспользоваться любым текстовым редактором. Для того чтобы программа-транслятор могла преобразовать исходный текст программы в машинные коды микропроцессора, этот текст программы должен быть записан с использованием символов ASCII или ANSI таблиц. К сожалению, некоторые текстовые редакторы для увеличения возможностей редактирования используют для записи текстов формат rtf, или свои собственные форматы (например, редактор WORD). Такие текстовые файлы не понимаются программами-трансляторами и, следовательно, не могут быть использованы для записи исходного текста программы.

2 Использование интегрированной среды программирования

Для облегчения процесса разработки программы часто используются интегрированные среды программирования. В состав интегрированной среды программирования включается определенный набор программных средств: редактор исходного текста, трансляторы с выбранного языка программирования, редакторы связей, загрузчики и так далее. Редактор текстов обычно является первой программой, которую приходится использовать в процессе разработки программ: благодаря нему пользователь получает возможность набирать исходные тексты программ, написанных на ассемблере или на каком-нибудь языке высокого уровня и сохранять их на жѐстком диске.

3 Работа с текстовым редактором интегрированной среды программирования Keil-C

Работа с текстовыми файлами начинается с создания нового файла. Создать текстовый файл можно несколькими способами. Первый способ – воспользоваться главным меню, как показано на рисунке 26.

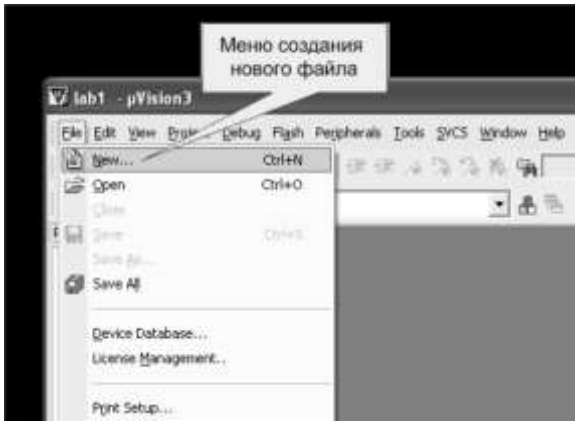


Рисунок 26 – Создание нового файла через главное меню

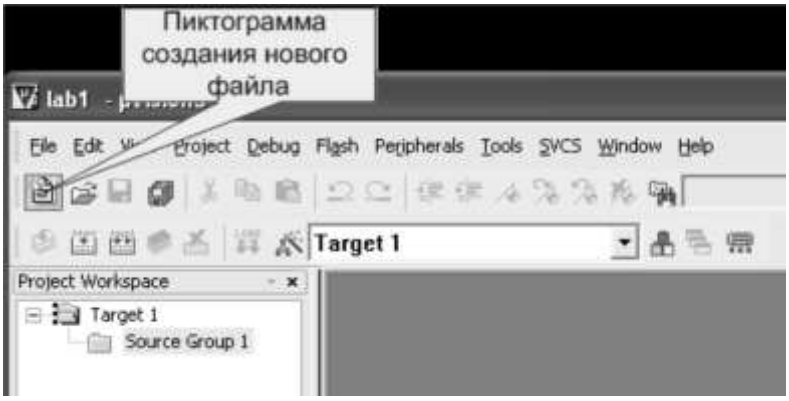


Рисунок 27 – Создание нового файла при помощи пиктограммы

Второй способ – использовать быстрые клавиши Ctrl+N. И третий способ – это нажать на пиктограмму создания нового файла, как показано на рисунке 27. После выполнения этих действий открывается окно текстового редактора, в котором можно вводить исходный текст программы.

Внешний вид программы с открытым окном текстового редактора показан на рисунке 28. Ввод программы производится с клавиатуры. Стирание одиночных ошибочно введѐнных символов возможно при помощи кнопок ―Delete‖ и ―Backspace‖

После создания окна нового файла можно вводить исходный текст программы, используя клавиатуру. Набрав исходный текст программы в окне текстового редактора, файл необходимо сохранить на диске компьютера. Для этого можно воспользоваться меню файл, как это показано на рисунке 28:

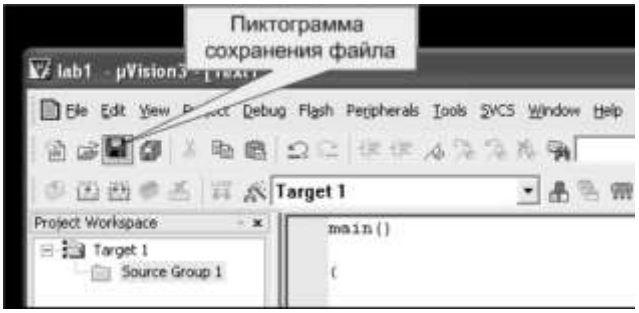


Рисунок 28 – Сохранение файла через главное меню

Рисунок 29 – Сохранение файла при помощи пиктограммы

Второй способ – использовать быстрые клавиши Ctrl+S. И третий способ – это нажать на пиктограмму сохранения файла, как показано на рисунке 29. При написании программ часто требуется копировать участки программ из одного файла в другой. Для этого в текстовом редакторе открываются оба файла. Затем необходимый участок текста выделяется при помощи мыши или клавиатуры. Для выделения строк нажимается левая кнопка мыши в начале выделяемого фрагмента и, не отпуская еѐ, курсор мыши перемещается в конец этого фрагмента. Для выделения столбцов производятся те же действия, но, кроме того, нажимается кнопка ―Alt‖ на клавиатуре.

После выделения необходимого фрагмента текста, этот фрагмент копируется в буфер обмена. Скопировать можно, щѐлкнув мышью по пиктограмме копирования, как это показано на рисунке 30, кроме того удобно копировать в буфер обмена комбинацией клавиш ―Ctrl+C‖.

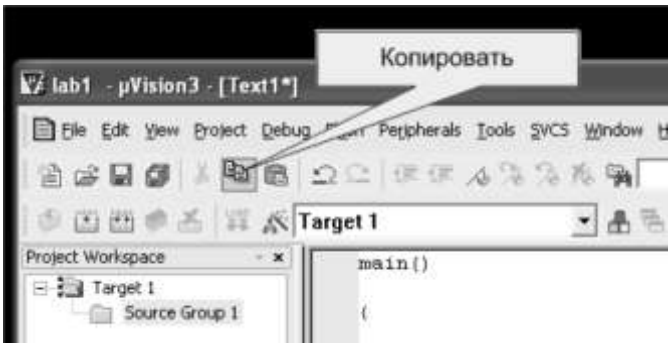


Рисунок 30 – Копирование выделенного фрагмента в буфер обмена при помощи пиктограммы

Теперь можно переключиться в окно редактирования файла, куда нужно поместить скопированный текстовый фрагмент при помощи главного меню, как это показано на рисунке 31, и вставить этот фрагмент перед текстовым курсором, вставку также можно осуществить нажатием ―Ctrl+V‖. Вставка фрагмента из буфера обмена может быть произведена либо из меню ―Edit‖, либо при помощи пиктограммы ―Paste‖ как это показано на рисунке 32. Фрагмент исходного текста будет вставлен в то место набираемого текста, где в настоящее время находится курсор.

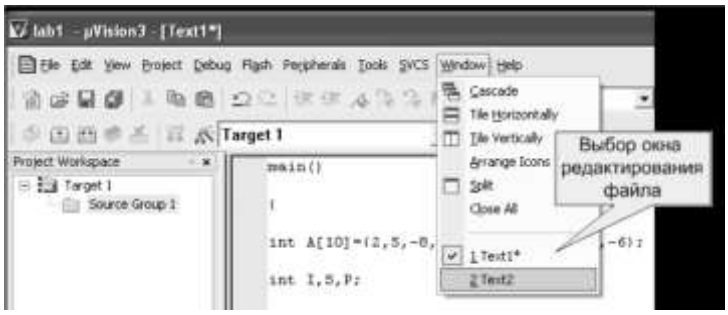


Рисунок 31 – Выбор окна редактирования файла

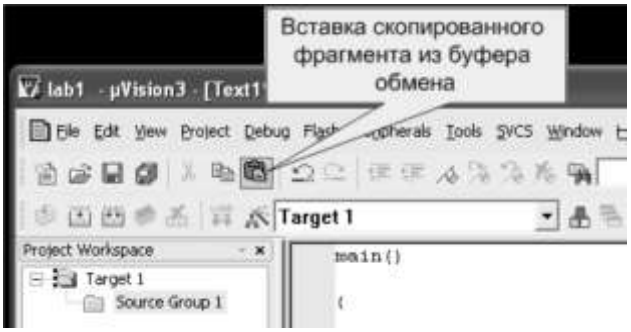


Рисунок 32 – Вставка скопированного фрагмента из буфера обмена

Напомним, что копирование участка исходного текста программы эквивалентно набору этого текста программы с клавиатуры и поэтому файл, в который производилось копирование текста, необходимо сохранить на диске любым из методов, описанных ранее.

4 Создание программных проектов

4.1 Разработка программных средств

На первом шаге разработки программных средств формулируются технические требования к системе, и составляется блок-схема процесса решения нужных задач, которая обеспечит реализацию заданных требований. Блок-схема должна быть надлежащим образом структурирована, чтобы гарантировалась высокая эффективность логики программ.

Для того чтобы было легко ориентироваться в файлах на компьютере, каждую задачу помещают в отдельную директорию (папку). Написание программы тоже следует начинать в отдельной папке, тем более что как будет показано далее, даже простейшая программа состоит из нескольких файлов.

В большинстве случаев программа состоит из нескольких программных модулей. Использование в составе одной программы нескольких программных модулей позволяет увеличить скорость трансляции программ, поручать написание программных модулей различным программистам, увеличивать понятность программ.

Принцип разбиения единой программы на программные модули заключается в том, что для реализации работы с отдельными узлами аппаратуры пишутся отдельные подпрограммы, которые относительно слабо связаны с остальными частями программы. Эти подпрограммы можно выделить в отдельную программу, которую можно хранить в отдельном файле, и транслировать отдельно от остальной программы. Часто одни и те же модули могут входить в состав нескольких программ, выполняющих различные задачи, но использующие при этом одни и те же устройства.

В качестве примера можно назвать работу с клавиатурой, индикацию различных видов информации, работу с последовательными портами, с АЦП, с ЦАП. Каждое из этих устройств может обслуживаться отдельными программными модулями. Этот список можно продолжать и далее, но для составления представления о программных модулях этого достаточно.

Программные проекты можно создавать, и поддерживать вручную, но в последнее время обычно используются различные системы поддержки разработок. Это создаѐт ряд дополнительных преимуществ.

Часто программа пишется, и отлаживается на одной аппаратуре (например, на оценочных платах, предлагаемых фирмами изготовителями микросхем), а используется на другой. При этом программа при отладке незначительно отличается от программы, которая будет использоваться в реальной аппаратуре. Для этого в составе программного проекта создаются назначения проекта. В качестве примера назначений программного проекта можно назвать отладку и реализацию, а также версии программы.

4.2 Использование системы поддержки разработок

Для облегчения процесса разработки программы часто используются системы поддержки разработок. В состав систем поддержки разработок включается определенный набор программных средств: редакторы текстов, ассемблер, редакторы связей, компиляторы, загрузчики и тому подобное. Каждая из этих программ создаѐт свой файл (или свои файлы) на диске компьютера.

Для того чтобы не запутаться в этих файлах при разработке программы, все эти файлы размещаются в своей, отдельной директории. Имя этой директории обычно назначают по имени программно-аппаратного проекта. Например: измеритель характеристик транзисторов или цифровой осциллограф.

4.3 Создание программного проекта в интегрированной среде программирования Keil-C

Работа с программными проектами начинается с создания нового файла проекта. Для создания файла проекта в интегрированной среде разработки программ можно воспользоваться главным меню, как показано на рисунке 33.

После создания новой директории и нового файла программного проекта, интегрированная среда программирования предлагает выбрать конкретную микросхему из семейства MCS-51, как это показано на рисунке 34.

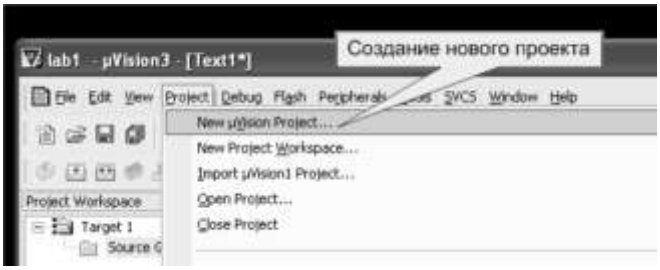


Рисунок 33 – Создание нового программного проекта

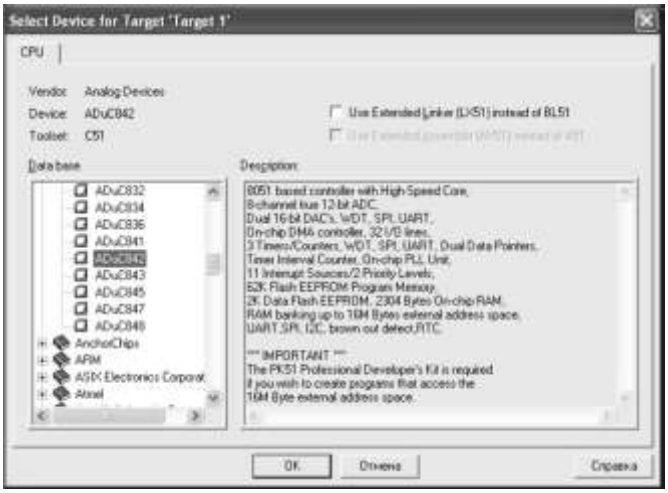


Рисунок 34 – Диалоговое окно выбора микросхемы для программного проекта

При этом в интегрированной среде программирования окно менеджера проекта приобретает вид, показанный на рисунке 35. Название назначения программного проекта можно изменить, щёлкнув манипулятором ―мышь по названию назначения программного проекта в окне менеджера проекта (Например: отладка, реализация или сопровождение). Точно так же можно изменить название устройства в составе программного проекта (Например: носимая радиостанция, автомобильная радиостанция, стационарная радиостанция или базовая радиостанция).

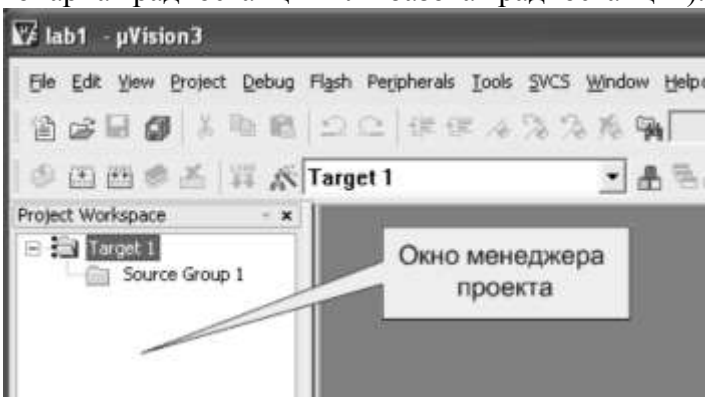


Рисунок 35 – Внешний вид окна менеджера проекта после создания программного проекта

4.4 Настройка свойств программного проекта в интегрированной среде программирования Keil-C

После создания программного проекта в интегрированной среде программирования keil-c конечным файлом трансляции является абсолютный файл. Для загрузки в микросхему обычно используется HEX файл. Для создания этого файла необходимо включить соответствующую опцию в свойствах программного проекта.

Изменить свойства программного проекта можно несколькими способами. Первый способ – воспользоваться главным меню, как показано на рисунке 36.

Второй способ – это нажать на кнопку изменения свойств программного проекта, как показано на рисунке 37.

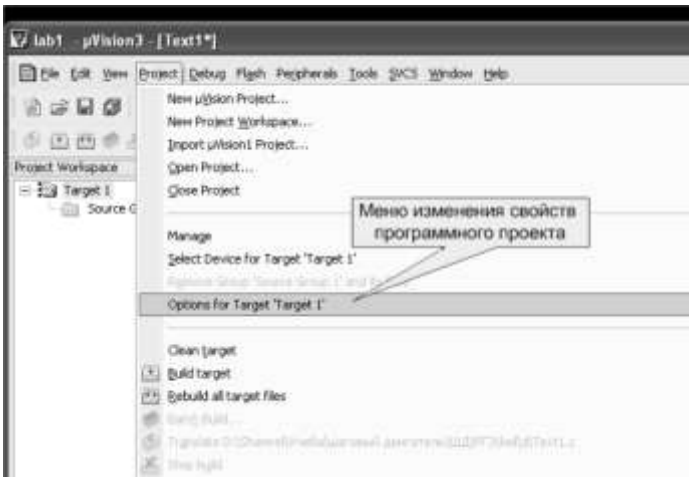


Рисунок 36 - Изменение свойств программного проекта

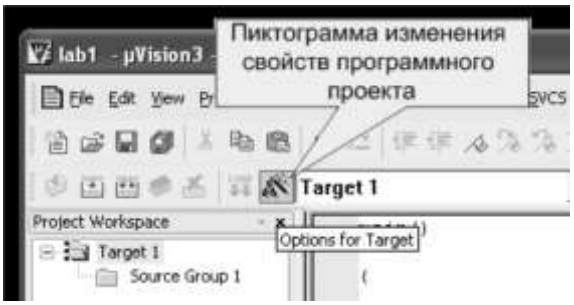


Рисунок 37 – Изменение свойств программного проекта при помощи пиктограммы

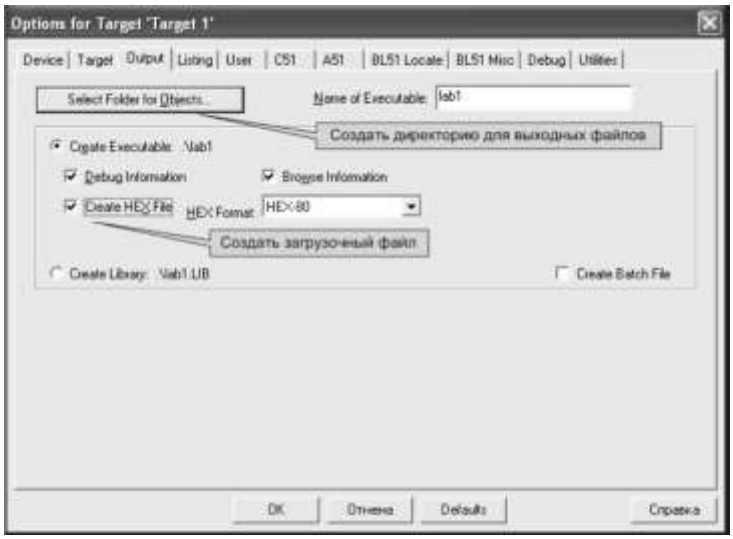
При этом на экране компьютера появляется диалоговое окно изменения свойств программного проекта как показано на рисунке 38. В этом окне необходимо ввести параметры внешней памяти программ и памяти данных.

Затем необходимо установить выходные параметры программного проекта. Для этого открываем закладку выход (Output), как это показано на рисунке 39. В этой закладке убеждаемся, что установлена галочка создания выходного загрузочного файла в .hex формате. Для того чтобы не загромождать директорию проекта файлами объектных кодов можно создать отдельную директорию. Например, с названием OBJ. Новая директория может быть создана после нажатия на кнопку ―Select Folder for Object‖.

Рисунок 38 – Диалоговое окно настройки свойств программного проекта

Точно так же можно создать директорию (папку) для файлов листингов. Для этого необходимо выбрать закладку —Listing. В файлах листингов помещается информация об ошибках, ассемблерный код и соответствующий ему машинный код программного модуля. Использование листингов позволяет оптимизировать программу, а при работе без интегрированной среды программирования и находить синтаксические ошибки программы. Отметим, что создание файлов листингов замедляет процесс трансляции. Обычно имя для папки листингов выбирают LST.
Необходимость создания отдельных папок для листингов и объектных кодов возникает при большом количестве программных модулей, а значит при большом количестве файлов в одной папке. Обычно директории для листингов и для объектных кодов располагают внутри папки программного проекта, где содержатся исходные тексты программы.

Для настройки параметров компиляции выбирается закладка —C51. В этой закладке настраивается уровень оптимизации транслируемого программного модуля и цель оптимизации (по скорости работы программы или по размеру выходного файла). Кроме того, в этой закладке заносится адрес векторов прерывания.После настройки свойств программного проекта в диалоговом окне, это окно закрывается нажатием кнопки —OK.



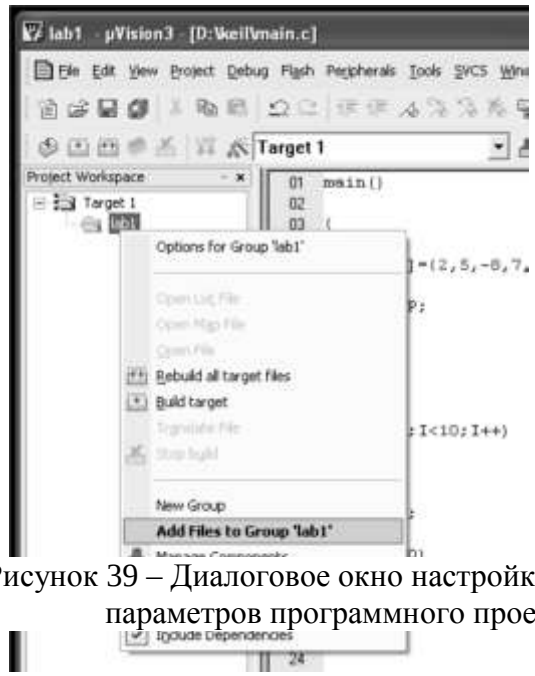


Рисунок 39 – Диалоговое окно настройки выходных параметров программного проекта

Рисунок 40 – Всплывающее меню менеджера проектов с выбранной опцией добавления файлов к программному проекту

Теперь можно подключать к программному проекту файлы с исходным текстом программных модулей. Для этого можно щѐлкнуть правой кнопкой мыши по значку группы файлов в окне менеджера проектов, как это показано на рисунке 40, и выбрать опцию добавления файлов к программному проекту. Добавляемые файлы должны быть предварительно созданы. Если файлом является программа на языке С, то этот файл должен иметь расширение .С. Например, LAB1.c. Если проект состоит из нескольких программных модулей, опцию добавления файлов следует повторить соответствующее число раз.

4.5 Работа с программным проектом в интегрированной среде программирования Keil-C

После завершения создания программного проекта можно открыть исходные тексты программных модулей, щѐлкнув левой кнопкой мыши на названии соответствующего файла в окне менеджера проектов.

Точно так же можно переключаться между программными модулями проекта, используя окно менеджера проектов.

В случае необходимости можно выключать окно менеджера программных проектов, но обычно это окно удобно при написании программы.

Если окно менеджера проектов отключено, то переключаться между модулями можно, используя меню ―window.

5 Указания по трансляции программ и программных проектов

5.1 Трансляция программных модулей

Как уже указывалось, программный проект может состоять из нескольких файлов. В программном проекте, состоящем из нескольких файлов, появляется возможность нескольких видов трансляции. Прежде всего, можно оттранслировать только один файл, не транслируя остальные файлы программного проекта. Это позволяет обнаружить, и исправить все синтаксические ошибки в отдельном программном модуле.

При трансляции каждого программного модуля на жѐстком диске формируются перемещаемый файл в объектном формате, который затем будет использоваться для создания загрузочного файла программного проекта. Этот файл называется объектный модуль. Одновременно с объектным модулем на диске формируется файл листинга программного модуля, в который помещается исходный текст программного модуля и сообщения о синтаксических ошибках.

При трансляции программного модуля, написанного на языке программирования ассемблер, в листинг помещаются машинные коды команд процессора, и их адрес относительно начала программного модуля. При трансляции исходного текста программы, написанной на языке программирования высокого уровня, таком как С программа транслятор может быть настроена так, что она будет создавать файл с текстом исходного модуля, написанного на ассемблере или помещать этот текст в листинг программного модуля. Таким образом, появляется возможность писать, и отлаживать программу на языке высокого уровня, а затем переводить еѐ на язык программирования ассемблер и оптимизировать еѐ вручную.

5.2 Связывание объектных модулей и получение загрузочного файла

После того, как оттранслированы без ошибок все программные модули, и тем самым получены файлы объектных модулей, производится трансляция всего программного проекта (связывание объектных модулей). При этом на диске формируются абсолютный и загрузочный файлы программного проекта. Для контроля ошибок связывания формируется файл листинга редактора связей с расширением m51.

Если обнаруживаются ошибки связывания, то абсолютный и загрузочный файлы программного проекта не формируются. В этом случае необходимо изменить исходный текст программного модуля из-за которого возникла ошибка связывания, оттранслировать его и снова попытаться произвести трансляцию программного проекта (связывание объектных модулей).

После получения загрузочного модуля можно начинать отладку программы.

5.3 Трансляция программных проектов

Интегрированная среда программирования позволяет максимально облегчить трансляцию программных проектов. Так как параметры программного проекта уже настроены, то для трансляции исходного текста программного модуля достаточно загрузить исходный текст этого программного модуля в окно текстового редактора. Это можно сделать одним из способов рассмотренных ранее.

После загрузки исходного текста программного модуля достаточно нажать на кнопку трансляции программного модуля, как это показано на рисунке 41.

Ещѐ один способ трансляции программного модуля, это воспользоваться главным меню, как это показано на рисунке 42.

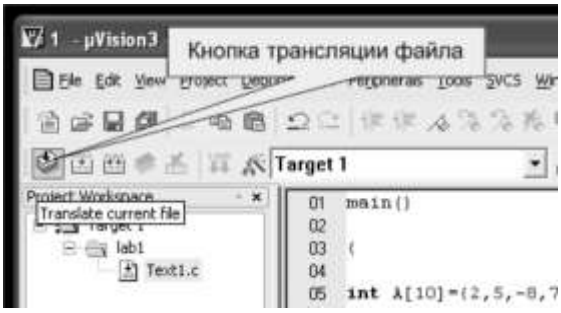


Рисунок 41 – Трансляция программного модуля при помощи кнопки трансляции файла

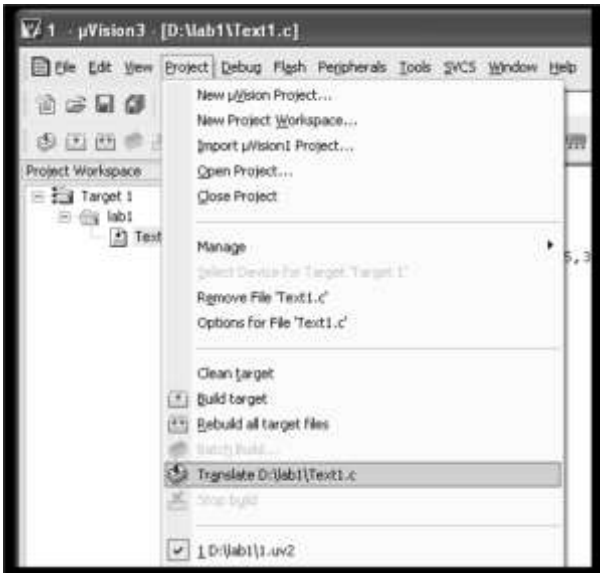


Рисунок 42 – Трансляция программного модуля при помощи главного меню

Надо отметить, что в составе интегрированной среды программирования для поиска синтаксических ошибок удобнее пользоваться не файлом листинга, а окном _build‘(расположено внизу рабочей области), где выводятся все сообщения об ошибках. При этом если дважды щѐлкнуть мышью по сообщению об ошибке в окне _build‘, то в окне текстового редактора будет выделена строка программы, где была обнаружена данная ошибка.

Трансляция программного модуля и получение загрузочного файла в интегрированной среде программирования производится нажатием кнопки _Build target‘, как это показано на рисунке 43.

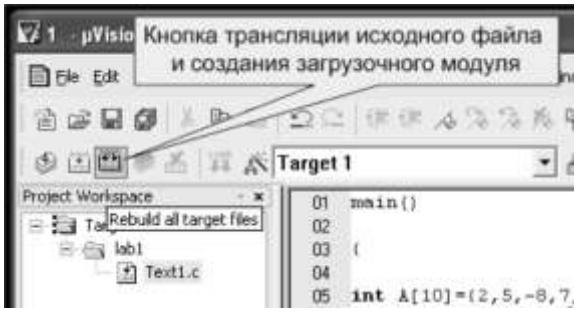


Рисунок 43 – Трансляция программного модуля проектапри помощи кнопки _Build target

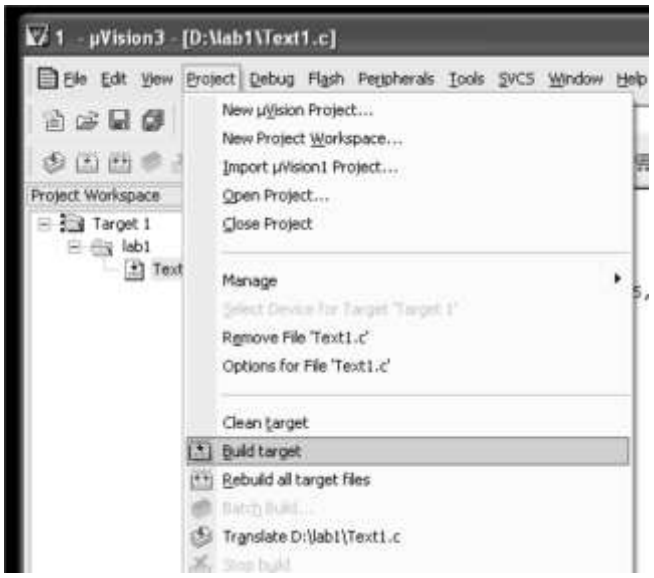


Рисунок 44 – Трансляция программного модуля при помощи главного меню

Ещѐ один способ трансляции программного проекта в интегрированной среде программирования, это воспользоваться главным меню, как это показано на рисунке 44, но удобнее всего нажать на клавиатуре клавишу —F7.

Если же необходимо оттранслировать все программные модули вне зависимости имеются объектные модули или нет, и получить загрузочный файл, то нажимается кнопка _Rebuild all target files‘ или выбирается соответствующее меню.

6 Указания по отладке программ во встроенном отладчике программ

6.1 Способы отладки программ

Отладка программ заключается в проверки правильности работы программы и аппаратуры. Программа, не содержащая синтаксических ошибок, тем не менее, может содержать логические ошибки, не позволяющие программе выполнять заложенные в ней функции. Логические ошибки могут быть связаны с алгоритмом программы или с неправильным пониманием работы аппаратуры, подключѐнной к портам микроконтроллера.

Встроенный отладчик позволяет отладить те участки кода программы, которые не зависят от работы аппаратуры, не входящей в состав микросхемы микроконтроллера. Для отладки программ обычно применяют три способа:

1. Пошаговая отладка программ с заходом в подпрограммы;
2. Пошаговая отладка программ с выполнением подпрограммы как одного оператора;
3. Выполнение программы до точки останова.

Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор.

Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ.

6.2 Использование встроенного отладчика программ

Вызов встроенного отладчика удобнее всего осуществить, нажав на кнопку отладчика на панели инструментов _file‘ как показано на рисунке 45 или воспользоваться быстрой кнопкой —Ctrl+F5‖

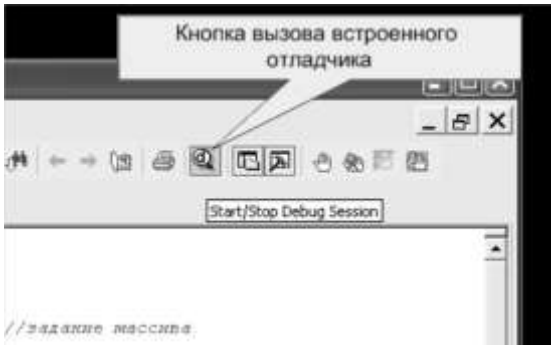


Рисунок 45 – Вызов встроенного отладчика с использованием кнопки на панели —file

После этого внешний вид интегрированной среды программирования принимает вид, показанный на рисунке 47. В верхней части программы появляется дополнительная панель инструментов отладчика программ (рисунок 46). В нижней части программы появляется окно просмотра памяти контроллера и окно контроля переменных —Watch.

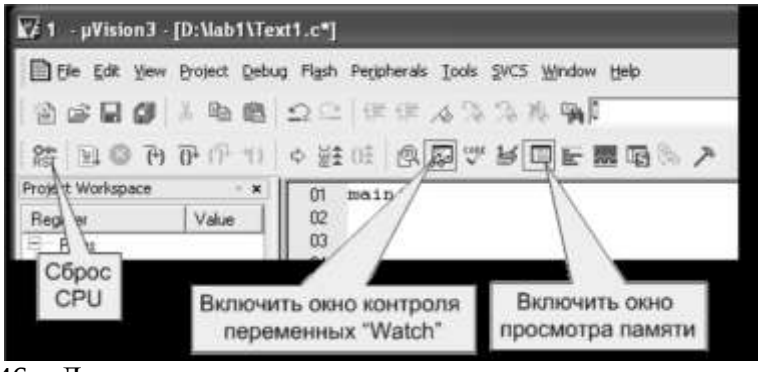


Рисунок 46 – Дополнительная панель инструментов отладчика программ.

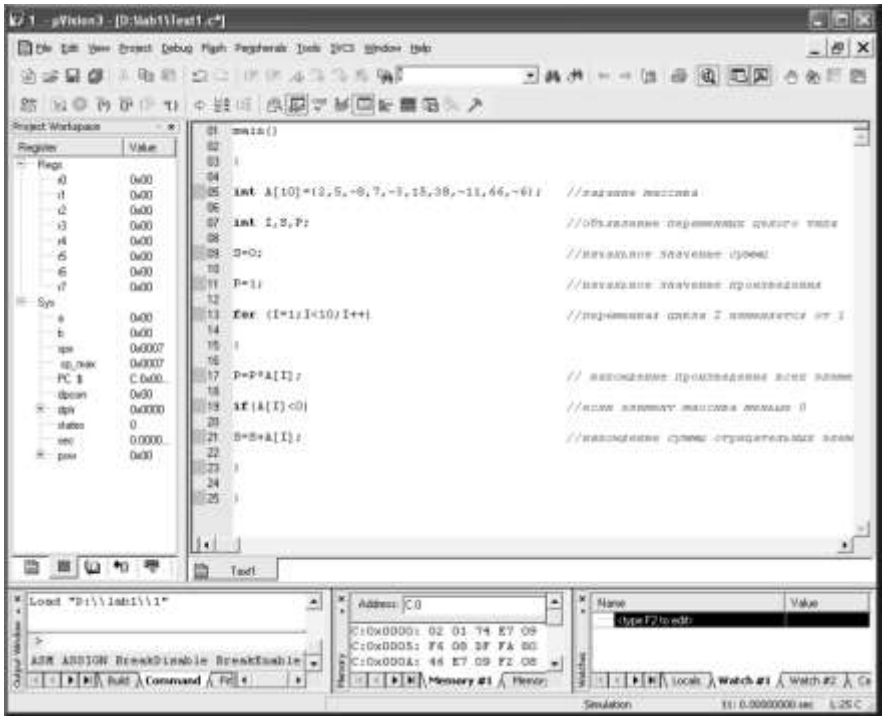


Рисунок 47 – Внешний вид интегрированной среды программирования в режиме отладки программ

Окно просмотра памяти контроллера можно настроить на просмотр памяти программ или памяти данных, введя в диалоговое окно ―адрес‖ ключ, двоеточие и адрес начальной ячейки памяти.

Например:

d:0 – просмотреть память данных начиная с нулевой ячейки;

c:0 – просмотреть память программ начиная с нулевой ячейки;

x:0 – просмотреть внешнюю память данных начиная с нулевой ячейки.

При использовании встроенного отладчика программ для контроля переменных можно воспользоваться окном ―Watch. В большинстве случаев это намного выгоднее, чем использовать просмотр памяти данных. Переменные в этом окне отображаются в том формате, в котором они были объявлены в программе. Для добавления переменной в окно ―Watch достаточно щѐлкнуть правой кнопкой мыши по переменной в тексте программы в окне отладчика программ, как это показано на рисунке 48.

Рисунок 48– Добавления переменной в окно просмотра ―Watch

Окно просмотра переменных содержит две закладки ―Watch #1 и ―Watch #2. Это позволяет группировать переменные, по какому либо признаку, например по отлаживаемым подпрограммам. При добавлении переменной Вы выбираете номер окна просмотра окна переменных.

Кроме просмотра глобальных переменных, которые существуют на протяжении всей программы, окно просмотра переменных содержит закладку ―locals‖. Эта закладка позволяет отслеживать локальные переменные, которые существуют только внутри подпрограммы. Вводить имена локальных переменных в эту закладку не нужно. Они появляются в этой закладке автоматически, как только Вы попадаете в подпрограмму, в которой используются локальные переменные.

При отладке программ на языке программирования ассемблер очень важно контролировать содержимое внутренних регистров микроконтроллера. Это позволяет

сделать закладка Regs в окне менеджера проектов, показанная на рисунке 22 (левая рабочая область). В этом окне можно проконтролировать содержимое регистров текущего банка, указателя стека и программного счѐтчика, содержимое аккумуляторов A и B, а также состояние рабочих флагов микроконтроллера в регистре PSW.

Один оператор программы может быть выполнен нажатием кнопки F11. Если вызов подпрограммы рассматривается как один оператор, то пошаговая отладка программы осуществляется нажатием кнопки F10.

Использование точек останова позволяет пропускать уже отлаженную часть программы. Для того, чтобы установить точку останова, можно воспользоваться кнопкой (Insert/remove Breakpoint) на панели файлов или воспользоваться главным или всплывающим меню. Перед тем как нажать на кнопку установки точки останова, необходимо установить курсор на строку исходного текста программы, где необходимо остановить выполнение программы.

Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаѐтся ли управление данному оператору.

После того, как установлены все необходимые точки останова осуществляется выполнение программы в свободнобегущем режиме. Для этого можно воспользоваться кнопкой (Run) или нажать на кнопку F5 на клавиатуре.

Может возникнуть ситуация, что программа не передаѐт управление ни одному из операторов, на которых установлены точки останова. В этом случае для прекращения выполнения программы следует воспользоваться кнопкой  (Halt) или нажать на кнопку ―Esc‖ на клавиатуре.

Точка останова может быть использована многократно. Иногда же возникает необходимость однократно пропустить часть операторов. В этом случае можно воспользоваться кнопкой выполнения программы до курсора  (Run to Cursor line). При нажатии на эту кнопку программа будет выполняться до тех пор, пока управление не будет передано оператору, на котором находится курсор. Как только это произойдѐт, выполнение программы будет остановлено, и можно будет проконтролировать переменные и продолжить выполнение программы в пошаговом или свободнобегущем режиме.

Порядок выполнения работы:

1. Войдите в интегрированную среду программирования 2. Создайте новый файл исходного текста программы. Имя файла может быть, например, L1.c (расширение *.c обязательно). Текст программы:

```
main()
{
//задание массива int A[10]={2,5,-8,7,-3,15,38,-11,66,-6};
//объявление переменных целого типа
int I,S,P;
//начальное значение суммы
S=0;
//начальное значение произведения
P=1;
//переменная цикла I изменяется от 1 до 10 с шагом 1
for (I=1;I<10;I++)
{
// нахождение произведения всех элементов массива
P=P*A[I];
//если элемент массива меньше 0
if(A[I]<0)
S=S+A[I];
//нахождение суммы отрицательных элементов массива
}}
```

Эта программа находит сумму отрицательных элементов массива A[10]. После выполнения программы результат (сумма) будет находиться в ячейке памяти S. 3. Создайте проект с именем LAB1.

4. Добавьте в проект файл с программой.

5. Оттранслируйте программный проект.

6. Убедитесь, что при трансляции программного модуля не обнаружены синтаксические ошибки.

7. Убедитесь, что в директории проекта созданы загрузочный файл с расширение .lst и загрузочный hex-файл с расширением .hex.

8. Выполните пошаговую отладку программы с использованием кнопки F11. На каждом шаге выполнения программы запишите значения используемых переменных программы: A[i] и S.

Ход работы:

1. Включите ЭВМ, и вызовите интегрированную среду программирования, щелкнув мышью по значку  .
2. Создайте новый файл исходного текста программы.
3. Введите заданный исходный текст программы, используя клавиатуру.
4. Откройте диалоговое окно сохранения файла, щелкнув мышью по значку  .
5. Создайте новую папку с именем LAB1. Для этого щѐлкните мышью по значку  .
6. Введите имя файла, например L1.c (расширение файла должно быть обязательно *.c), и нажмите на кнопку клавиатуры —Enter.
7. Создайте новый проект. Для этого выберите подменю —New project‖ из меню —project. Выберите папку размещения нового проекта – LAB1. Укажите имя проекта – L1.
8. Теперь можно подключать к программному проекту файл с исходным текстом программы. Для этого можно щѐлкнуть правой кнопкой мыши по значку группы файлов в окне менеджера проектов, как это показано на рисунке 40, и выбрать опцию добавления файлов к программному проекту.
9. Оттранслируйтепроект, нажавкнопку —F7 (Rebuild all target files). Если есть ошибки, исправьте их в текстовом редакторе среды Keil.
10. При отсутствии ошибок убедитесь, что в директории проекта появился загрузочный файл с расширением .hex.
11. Для вызова отладчика щелкните мышью значок его запуска  .
12. Выполните пошаговую отладку программы с использованием кнопки —F11. На каждом шаге выполнения программы запишите значения используемых переменных программы A[I] и S. Если при вызове отладчика в нижней части экрана отсутствует окно просмотра переменных, его можно включить, выбрав в меню —View команду —Watch & call stack window, либо воспользоваться иконками, показанными на рисунке 46. Для добавления переменной в окно —Watch достаточно щёлкнуть правой кнопкой мыши по нужной переменной в окне программы и в появившемся окне выбрать ADD to Watch window.

Форма представления результата:

Отчет по работе должен содержать:

1. наименование работы и цель работы;
2. результаты работы;
3. выводы по работе.

Лабораторная работа №2.

Организация ввода-вывода информации через параллельные порты МК ADuC842

Формируемые компетенции:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем.

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы:разработка программы управления светодиодами с использованием параллельного порта МК

Выполнив работу, Вы будете:

уметь:

- составлять программы на языке ассемблера для микропроцессорных систем;
- производить тестирование и отладку МПС;

Материальное обеспечение:

1. Компьютер с лицензионным программным обеспечением.
1. Мультимедиа проектор.
2. Лабораторные стенды «LESO1»

Подготовка к работе

Изучить теоретический материал по теме (см. Практическую работу №2)

Работа с регистрами специальных функций

Все программно доступные регистры управления, конфигурации и данных микроконтроллера включены в область регистров специальных функций (Special Function Register — SFR). Эта область формально занимает старшие 128 байт внутренней памяти данных, но обращение должно осуществляться по определенным адресам ячеек или отдельным битам. Регистры SFR реализуют интерфейс между микропроцессорным ядром и внутренней периферией микроконтроллера. Все регистры специальных функций имеют как символические имена (мнемоники), так и адреса в качестве ячеек внутренней памяти. Информацию об SFR можно найти в документации на микроконтроллер, обычно указывается не только название и адрес регистра, но и значение, записанное в них после сброса контроллера.

Часть регистров, адреса которых заканчиваются на 0h или 8h, содержит прямо адресуемые биты, адреса этих битов находятся в диапазоне 80h – FFh. Кроме адресов эти биты имеют еще и имена, которые могут быть определены в языках программирования. Таким образом, к прямо адресуемым битам регистрам специальных функций можно обращаться по именам. Если в регистре биты не прямо адресуемые, то их имена служат только для удобства описания функционирования соответствующего блока микроконтроллера. Для обращения к определенному биту такого регистра должна производиться выборка байта целиком, а затем накладываться нужная маска.

Для работы с портами микроконтроллера на языке C51 введены специальные типы переменных – регистры специальных функций sfr. Синтаксис определения регистра выглядит следующим образом:

```
sfr name = address;
```

где `name` — имя регистра SFR (любой идентификатор);

`address` — адрес регистра SFR.

Пример объявления:

```
sfr P0 = 0x80; /* Порт-0, адрес 80h */
sfr P1 = 0x90; /* Порт-1, адрес 90h */
sfr P2 = 0xA0; /* Порт-2, адрес A0h */
sfr P3 = 0xB0; /* Порт-3, адрес B0h */
```

После такого объявления в программе можно использовать регистр как обычную однобайтовую переменную.

Объявление отдельных битов в побитно адресуемых регистрах происходит с помощью ключевого слова sbit одним из следующих способов:

```
sbit name = sfr-name^bit-position;
sbit name = sfr-address^bit-position;
sbit name = sbit-address;
```

где:

`name` — имя бита в SFR;

`sfr-name` — имя заранее объявленного SFR;

`bit-position` — позиция бита в соответствующем регистре;

`sfr-address` — адрес регистра SFR;

`sbit-address` — непосредственный адрес бита.

Таким образом, второй бит порта P0 может быть записан одним из способов:

```
sbit p0_2 = 0x82; // непосредственный адрес
sbit p0_2 = 0x80^2; // используя адрес SFR
sbit p0_2 = P0^2; // заранее объявленный P0
```

Для определения адресов регистров и отдельных разрядов в них следует обратиться к рисунку 5.

Объявление регистров специальных функций следует делать перед основной подпрограммой. SFR не может быть объявлен внутри функции.

Работа загрузчика nwFlash

Для загрузки исполняемого кода во внутреннюю память микропроцессора и взаимодействия лабораторного стенда с ПК разработана программа nwFlash. Программа nwFlash позволяет:

— производить поиск подключенных к компьютеру по USB интерфейсу лабораторных стендов;
— активировать соединение с одним из найденных стендов;
— выполнять сброс микроконтроллера (Reset);
— загружать во flash – память микроконтроллера пользовательскую программу;
— принимать и отправлять данные в текстовом и шестнадцатеричном виде по интерфейсу UART (режим терминала).

Интерфейс nwFlash состоит из трех элементов: главного меню, окна терминала и окна состояния.

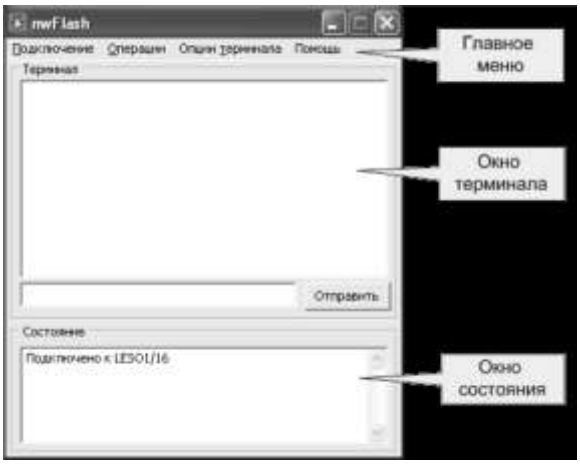


Рисунок 49 – Интерфейс загрузчика nwFlash

Главное меню позволяет производить операции со стендом, а также настраивать параметры терминала. Окно терминала служит для отображения данных, посылаемых микроконтроллером по интерфейсу UART, а также для отправки пользовательских данных от компьютера микроконтроллеру. В окне состояния отображаются результаты всех проведѐнных операций для контроля.

Для работы с программой nwFlash следует запустить программу. При нажатии на пункт главного меню "Подключение" программа выполнит поиск подключенных стендов и отобразит их названия в раскрывшемся меню. Если вы забыли подключить стенд, то появится надпись "нет подключенных стендов", в этом случае подключите стенд и снова раскройте меню "Подключение".

После выбора стенда из меню "Подключение". В окне состояния должна появиться надпись "Подключено к "имя_стенда". После этого становится доступным пункт меню "Операции", где можно:

— выполнить сброс микроконтроллера. На стенде начнѐт выполняться программа, записанная в микроконтроллер в последний раз;
— стереть flash-память микропроцессора;
— загрузить исполняемый *.hex файл в память микроконтроллера.

В появившемся окне необходимо указать путь к *.hex файлу.

Рисунок 50 – Интерфейс загрузчика nwFlash. Операция.

После выполнения работы со стендом, выберите пункт "Отключиться" в меню "Подключение", затем закройте программу.

Порядок выполнения работы

1. Вывод информации через параллельный порт

1. По принципиальной схеме установите, к каким портам микроконтроллера подключены светодиоды.
2. По таблице регистров специальных функций (SFR) определите адреса регистров требуемых портов.
3. Войдите в интегрированную среду программирования Keil-C.
4. Создайте файл проекта.
5. Введите текст программы в соответствии с заданием:
6. Каждому студенту требуется зажечь светодиоды, соответствующие номеру своего варианта в бинарном виде.
7. Оттранслируйте программу, и исправьте синтаксические ошибки.
8. Загрузите полученный *.hex файл в лабораторный стенд LESO1.
9. Убедитесь, что на лабораторном стенде LESO1 зажигаются требуемые светодиоды.
10. Покажите результат преподавателю.

2. Ввод информации через параллельный порт (дополнительно)

1. По принципиальной схеме установите, к какому порту микроконтроллера подключена кнопка S2.
2. Определите адрес порта, к которому подключена кнопка.
3. Измените исходный текст программы таким образом, чтобы требуемые светодиоды загорались только по нажатию кнопки.
4. Оттранслируйте программу, и исправьте синтаксические ошибки.
5. Загрузите полученный ***.hex** файл в лабораторный стенд LESO1.
6. Убедитесь, что при нажатии на кнопку зажигается светодиоды, соответствующие вашему варианту.
7. Покажите результат преподавателю.

3. Примеры программ приведены в файлах 1.c и 2.c (см. Приложение)

Внесите изменения (по вариантам) в программу 1.c для включения заданных светодиодов, используя параллельный порт МК. Включенные светодиоды помечены X, отключенные – 0.

№ варианта	№ светодиода			
	1	2	3	4
1	X	0	0	0
2	0	X	0	0
3	0	0	X	0
4	0	0	0	X
5	X	0	0	X
6	X	0	X	0

Внесите изменения в программу 2.с для включения заданных светодиодов по нажатию на кнопку, используя параллельный порт МК. Включенные светодиоды помечены X, отключенные – 0.

№ варианта	№ светодиода				№ светодиода			
	1	2	3	4	1	2	3	4
1	X	0	0	0	0	0	0	X
2	0	X	0	0	0	0	X	0
3	0	0	X	0	0	X	0	0
4	0	0	0	X	X	0	0	0
5	X	0	0	X	0	X	X	0
6	X	0	X	0	0	X	0	X

Форма представления результата:

Отчет должен содержать:

1. Цель работы.
2. Графическую схему алгоритма программы.
3. Исходный текст программы.
4. Содержимое файла листинга программного проекта.
5. Выводы по выполненной лабораторной работе.

Лабораторная работа №3.
Разработка программы управления клавиатурой матричного типа

Формируемые компетенции:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем.

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы:разработать и отладить программу управления матричной клавиатурой с помощью МК

Выполнив работу, Вы будете:

уметь:

- составлять программы на языке ассемблера для микропроцессорных систем;
- производить тестирование и отладку МПС;

Материальное обеспечение:

1. Компьютер с лицензионным программным обеспечением.
2. Мультимедиа проектор.
3. Лабораторные стенды «LESO1»

Подготовка к работе:

изучить необходимый теоретический материал по теме (см. Практическая работа №3)

Рекомендации к составлению программы

Программа для микроконтроллера жестко зависит от принципиальной схемы разрабатываемого устройства. Невозможно написать программу для микроконтроллерного устройства не имея перед глазами его схемы. Поэтому, перед началом работы по принципиальной схеме учебного стенда LESO1 следует изучить способ подключения клавиатуры и светодиодов к микроконтроллеру: определить, к каким портам подключены светодиоды, столбцы и строки клавиатуры. Затем по таблице SFR нужно узнать адреса регистров задействованных портов ввода-вывода.

Программа, управляющая микроконтроллером, запускается при включении питания устройства и не завершает свою работу, пока не будет выключено питание. Поэтому в программе обязательно должен быть организован бесконечный цикл. В теле цикла должен производиться опрос клавиатуры, анализ полученных данных и вывод результата на светодиод. Опрос клавиатуры заключается в последовательном сканировании каждого столбца, для этого на соответствующую линию порта вывода подается логический ноль (эквивалент общего провода), на остальных столбцах должен быть высокий уровень, после чего с порта ввода, к которому подключены строки, считывается код. Если считаны все единицы, то ни одна из клавиш не нажата, в противном случае код содержит информацию о нажатых клавишах. Стоит заметить, что считанный код содержит не только номер замкнутого контакта, но и информацию о нажатии нескольких кнопок одновремен-

но, поэтому лучше хранить в памяти котроллера непосредственно считанный код, а не готовый номер кнопки. Для хранения считанного кода следует ввести специальную переменную.

При написании программы нужно помнить об особенности параллельного порта P1 в микроконтроллере ADuC842. Этот порт по умолчанию настроен на ввод аналоговых сигналов (функция АЦП). Для того чтобы перевести порт в режим цифрового входа, в соответствующий бит порта необходимо записать логический ноль. Сделать это нужно один раз при инициализации микроконтроллера. Порт не имеет внутреннего усиливающего транзистора, и потому при вводе дискретной информации через него не требуется записывать в разряды логическую единицу.

Порядок выполнения работы:

1. По принципиальной схеме установите, к каким портам микроконтроллера подключены светодиоды, а также столбцы и строки клавиатуры.
2. По таблице регистров специальных функций (SFR) определите адреса регистров требуемых портов.
3. Войдите в интегрированную среду программирования Keil-C.
4. Создайте и настройте должным образом проект.
5. Введите текст программы в соответствии с заданием: При нажатии на кнопку, согласно варианту, загорается комбинация светодиодов, соответствующая в бинарном виде номеру кнопки; при отпускании кнопки, светодиоды должны погаснуть.
6. Оттранслируйте программу, и исправьте синтаксические ошибки.
7. Загрузите полученный ***.hex** файл в лабораторный стенд LESO1.
8. Убедитесь, что программа функционирует должным образом.

Пример программы приведен в файле 3.c (см. Приложение)

Внесите изменения в программу 3.c для включения в работу только определенных клавиш клавиатуры, остальные - отключены.

№ варианта	Включенные клавиши		
1	1	2	3
2	4	5	6
3	7	8	9
4	1	4	7
5	2	5	8
6	3	6	9

Форма представления результата:

Отчет должен содержать:

1. Цель работы.
2. Графическую схему алгоритма программы.
3. Исходный текст программы.
4. Содержимое файла листинга программного проекта.
5. Выводы по выполненной лабораторной работе

Лабораторная работа №4.
Разработка программы управления таймерами МК ADuC842

Формируемые компетенции:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем.

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы:разработать и отладить программу для формирования временных интервалов с помощью таймеров МК с индикацией на светодиод

Выполнив работу, Вы будете:

уметь:

- составлять программы на языке ассемблера для микропроцессорных систем;

- производить тестирование и отладку МПС;

Материальное обеспечение:

1. Компьютер с лицензионным программным обеспечением.
2. Мультимедиа проектор.
3. Лабораторные стенды «LESO1»

Подготовка к работе:

изучить необходимый теоретический материал по теме (см. Практическая работа №4)

Способы программной реализации работы таймера

Перед первым запуском таймера должна быть выполнена инициализация: выбран режим работы таймера, способ запуска, источник тактирования. Для настройки таймера в регистр специальной функции **TMOD** следует записать соответствующее число.

Допустим, тактовый генератор сконфигурирован для генерации с частотой 1МГц, а нам требуется сформировать временные интервалы 3500 мкс. В этом случае на вход таймера будут поступать импульсы с периодом 1мкс. Для 16-битного счетчика максимальное время переполнения составит 65536 мкс. Для формирования меньшего интервала времени в счетчики таймера следует загрузить начальное значение *Code.* Из формулы 1 выразим *Code:*

$Code$=2n−$TTTG$ =65536−35001=62036

Счетчик таймера состоит из двух отдельных 8-битных счетчиков **TH**x и **TL**x, поэтому полученное значение *Code* следует расщепить на два отдельных байта. Сделать это можно разными способами. Можно с помощью калькулятора перевести число *Code* из десятеричной системы счисления в шестнадцатеричную: 62036 10=F2 5416. Тогда старший байт **TH**x будет равен F216, а младший **TL**x равен 5416.

При расщеплении константы можно воспользоваться делением на 256, тогда:

THx = 62036/256; // заносим старший байт числа 62036

TLx = 62036; // заносим младший байт числа 62036

Операция деления числа на 256 эквивалентна сдвигу этого числа на 8 разрядов вправо, тогда приведенный участок программы можно записать так:

TH0 = 62036>>8; // заносим старший байт числа 62036

TLx = 62036; // заносим младший байт числа 62036

Существует иной способ загрузки начального значения в таймер. Число, которое необходимо загружать в таймер, можно найти как отношение требуемого интервала времени, 3500мкс, и периода импульсов на входе таймера, 1мкс. Так как таймер суммирующий, то в регистры таймера будем загружать отрицательное число с абсолютным значением, равным требуемому отношению. Для выделения старшего байта из 16-разрядного числа используем функцию сдвига этого числа на 8 разрядов вправо, тогда соответствующий код программы будет выглядеть следующим образом:

THx = (-3500)>>8; // заносим старший байт числа 62036

TLx = -3500; // заносим младший байт числа 62036

После того как регистром **TMOD** установлен режим работы и загружены начальные значения счетчиков, можно запускать таймер. Для этого в бит запуска **TRx** следует записать логическую единицу. Теперь в тело основного цикла нужно включить участок программы, который будет ожидать окончания работы таймера и только после этого приступать к выполнению следующего прохода по циклу. Это можно сделать с помощью команды, которая будет проверять флаг переполнения таймера **TFx**. Затем необходимо снова задать следующий интервал времени. Следует помнить, что перед запуском таймера, следует обнулить флаг переполнения.

Ниже приведен один из вариантов программы, реализующей задержку на 3,5 мс.

TH0=62036>>8; // заносим старший байт числа 62036

TL0=62036; // заносим младший байт числа 62036

TR0=1; // запускаем таймер while (!TF0); // ждем переполнения таймера

TF0=0; // обнуляем флаг переполнения

Для формирования временных интервалов большей длительности данный участок программы можно запустить в цикле необходимое количество раз.

Порядок выполнения работы

1. Разработайте алгоритм программы соответственно заданию: сформировать сигнал прямоугольной формы скважности 2 заданного периода, согласно варианту (таблица 4). Сигнал должен выводиться на светодиод.
2. По принципиальной схеме установите, к каким портам микроконтроллера подключен светодиод. 3. По таблице регистров специальных функций (SFR) определите адреса регистров управления таймерами.
4. Рассчитайте значение регистров **TLx** и **THx** для формирования заданного времени работы таймера (таблица 4).
5. Рассчитайте требуемое количество итераций цикла для формирования сигнала с периодом **T** (таблица 4)
6. Войдите в интегрированную среду программирования Keil-C.
7. Создайте и настройте должным образом проект.
8. Разработайте и введите текст программы в соответствии с созданным алгоритмом.
9. Оттранслируйте программу, и исправьте синтаксические ошибки.
10. Загрузите полученный *.hex файл в лабораторный стенд LESO1.
11. Убедитесь, что программа функционирует должным образом. С помощью секундомера измерьте период сигнала.

Пример программы приведен в файле 4.c (см. Приложение)

Внесите изменения в программу 4.c, задав время таймера согласно варианта:

Таблица 4– Варианты заданий

номер варианта	таймер	время таймера	период сигнала T
1	таймер 0	5 мс	2 с
2	таймер 1	15 мс	3 с
3	таймер 0	10 мс	4 с
4	таймер 1	50 мс	5 с
5	таймер 0	30 мс	6 с
6	таймер 1	14 мс	7 с
7	таймер 0	20 мс	8 с
8	таймер 1	25 мс	9 с
9	таймер 0	40 мс	10 с
10	таймер 1	4 мс	11 с
11	таймер 0	60 мс	12 с
12	таймер 1	65 мс	13 с
13	таймер 0	35 мс	14 с
14	таймер 1	7,5 мс	15 с
15	таймер 0	40 мс	16 с

Форма представления результата:

Отчет должен содержать:
1. Цель работы.
2. Структурную схему таймера микроконтроллера ADuC842 в выбранном режиме.
3. Расчет начальных значений счетных регистров таймера.
4. Обоснование выбора значения регистра **TMOD.**
5. Графическую схему алгоритма работы программы.
6. Исходный текст программы.
7. Содержимое файла листинга программного проекта.
8. Выводы по выполненной лабораторной работе.

Лабораторная работа №5.
Организация ввода-вывода информации через последовательный порт МК ADuC842

Формируемые компетенции:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем.
ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы:разработка и отладка программы обмена информацией через последовательный порт МК

Выполнив работу, Вы будете:

уметь:

- составлять программы на языке ассемблера для микропроцессорных систем;
- производить тестирование и отладку МПС;

Материальное обеспечение:

1. Компьютер с лицензионным программным обеспечением.
2. Мультимедиа проектор.
3. Лабораторные стенды «LESO1»

Подготовка к работе:

изучить необходимый теоретический материал по теме (см. Практическая работа №5)

Способы программной реализации работы UART

Перед первым обращением к приемо-передатчику UART последовательный порт должен быть настроен: определен режим работы, выбран и настроен источник синхронизации. Режим работы UART устанавливается битами **SM0** и **SM1** регистра **SCON**. Так как регистр имеет как байтовую, так и битовую адресацию, выполнить настройку можно разными способами: записать в регистр **SCON** требуемое число или установить каждый бит отдельно. Источник синхронизации определяется битом **T3EN** регистра **T3CON**: если в этот бит записать логическую единицу, то синхронизация будет происходить от Таймера 3, если ничего не записывать (по умолчанию **T3EN = 0**), то синхронизация от Таймера 1.

При использовании Таймера 1 необходимо сконфигурировать его для работы в режиме 2 (свободнобегущий таймер с автоперезагрузкой), для этого в старшие четыре бита регистра **SMOD** следует записать двоичную комбинацию 0010b. Регистр счетчика **TH1** определяет скорость передачи информации по UART, его значение следует рассчитать по формуле 3. После записи **TH1** таймер нужно запустить, делается это записью в бит **TR1** регистра **TMOD** логической единицы.

При синхронизации от Таймера 3, по формулам 5 и 6 рассчитываются делители DIV и T3FD. Если запись делителя T3FD делается непосредственно в регистр **T3FD**, то делитель DIV определяется младшими тремя битами регистра **T3CON**, при этом в старший бит этого регистра (**T3EN**) должна быть записана логическая единица. Запуск таймера происходит автоматически.

Отправление данных по UART начинается любой командой, результат выполнения которой записывается в регистр **SBUF**:

```
SBUF = 0x45;          // отправить символ “E”
```

Можно каждый раз не пользоваться таблицей ASCII для определения кода символа, в языке «Си» для этого есть удобный инструмент: достаточно взять требуемый символ в апострофы, компилятором это будет интерпретировано как код символа.

```
SBUF = ‘E’;           // отправить символ “E”
```

Если же требуется отправить не один символ, то прежде чем следующий код будет записан в **SBUF**, следует подождать, пока предыдущий символ будет отправлен. О конце передачи сигнализирует флаг **TI** регистра **TCON**, когда передача завершена, в бит **TI** аппаратно записывается логическая единица. Можно программно организовать в цикле проверку **TI** на равенство нулю, а следующий байт отправлять только тогда, когда **TI** окажется равен единице:

```
SBUF = 0x45;          // отправить символ “E”
while (!TI);          // пока TI равен нулю, выполнять пустой цикл
TI = 0;               // сбросить флаг для следующей передачи
```

Аналогично выполняется и прием байта. Принятый байт может быть считан из буферного регистра лишь тогда, когда был принят последний бит и флаг приема **RI** установлен.

```
while(!RI);           // ждем завершение приема байта
cmd = SBUF;           // считываем принятый байт в переменную cmd
RI = 0;               // сброс флага приема
```

В приложениях, где время выполнение критично недопустимо тратить много времени при передаче на ожидание пока байт будет отправлен и буфер освободится, в этом случае можно не дожидаясь полной отправки байта приступить к выполнению дальнейшей программы, но перед отправкой следующего байта нужно убедиться, что буфер освободился и передатчикготов к работе. Участок программы, отправляющий байт данных можно переделать следующим образом:

```
while(!TI);          /*подождать, пока буфер передачи не освободится (если занят)*/
SBUF = 0x45;         // заполнить буфер и начать передачу
TI = 0;              // сбросить флаг передачи в нуль
```

С таким вариантом реализации устраняются паузы на выполнение программы между передачами отдельных байтов.

Ниже на рисунке 51 приведены адреса регистров специальных функций, используемых в работе.

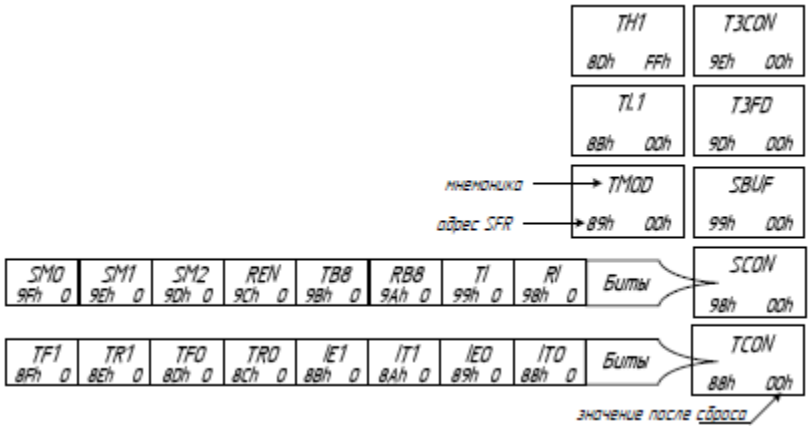


Рисунок51 – Адреса регистров специальных функций

При написании программы следует помнить, что программа для микроконтроллера должна выполняться до отключения питания устройства и не может быть завершена. Поэтому программа должна содержать бесконечный цикл.

Взаимодействие микроконтроллера с персональным компьютером

Учебный лабораторный стенд LESO1 подключается к персональному компьютеру через микросхему преобразователя интерфейсов USB-UART. Для связи с микроконтроллером в программе загрузчика nwFlash реализован терминал. Терминал позволяет посылать через последовательный порт в микроконтроллер информацию, принимать и отображать принятую из микроконтроллера информацию. Настройка терминала осуществляется в пункте главного меню «Опции терминала».

Опции терминала позволяют:

1. выбрать режим отображения данных: текстовый или шестнадцатеричный, при этом изменяется также тип посылаемых данных;
2. выбрать кодировку **ANSI** (Windows-1251) или **ASCII** (DOS-866);
3. включать и выключать режим автоматической прокрутки текста;
4. очистить окно терминала;
5. сохранять принятую от микроконтроллера информацию в файл:
   - в том виде, как она пришла — пункт меню «Сохранить»;
   - в том виде, как она отображается в терминале — пункт меню «Сохранить как текст».

На рисунке 52 показана вкладка главного меню «Опции терминала». При обмене данными с учебным стендом необходимо установить требуемую скорость подключения. Сделать это можно в меню «подключение», как это показано на рисунке 53.

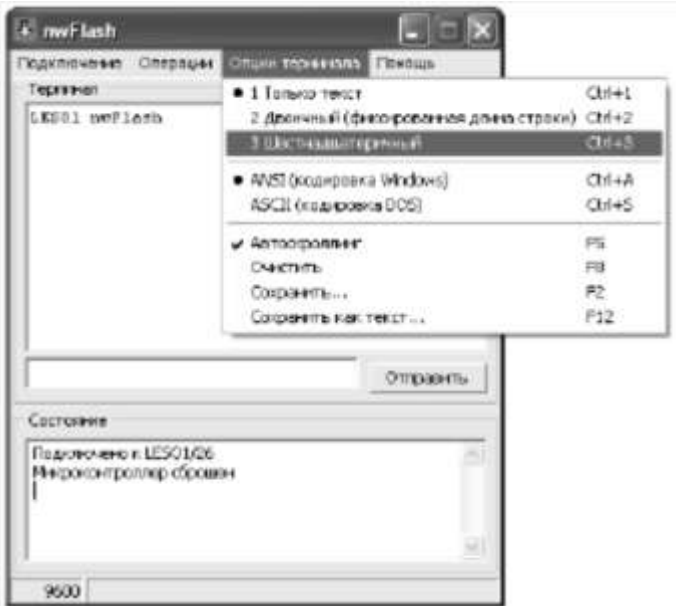


Рисунок 52 – Настройка опций терминала

Рисунок 53 – Настройка скорости UART

Порядок выполнения работы:

1 Вывод информации через последовательный порт

1. Разработайте алгоритм программы, передающей через последовательный порт UART данные в персональный компьютер — фамилию студента. Скорость передачи данных подобрать опытным путем. Источник синхронизации UART (Таймер 1 или Таймер 3) согласовывается с преподавателем. Взаимодействие микроконтроллера с компьютером осуществляется через терминал программы загрузчика — nwFlash.
2. По таблице регистров специальных функций (SFR) определите адреса регистров управления и настройки последовательного порта.
3. Определите значение регистров настройки последовательного порта и таймера, используемого для синхронизации.
4. Рассчитайте значения регистров таймера, используемого для синхронизации.
5. Войдите в интегрированную среду программирования Keil-C.
6. Создайте и настройте должным образом проект.
7. Разработайте и введите текст программы в соответствии с созданным алгоритмом.
8. Оттранслируйте программу, и исправьте синтаксические ошибки.
9. Настройте скорость UART терминала программы nwFlash соответственно заданию.
10. Загрузите полученный *.hex файл в лабораторный стенд LESO1.
11. Убедитесь, что в окне терминала вывелось имя студента.

Пример программы приведен в файле 5.c (см. Приложение)

Внесите изменения в программу 5.c для отображения**на мониторе порта**Вашего имени.

Форма представления результата:

Отчет должен содержать:

1. Цель работы.
2. Диаграмму передачи данных по последовательному порту.
3. Расчет параметров синхронизации (настройки таймеров).
4. Графическую схему алгоритма работы программы.
5. Исходный текст программы.
6. Содержимое файла листинга программного проекта.
7. Выводы по выполненной лабораторной работе.

Лабораторная работа №6.
Разработка программы управления символьным ЖКИ

Формируемые компетенции:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем.

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы: разработка и отладка программы вывода информации на ЖКИ с помощью МК

Выполнив работу, Вы будете:

уметь:

- составлять программы на языке ассемблера для микропроцессорных систем;
- производить тестирование и отладку МПС;

Материальное обеспечение:

1. Компьютер с лицензионным программным обеспечением.
2. Мультимедиа проектор.
3. Лабораторные стенды «LESO1»

Задание: Задание: изучить необходимый теоретический материал по теме (см. Практическая работа №6)

Перед началом работы требуется произвести инициализацию ЖКИ согласно алгоритму, показанному на рисунке 54.

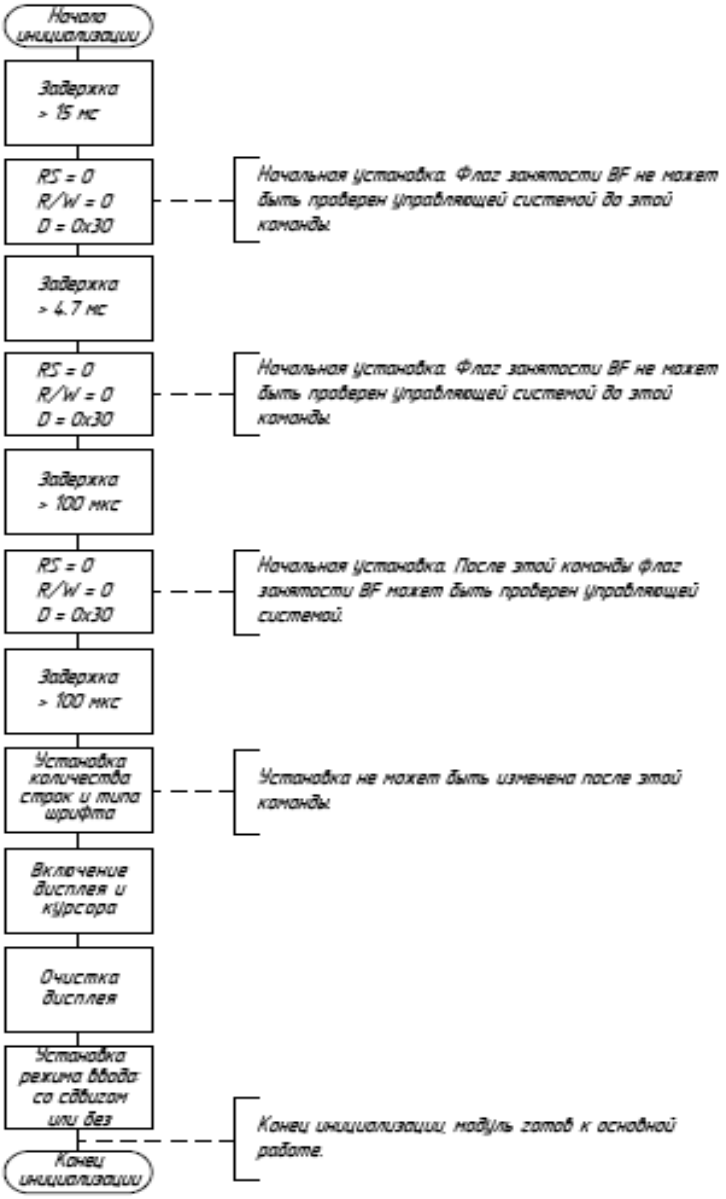


Рисунок 54 – Алгоритм инициализации ЖКИ

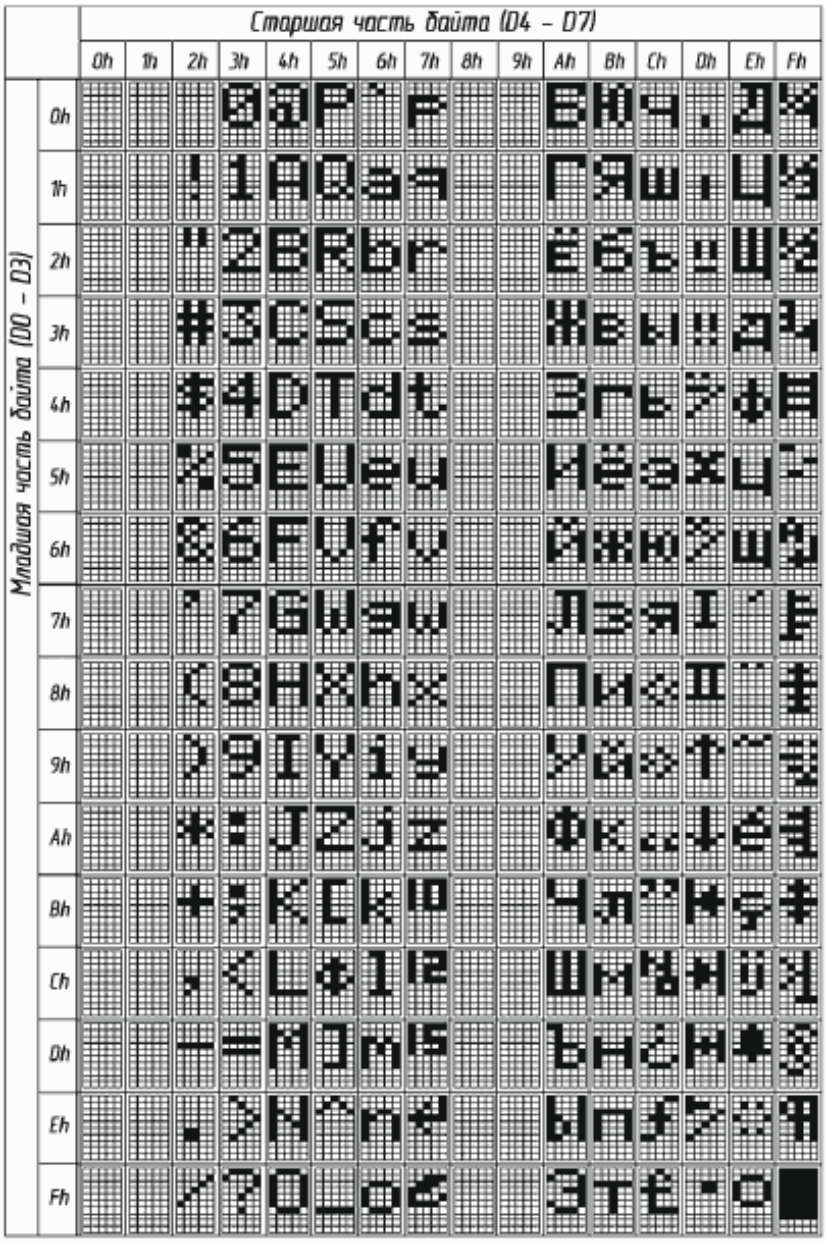


Рисунок 55 – Таблица символов знакогенератора

2 Рекомендации по программному управлению ЖКИ

Программу для работы с ЖКИ следует организовать в виде функций, выполняющих определенные действия, причем более сложные функции могут включать в себя простейшие. Простейшими могут быть такие подпрограммы, как функция, отправляющая команду контроллеру дисплея; функция, устанавливающая счетчик адреса; или функция, записывающая данные в DDRAM. В любом случае, общий алгоритм передачи информации контроллеру не изменится. Руководствуясь диаграммой передачи информации (рисунок 24), определим последовательность действий при передаче информации в ЖКИ следующим образом: устанавливаем требуемое значение **RS,** на линию **R/W** подаем логический ноль, затем на линию **E** выводим логическую единицу, после чего подаем на шину **D** значение передаваемого байта. Контроллер ЖКИ считает этот байт и состояние управляющих линий (**RS, R/W**) только после подачи на линию **E** логического ноля. При этом, если временные задержки, указные на диаграмме, меньше длительности машинного цикла, то ими можно пренебречь.

Код программы, реализующей запись в память ЖКИ байта данных, показан ниже:

```
RS=1;                    // выбираем команды или данные
RW=0;                    // выбираем направление передачи
E=1;
```

```
Data=symbol;              // выводим байт данных на шину D
E=0;                      // переводим сигнал на линии E из 1 в 0
delay ();                 /* ждем, пока контроллер выполняет внутренние операции*/
```

В приведенном участке программы подразумевается, что переменные **RS**, **RW** и **E** объявлены как **sbit**, а переменная **Data** – как **sfr**. Аналогично будет происходить передача любой команды контроллеру ЖКИ.

При реализации чтения информации из контроллера необходимо пользоваться диаграммой, приведенной на рисунке 25. Следует помнить, что для того, чтобы ввести информацию с параллельного порта, в него предварительно должны быть записаны логические единицы.

Для того, чтобы не загромождать основную программу алгоритм инициализации (рисунок 54) можно реализовать в виде отдельной подпрограммы. Временные задержки, указанные в алгоритме, следует задавать с помощью таймеров, как это делалось в лабораторной работе «Изучение таймеров микроконтроллера».

Порядок выполнения работы

1 Вывод символа на ЖКИ

1. Разработайте алгоритм программы, выводящей на экран ЖКИ ваше имя в заданной строке. Режим работы ЖКИ и номер строки определяется согласно варианту задания (таблица 10).
2. По принципиальной схеме учебного стенда LESO1 определите, к каким выводам микроконтроллера ADuC842 подключен ЖКИ. По таблице SFR определите адреса используемых портов ввода-вывода.
3. Разработайте и введите текст программы в соответствии с созданным алгоритмом.
4. Оттранслируйте программу, и исправьте синтаксические ошибки.
5. Загрузите полученный *.hex файл в лабораторный стенд LESO1.
6. Убедитесь, что на экране дисплея в заданной позиции появился требуемый символ.

2 Управление ЖКИ через последовательный порт персонального компьютера (дополнительно)

1. Измените программу таким образом, что бы на экране ЖКИ выводилась информация, переданная с персонального компьютера через UART. Передача команды осуществляется через терминал nwFlash. Выбор источника синхронизации и скорости передачи данных осуществляется по усмотрению студента.
2. Загрузите полученный ***.hex** файл в лабораторный стенд LESO1.
3. Через терминал nwFlash передайте коды символов, убедитесь, что соответствующие символы выводятся на экране индикатора.

Пример программы приведен в файле 6.c (см. Приложение)

Внесите изменения в программу 6.c для отображения**на ЖКИ** Вашего имени.

Форма представления результата:

Отчет должен содержать:

1. Цель работы.
2. Принципиальную схему подключения ЖКИ к управляющему микроконтроллер.
3. Структурную схему ЖКИ.
4. Диаграммы передачи данных по параллельному интерфейсу.
5. Расчет параметров таймера.
6. Графическую схему алгоритма работы программы.
7. Исходный текст программы.
8. Содержимое файла листинга программного проекта.
9. Выводы по выполненной лабораторной работе.

ПРИЛОЖЕНИЕ А
Листинг программ

1.c

```
sbitP1 = 0x80;
sbit P2 = 0x81;
sbit P3 = 0x82;
sbit P4 = 0x83;

main()
{
P1 = 1;
P2 = 1;
P3 = 0;
P4 = 1;
}
```

2.c

```
sbit p0_0=0x80;
sbit p0_1=0x81;
sbit p0_2=0x82;
sbit p0_3=0x83;
sbit p3_2=0xB2;


main()
{
int I;
I=0;
if(p3_2==I)
{
p0_0=1;
p0_1=1;
p0_2=0;
p0_3=0;
}
else {p0_0=0;p0_1=0;p0_2=0;p0_3=0;}
}
```

3.c

```
sfr p0=0x80;
sfr p1=0x90;
sbit p0_1=0x81;
sbit p0_2=0x82;
sbit p0_3=0x83;
sbit p0_4=0x84;
sbit p0_5=0x85;
sbit p0_6=0x86;
sbit p1_0=0x90;
sbit p0_0=0x80;

sbit p1_3=0x93;
sbit p1_2=0x92;
```

```
sbit p1_1=0x91;

 main()
 {
while(1)
{
p0=0xf0;
p1=0x00;
p0_4=0;
if (p1_0==0)
{p0=0xf1;}
else
 if(p1_1==0)
 {p0=0xf4;}
 else
  if(p1_2==0)
  {p0=0xf7;}
   else
  if(p1_3==0)
  {p0=0xff;}
   else{
p0=0xf0;
p1=0x00;
p0_5=0;}
if (p1_0==0)
{p0=0xf2;}
else
 if(p1_1==0)
 {p0=0xf5;}
 else
  if(p1_2==0)
  {p0=0xf8;}
 else{
p0=0xf0;
p1=0x00;
p0_6=0;}
if (p1_0==0)
{p0=0xf3;}
else
 if(p1_1==0)
 {p0=0xf6;}
 else
  if(p1_2==0)
  {p0=0xf9;}
   else
  if(p1_3==0)
{p0=0xff;}

 } }
```

4.c

```
sbit p0_0=0x80;
sbit p0_1=0x81;
```

```
sbit p0_2=0x82;
sbit p0_3=0x83;
sfr TH0=0x8C;
sfr TL0=0x8A;
sfr TMOD=0x89;
sfr TCON=0x88;
 main ()
 {
 int T=6200;
 int C=5000;
 TMOD=0x32;
 p0_1=0;
 p0_2=0;
 while((C--)>0)
 {TH0=T>>8;
 TL0=T;}
 TCON=0x10;
 while((TCON&&0x20)!=0x20)
{p0_0=!p0_0;
 p0_3=!p0_3;
 }


 }
```

5.c

```
sfr SCON =0x98;
sfrT3FD = 0x9D;
sfr T3CON = 0x9E;              // регистр конфигурации Таймером
sfr SBUF = 0x99;               // последовательный буфер
sbit TI = 0x99;
sbit REN = 0x9C;

main()
{

   SCON =0x40;
T3FD =0x2D;
   T3CON = 0x85;
   REN=1;                      //бит разрешения

   while(1){                   //цикл для бесконечного вывода сообщения

   SBUF = 'V';            //вывод буквы V
   while(!TI);
TI=0;

  SBUF = 'a';             //выводбуквы a
   while(!TI);
   TI=0;
  SBUF = 's';             //выводбуквы s
   while(!TI);
   TI=0;
```

```
  SBUF = 'i';               //выводбуквы i
   while(!TI);
TI=0;

  SBUF = 'l';               //выводбуквыl
while(!TI);
   TI=0;

  SBUF = 'y';               //выводбуквы y
   while(!TI); //
   TI=0;

   }
}
```

6.c

```
sfr D=0xA0;
sbit RS=0xB5;
sbit RW=0xB6;
sbit E=0xB7;
int i=0;
void delay();
void sleep();
void main()
{
delay();
RS=0; RW=0; E=1; D=0x30; E=0;
delay();
RS=0; RW=0; E=1; D=0x30; E=0;
delay();
RS=0; RW=0; E=1; D=0x30; E=0;
delay();

RS=0; RW=0; E=1; D=0x38; E=0;
delay();
RS=0; RW=0; E=1; D=0x06; E=0;
delay();
RS=0; RW=0; E=1; D=0x0C; E=0;
delay();
RS=0; RW=0; E=1; D=0x01; E=0;
delay();
RS=0; RW=0; E=1; D=0x02; E=0;
delay();
RS=0; RW=0; E=1; D=0xC0; E=0;
delay();


RS=1; RW=0; E=1; D=0x48; E=0;    delay();
RS=1; RW=0; E=1; D=0x69; E=0;      delay();
RS=1; RW=0; E=1; D=0x74; E=0;      delay();
RS=1; RW=0; E=1; D=0x6C; E=0;     delay();
RS=1; RW=0; E=1; D=0x65; E=0;      delay();
```

```
RS=1; RW=0; E=1; D=0x72; E=0;      delay();

RS=0; RW=0; E=10; D=0xC0; E=0;
delay();


RS=1; RW=0; E=1; D=0x4B; E=0;      delay();
RS=1; RW=0; E=1; D=0x61; E=0;      delay();
RS=1; RW=0; E=1; D=0x70; E=0;      delay();
RS=1; RW=0; E=1; D=0x75; E=0;      delay();
RS=1; RW=0; E=1; D=0x74; E=0;      delay();
RS=1; RW=0; E=1; D=0x21; E=0;      delay();
RS=1; RW=0; E=1; D=0x21; E=0;      delay();
RS=1; RW=0; E=1; D=0x21; E=0;      delay();
sleep();
}
void delay(){
int i=0;
while (i<30000){
i++;
}
i=0;
while (i<30000){
i++;
}
i=0;
while (i<30000){
i++;
}
}
void sleep(){
while (1) {
delay();
}
}
```