МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение
высшего профессионального образования
«Оренбургский государственный университет»

Е.В. БУРЬКОВА
А.С. БОРОВСКИЙ

МИКРОПРОЦЕССОРНЫЙ КОМПЛЕКС SDK-1.1. АРХИТЕКТУРА И ПРОГРАММИРОВАНИЕ

Рекомендовано Ученым советом государственного образовательного учреждения высшего профессионального образования «Оренбургский государственный университет» в качестве учебного пособия для студентов, обучающихся по программам высшего профессионального образования по направлению 230100 «Информатика и вычислительная техника»

Оренбург 2009

УДК 004.41(075.8)
ББК 32.973-018я73
Б 91

Рецензент
кандидат физико-математических наук, доцент Е.А. Корнев

Бурькова Е.В.
Б 91 **Микропроцессорный комплекс SDK-1.1. Архитектура и программирование: учебное пособие / Е.В. Бурькова, А.С. Боровский – Оренбург, ГОУ ОГУ, 2009. – 107 с.**

В учебном пособии рассмотрены архитектурные особенности учебного лабораторного микропроцессорного комплекса SDK-1.1, этапы программирования и возможности его применения для изучения курсов «Организация ЭВМ и систем» и «Микропроцессорные системы».

Учебное пособие предназначено для студентов специальностей направления 230100 «Информатика и вычислительная техника» и содержит задания и примеры программ для выполнения лабораторного практикума по выше названным курсам.

УДК 004.41(075.8)
ББК 32.973-018я73

Содержание

Введение

Потребность в проектировании контроллеров на основе микропроцессоров и программируемой логики продолжает стремительно увеличиваться. Сегодня происходит автоматизация практически всей окружающей нас среды с помощью дешевых и мощных микроконтроллеров. Микроконтроллер - это самостоятельная компьютерная система, которая содержит процессор, вспомогательные схемы и устройства ввода-вывода данных, размещенные в общем корпусе. Микроконтроллеры, используемые в различных устройствах, выполняют функции интерпретации данных, поступающих с клавиатуры пользователя или от датчиков, определяющих параметры окружающей среды, обеспечивают связь между различными устройствами системы и передают данные другим приборам. Эффективность проектирования контроллеров определяется в первую очередь квалификацией разработчика и арсеналом инструментальных средств.

Учебные микропроцессорные комплексы (стенды) на базе микроконтроллеров предназначены для изучения принципов организации и работы микропроцессорной элементной базы, вспомогательных элементов (память, контроллеры ввода-вывода и др.), получения навыков проектирования и программирования микропроцессорных систем различного назначения. Внимания заслуживает опыт ООО «ЛМТ» (Санкт-Петербург), которое разработало и последовательно развивает семейство микропроцессорных стендов инструментального и учебного назначения - SDK.

Основу лабораторного комплекса составляет контроллер-конструктор (микропроцессорный стенд) SDK-1.1 на базе ОКЭВМ фирмы Analog Devices ADuC812. Сам лабораторный комплекс представляет собой совокупность контроллера-конструктора, подключенного к персональному компьютеру, и программного обеспечения для ПК и SDK-1.1. Подключение осуществляется к COM-порту ПК через кабель RS232, комплекс инструментальных программ обеспечивает весь процесс программирования SDK-1.1: компиляцию, доставку и запуск программ в SDK-1.1.

Контроллер-конструктор имеет в своем составе устройства для ввода и отображения информации, снабжен блоком питания и может работать автономно от ПК.

Основными областями использования комплекса являются:

- обучение основам вычислительной и микропроцессорной техники, систем управления;
- автоматизация простых технологических процессов и лабораторных исследований;

– макетирование микропроцессорных систем, отладка программного обеспечения для систем на базе широко распространенного ядра Intel MCS-51;
– радиолюбительство, управление бытовой техникой.

Контроллер SDK-1.1 оснащен устройствами для обработки и формирования аналоговых и дискретных сигналов, а также приспособлениями для замыкания выходных цепей на входные и симуляции внешних событий. Это дает возможность использовать SDK-1.1 для обучения основам цифровой обработки сигналов, в качестве контрольно-измерительной панели при проведении экспериментов в различных областях, при настройке оборудования и т.п., а также для формирования аналоговых и дискретных сигналов с заданными параметрами в процессе управления различными объектами.

Главной областью применения микропроцессорного стенда SDK-1.1, безусловно, является обучение различным аспектам встраиваемой вычислительной техники. Студенты имеют возможность ознакомиться на практике с проектированием, программированием, отладкой и использованием создаваемой ими из "конструктора" SDK-1.1. Разнообразные устройства, входящие в состав стенда, позволяют изучить круг вопросов, связанных с организацией взаимодействия с ними через типичные интерфейсы, применяемые во встраиваемых вычислительных системах.

В первом разделе учебного пособия рассматриваются особенности архитектуры учебного стенда, приводится полное описание функциональных блоков, входящих в его состав. Приведены основные режимы работы стенда.

Во втором разделе приведены способы программирования стенда, инструментальные средства отладки и загрузки программ.

В разделах с третьего по пятый рассмотрены вопросы ввода-вывода, организации прерываний и многозадачности.

Шестой и седьмой разделы пособия содержит задания для лабораторного практикума по курсам «Организация ЭВМ и систем» и «Микропроцессорные системы», а также примеры текстов программ по предложенным заданиям.

Учебное пособие предназначено для студентов специальностей направления 230100 «Информатика и вычислительная техника» и может быть полезно преподавателям, использующим стенд SDK-1.1 в учебном процессе.

1 Архитектура лабораторного комплекса SDK-1.1

Учебный лабораторный комплекс SDK–1.1 предназначен для освоения студентами архитектуры и методов проектирования вычислительных систем на базе микроконтроллера ADuC812. Процессор ADuC812 является клоном Intel 8051 со встроенной периферией, имеет ядро MCS–51. Разнообразные устройства, входящие в состав стенда, позволяют изучить круг вопросов, связанных с организацией взаимодействия с ними через типичные интерфейсы, применяемые во встраиваемых вычислительных системах.

Стенд SDK–1.1 отличается следующими особенностями:

1) Вычислительное ядро:

а) центральный процессор имеет распространенную архитектуру MCS–51. Объем памяти контроллера позволяет реализовывать программные комплексы средней сложности;

б) развитая структура подсистемы временной синхронизации (встроенные часы реального времени, таймеры–счетчики) обеспечивает возможность глубокого исследования принципов и проблем организации систем реального времени, планирования, синхронизации процессов и т. п.;

в) центральный микроконтроллер ADuC812 позиционируется как микроконтроллер для построения простейших систем обработки сигналов. Он обладает широким набором каналов дискретного и аналогового ввода–вывода.

2) Система ввода–вывода:

а) дискретные входы и выходы позволяют подключать к SDK–1.1 внешние устройства для изучения их архитектуры и цифровых интерфейсов;

б) параллельная шина предназначена для подключения платы расширения SDX, увеличивающей количество портов дискретного и аналогового ввода–вывода комплекса SDK–1.1.

3) Разработка и отладка программного обеспечения:

а) SDK–1.1 поддерживает загрузку и запуск тестовых программ в ОЗУ контроллера без перепрограммирования энергонезависимой памяти программ;

б) обновление встроенного системного программного обеспечения в энергонезависимой памяти программ производится с персонального компьютера по стандартному последовательному каналу (RS–232C) без применения специальных программаторов.

1.1 Структура аппаратной части

Структура аппаратной части стенда SDK-1.1 представлена на рисунке 1.1.

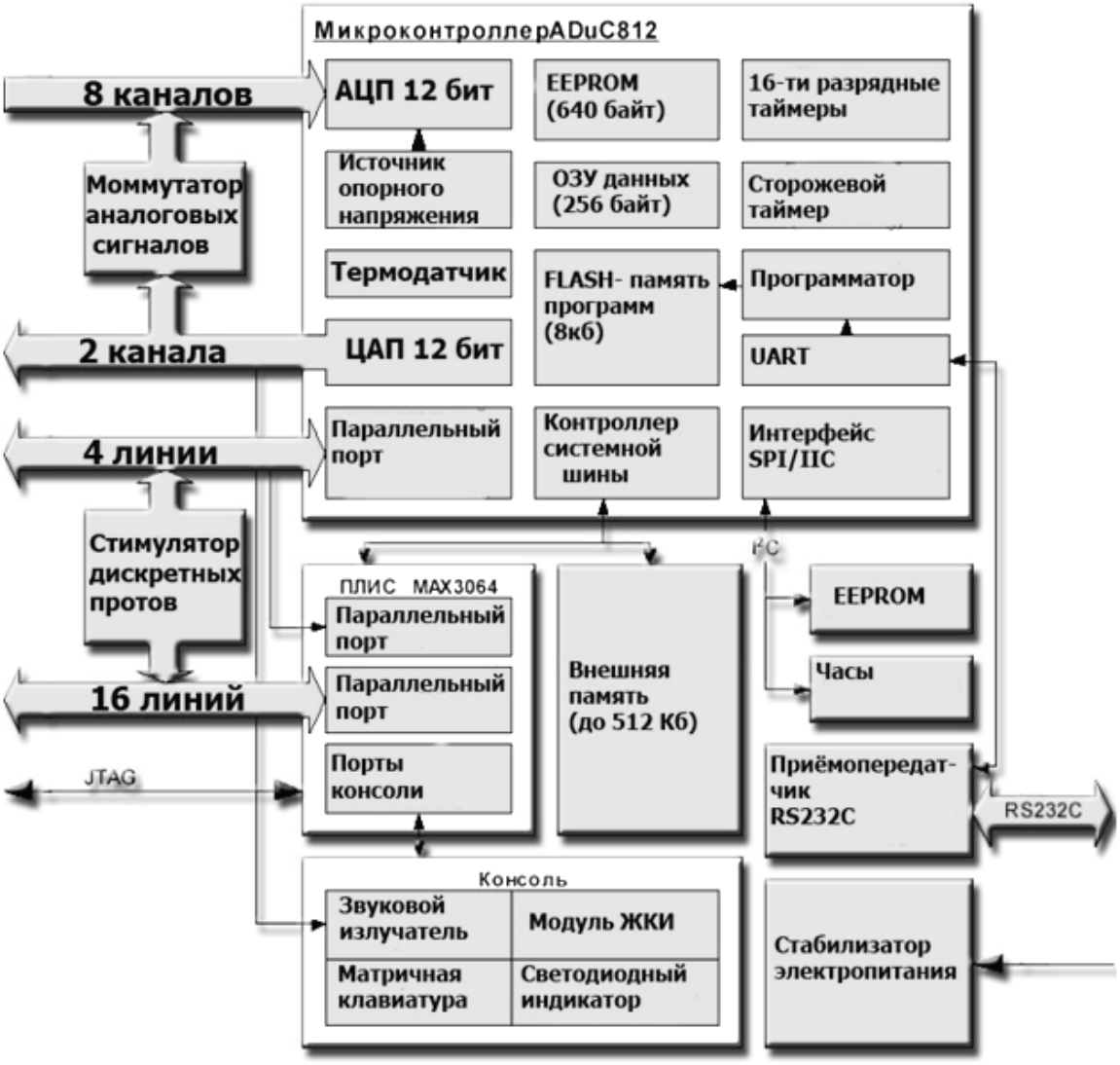


Рисунок 1.1 – Структура аппаратной части стенда SDK-1.1

В состав учебного стенда SDK-1.1 входят:

– вычислительное ядро на основе ОКЭВМ ADuC812;
– внешняя память E2PROM объемом 256 байт;
– 128К внешней памяти SRAM с возможностью расширения до 512К;
– FLASH-память (8 Кб);
– гальванически изолированный порт RS232C для связи с ПК;
– 8, 16, 20-ти разрядный порт дискретного ввода-вывода;
– аналоговый порт ввода на базе 8-канального 12-разрядного высокоскоростного АЦП со встроенным термодатчиком и возможностью работы в режиме прямого доступа к памяти (ПДП);

– аналоговый порт вывода на основе двух 12-рарядных ЦАП;
– второй блок EEPROM-памяти емкостью до 32 Кб, подключенный к вычислителю через интерфейс I2C;
– три 16-разрядных таймера-счетчика с внешними счетными входами (возможностью подачи сигналов через переключатели стенда) и блоком захвата/сравнения для измерения параметров и/или формирования дискретных сигналов;
– сторожевой таймер (Watchdog);
– жидкокристаллический индикатор для вывода текста;
– линейка из 8 сигнальных светодиодов;
– акустический пьезокерамический излучатель;
– матричная клавиатура на 16 клавиш;
– переключатели-стимуляторы 10 линий параллельного порта, сигналов от внешних источников прерываний, коммутаторы сигналов с выходов ЦАП на входы АЦП;
– часы/календарь с возможностью подключения внешней батареи питания.

Несмотря на большое количество устройств, входящих в стенд SDK-1.1, он имеет небольшие габариты: 13 x 12,5 x 2 (см), выполнен в прочном пластмассовом корпусе и снабжен защитой от возможных повреждений, связанных с постоянным использованием студентами. На рисунке 1.2 показан внешний вид стенда SDK-1.1.

Рисунок 1.2 – Внешний вид стенда SDK-1.1

На рисунке 1.3 дано схематическое изображение стенда SDK-1.1. Расшифровка обозначений на схеме дана в таблице 1.1.

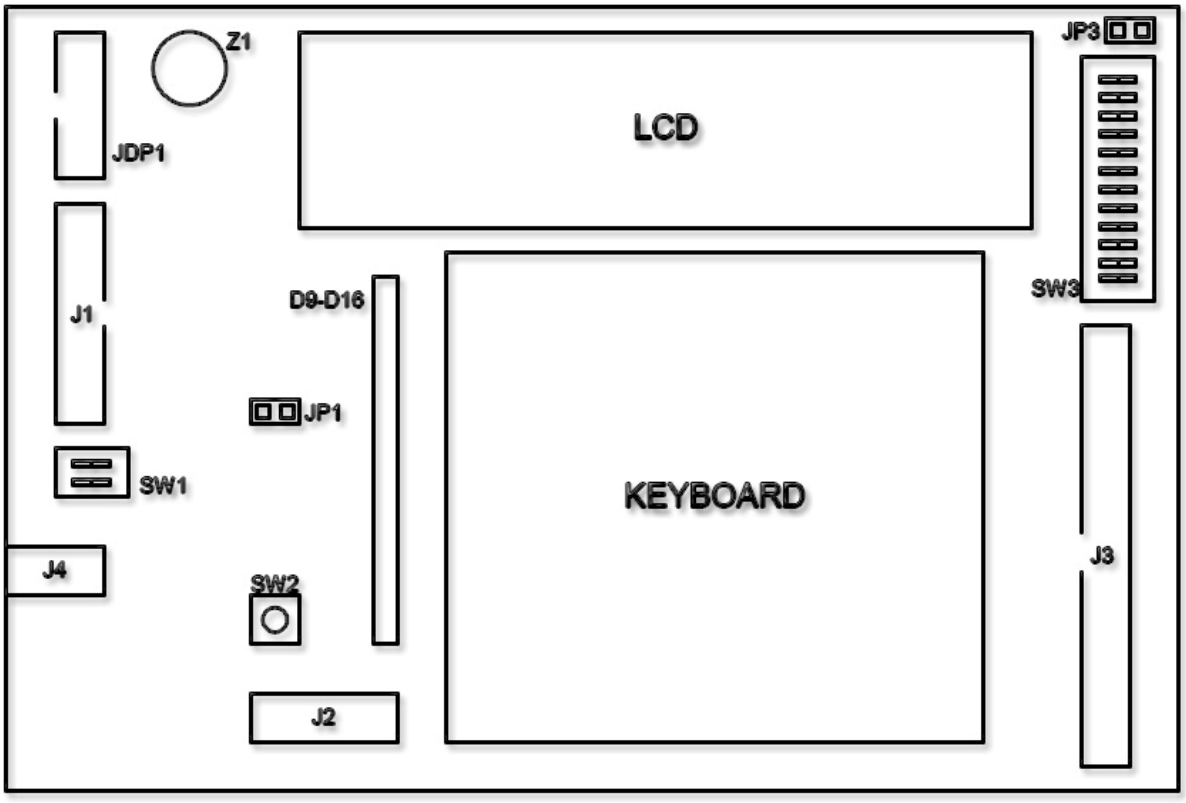


Рисунок 1.3 - Схематическое изображение стенда SDK-1.1.

Таблица 1.1 – Расшифровка обозначений на схеме стенда SDK-1.1

Элемент	Описание
LCD	Жидкокристаллический индикатор WH1602B-YGK-CP.
KEYBOARD	Матричная клавиатура AK1604A-WWB.
Z1	Звуковой пьезокерамический излучатель.
SW2	Кнопка сброса RESET.
J4	Разъем питания стенда типа "JACK", полярность безразлична.
JDP1	Разъем последовательного порта стенда.
J1	Выводы каналов АЦП и ЦАП.
SW1	Переключатель, занимающий каналы 0 и/или 1 ЦАП на входы соответствующих (0, 1) каналов ЦАП.
J3	16 линий параллельного порта ПЛИС MAX и 4 линии параллельного порта P3 микроконтроллера ADuC812 (INT0/1,T0/1).
SW3	Набор переключателей, замыкающих соответствующие выводы J3 на корпус (переключение в лог. "0").
J2	Выводы JTAG - интерфейса ПЛИС MAX.
JP1	Перемычка, замыкающая вывод PSEN микроконтроллера ADuC812 на корпус.

Продолжение таблицы 1.1

Элемент	Описание
JP3	Разъемы подключения внешней батареи питания часов реального времени PCF8583.
D9-D16	Набор сигнальных светодиодов.

Рассмотрим более подробно основные функциональные модули стенда.

Микроконтроллер ADuC812BS. Процессор ADuC812 является клоном Intel 8051 со встроенной периферией.

Основные характеристики микропроцессора:

– рабочая частота 11,0592 МГц;

– 8-канальный 12-битный АЦП со скоростью выборок 200 К/с (в режиме ПДП);

– два 12-битных ЦАП (код-напряжение);

– внутренний температурный сенсор;

– 640 байт программируемого E2PROM со страничной организацией (256 страниц по 4 байта);

– 256 байт внутренней памяти данных (участок регистров общего назначения, битовый сегмент, свободный участок, участок регистров специального назначения);

– адресное пространство 16 Мб;

– режим управления питанием;

– асинхронный последовательный ввод-вывод;

– интерфейс I2C;

– три 16-битных таймера/счетчика и таймер WatchDog.

Внешняя E2PROM. E2PROM - перепрограммируемое электрически стираемое постоянное запоминающее устройство. Объем памяти E2PROM, установленной в стенде SDK-1.1, составляет 128 байт (возможна установка E2PROM большего объема, до 32 Кб). Микросхема E2PROM взаимодействует с процессором посредством интерфейса I2C. В таблице 1.2 дана адресация микросхем E2PROM.

Основные характеристики E2PROM:

- возможность перезаписи до 1 млн. раз;
- возможность побайтной и постраничной записи (в текущей конфигурации размер страницы составляет 8 байт).

Таблица 1.2 – Адресация микросхем E2PROM

Бит	Значение	Описание
0x7	RW	Переключатель
0x6	A0	Адреса устройств (для расширения)
0x5	A1	
0x4	A2	
0x3	0	Обязательная по2следовательность для всех устройств E2PROM
0x2	1	
0x1	0	
0x0	1	

Матричная клавиатура АК1604A-WWB. Клавиатура организована в виде матрицы 4x4. Доступ к колонкам и рядам организован как чтение/запись определенного байта внешней памяти (4 бита соответствуют 4 колонкам, другие 4 бита - рядам).

Жидкокристаллический индикатор WH1602B-YGK-CP. ЖКИ работает в текстовом режиме (2 строки по 16 символов), имеет подсветку (цвет желто-зеленый).

Основные характеристики жидкокристаллического индикатора:

- габариты: 80x36x13.2 мм;
- активная область 56.21x11.5 мм;
- размеры точки 0.56x0.66 мм; размеры символа 2.96x5.56 мм;
- встроенный набор 256 символов (ASCII + кириллица);
- генератор символов с энергозависимой памятью на 8 пользовательских символов.

Часы реального времени PCF8583. PCF8583 - часы/календарь с памятью объемом 256 байт, работающие от кварцевого резонатора с частотой 32.768 кГц. Питание осуществляется ионистором (0.1 ф). Из 256 байт памяти собственно часами используются только первые 16 (8 постоянно обновляемых регистров-защелок на установку/чтение даты/времени и 8 на будильник), остальные 240 байт доступны для хранения данных пользователя. Точность измерения времени - до сотых долей секунды. Взаимодействие с процессором осуществляется через интерфейс I2C.

1.2 Распределение памяти в SDK-1.1

Адресное пространство процессора разделяется на два, не отображаемых друг на друга участка - внешнюю память и внутреннюю.

На рисунке 1.4 показана схема распределения памяти в SDK-1.1. На представленной схеме внутренняя память расположена в левой части, а внешняя – в правой части.

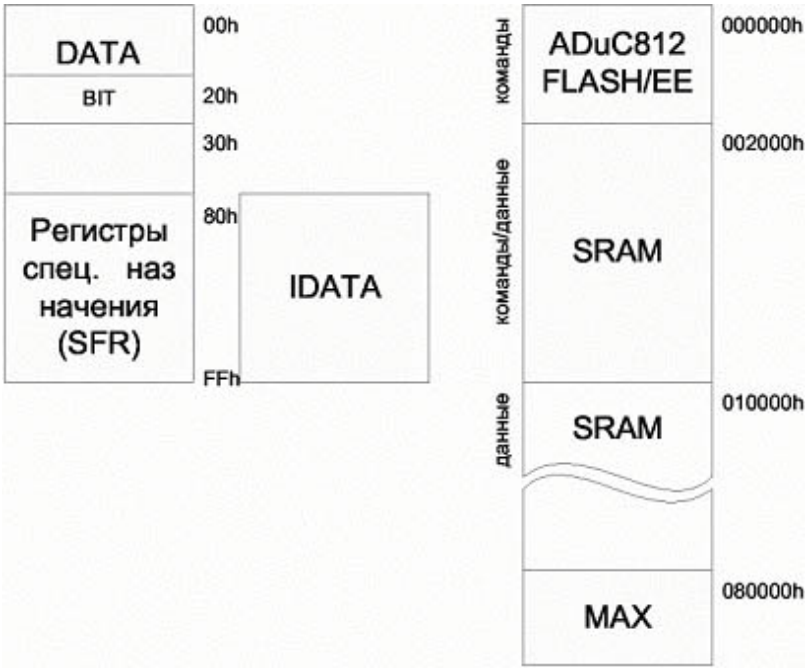


Рисунок 1.4 - Схема распределения памяти SDK-1.1

Внутренняя память. Микропроцессор ADuC812BS, являясь аналогом процессора Intel 8051, унаследовал типичную для процессоров этого семейства структуру организации внутренней памяти. Внутренняя память (256 байт) разделена на 4 участка:

- участок регистров общего назначения;
- битовый сегмент;
- свободный участок;
- участок регистров специального назначения.

В таблице 1.3 содержится информация о распределении внутренней памяти в SDK-1.1.

Стандартная для архитектуры 8051 структура внутренней памяти представлена четырьмя банками по восемь регистров общего назначения (диапазоны адресов 00h-07h, 08h-0Fh, 10h-17h, 18h-lFh), битовым сегментом (20h-2Fh), свободным участком 30h-7Fh, областью размещения SFR (регистров специаль-

ного назначения) 80h-FFh, доступной при прямой адресации, и свободной областью 80h-FFh, доступной при косвенной адресации.

Таблица 1.3 – Распределение внутренней памяти

	Регистры общего назначения: 4 банка				Битовый сегмент	Свободный участок	Регистры спец. назначения
	1	2	3	4			
Адрес	00-07	08-0F	10-17	18-1h	20-2F	30-7F	80-FF

Внешняя память. Внешняя память SDK-1.1 разбита на следующие области: AduC812 Flash/EE, SRAM, MAX. Распределение адресов внешней памяти представлено в таблице 1.4.

Таблица 1.4 - Распределение адресов внешней памяти

Flash/EE	SRAM			
	Page 1	Pages 2..7	Page 8	
0x0000 0x2000	0x02000 0x0FFFF	0x10000 0x7FFFF	0x80000 0x8FFFF	Диапазон адресов

ADuC812 Flash/EE. Память Flash/EE представляет собой постоянную память, в которой хранится сервисная программа обслуживающая стенд, в ней находится набор тестов оборудования и драйвер RS232, позволяющий загружать пользовательский программы. При подаче питания или сбросе управление передаётся по нулевому адресу и происходит инициализация всех регистров. Если пользовательская программа обратится к адресу 0, то стенд пройдёт процедуру реинициализации, что равносильно нажатию кнопки «сброс». Запись в эту область памяти возможна только в режиме программирования Flash-памяти, в простом режиме доступ к ним закрыт. Это необходимо учесть при разработке программы - код должен располагаться по адресам не ниже 0x2000. Это область, в которой располагается таблица векторов прерываний и резидентный загрузчик файлов в формате HEX в память SRAM.

Память SRAM. Статическая память имеет страничную организацию и представляет собой восемь страниц размером 64Кб каждая. Условно всю память SRAM разделяют на три участка: первая страница, страницы со второй по седьмую и восьмая страница.

Выборка кода может осуществляться только из первой страницы, и поэтому весь код должен располагаться именно здесь. Страницы со второй по седьмую могут быть использованы только для хранения данных. Восьмая стра-

ница особенна тем, что в первых 8 байтах расположены регистры микросхемы MAX 8064.

В младших адресах восьмой страницы адресного пространства (080000h-080007h) располагается 8 ячеек-регистров ПЛИС MAX8064 (MAX8128). Эта область предназначена для взаимодействия с периферийными устройствами стенда.

1.3 Карта портов ввода-вывода

В стенде SDK-1.1 ввод-вывод данных осуществляется с помощью портов микроконтроллера и микросхемы ПЛИС, которая имеет восемь регистров, отображаемых на внешнее адресное пространство процессора. Информация о портах ввода-вывода микроконтроллера и их назначении содержится в таблице 1.5.

Таблица 1.5 – Порты ввода-вывода микроконтроллера

Порт	Назначение
P0.7-P0.0	Шина адреса/данных AD(7-0) системного интерфейса
P1.7-P1.0	Аналоговый вход, линии которого мультиплексируются с линиями 7-0 АЦП.
P2.7-P2.0	Адресная шина системного интерфейса A(15-8).
P3.0	RxD - входные данные приемопередатчика UART.
P3.1	TxD - выходные данные приемопередатчика UART.
P3.2	#INT0 - сигнал внешнего прерывания 0, активный уровень - лог. "0".
P3.3	#INT1 - сигнал внешнего прерывания 1, фиктивный уровень - лог. "0".
P3.4	Счетный вход таймера-счетчика T0, активный уровень - лог. "0".
P3.5	Счетный вход таймера-счетчика T1, активный уровень - лог. "0".
P3.6	#WR - сигнал записи во внешнюю память XRAM.
P3.7	#RD - сигнал чтения из внешней памяти XRAM.

1.4 Состав и назначение ПЛИС MAX

Микросхема ПЛИС MAX имеет однобайтные регистры, с помощью которых происходит управление вводом-выводом. Данные регистры отображаются во внешнем адресном пространстве микроконтроллера как интервал адресов от 0x080000-0x80007, то есть находятся в начале восьмой страницы расширенной памяти. В таблице 1.6 содержится перечень регистров ПЛИС и их адреса.

Таблица 1.6 – Перечень регистров ПЛИС

Адрес	Регистр	Описание
080000H	KB	Регистр клавиатуры
080001H	DATA_IND	Регистр шины данных LCD
080002H	EXT_LO	Регистр параллельного порта (младший байт)
080003H	EXT_HI	Регистр параллельного порта (старший байт)
080004H	ENA	Регистр разрешений
080006H	C_IND	Регистр управления LCD
080007H	SV	Регистр состояния светодиодов

Регистр клавиатуры KB. Ввод данных непосредственно от пользователя осуществляется с помощью матричной клавиатуры.. Таблица 1.7 содержит информацию о регистре клавиатуры. Назначение битов регистра клавиатуры приведено в таблице 1.8.

Таблица 1.7 – Регистр клавиатуры KB

7	6	5	4	3	2	1	0
R	R	R	R	W	W	W	W
ROW				COL			

Таблица 1.8 – Назначение битов регистра KB

Биты	Поле	Описание
0..3	COL	Поле предназначено для сканирования клавиатуры (столбца). Сканирование производится посредством записи логического "0" в один из разрядов поля.
4..7	ROW	Поле предназначено для считывания данных с клавиатурной матрицы (строки). Если ни одна из кнопок в строке не нажата, все биты поля ROW содержат логические "1", иначе - логический "0".

Регистр шины данных ЖКИ. Адрес регистра шины данных ЖКИ DATA_IND - 080001H. Значение после сброса - 00000000B. Таблица 1.9 содержит информацию о регистре шины данных ЖКИ. В таблице 1.10 приведено назначение битов регистра DATA_IND.

Таблица 1.9 - Регистр шины данных ЖКИ DATA_IND

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
D7	D6	D5	D4	D3	D2	D1	D0

Таблица 1.10 - Назначение битов регистра DATA_IND

Биты	Поле	Описание
0..7	D0..D7	Регистр DATA_IND позволяет устанавливать данные на шине данных ЖКИ и считывать их оттуда. Для организации взаимодействия с ЖКИ (формирования временных диаграмм чтения и записи) необходимо использование регистра C_IND.

Регистр данных параллельного порта EXT_LO. Адрес 080002H. Значение после сброса 00000000B. В таблице 1.11 приведены сведения о регистре данных параллельного порта EXT_LO. Таблица 1.12 содержит информацию о назначении битов регистра EXT_LO.

Таблица 1.11 - Регистр данных параллельного порта EXT_LO

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
D7	D6	D5	D4	D3	D2	D1	D0

Таблица 1.12 - Назначение битов регистра EXT_LO

Биты	Поле	Описание
0..7	D0..D7	Регистр EXT_LO позволяет считывать и записывать биты 0..7 параллельного порта. Для того, чтобы данные из регистра попали на выход, необходимо установить бит EN_LO в логическую "1" (см. регистр ENA).

Регистр данных параллельного порта EXT_HI. Адрес 080003H. Значение после сброса 00000000B. В таблице 1.13 приведена информация о регистре данных параллельного порта EXT_HI. Таблица 1.14 содержит информацию о назначении битов регистра EXT_HI.

Таблица 1.13 - Регистр данных параллельного порта EXT_HI

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
D7	D6	D5	D4	D3	D2	D1	D0

Таблица 1.14 - Назначение битов регистра EXT_HI

Биты	Поле	Описание
0..7	D0..D7	Регистр EXT_HI позволяет считывать и записывать биты 0..7 параллельного порта. Для того, чтобы данные из регистра попали на выход, необходимо установить бит EN_HI в логическую "1" (см. регистр ENA). Для чтения данных необходимо установить этот бит в логический "0".

Регистр управления ENA. Адрес 080004H. Значение после сброса xx000000B. В таблице 1.15 содержится информация о регистре управления ENA. Назначение и описание битов регистра ENA даны в таблице 1.16.

Таблица 1.15 - Регистр управления ENA

7	6	5	4	3	2	1	0
-	-	W	W	W	W	W	W
-	-	INT0	SND2	SND1	SND0	EN_HI	EN_LO

Таблица 1.16 - Назначение битов регистра ENA

Биты	Поле	Описание
0	EN_LO	Бит EN_LO нужен для управления младшими 8 разрядами (биты 0..7) 16-разрядного порта ввода-вывода. Если в EN_LO логический "0", то порт ввода-вывода переводится в Z-состояние и появляется возможность чтения данных из EXT_LO. При записи логической "1" порт EXT_LO переключается на вывод и данные, записанные в регистр EXT_LO, попадают на выход порта ввода-вывода.
1	EN_HI	Полностью аналогичен EN_LO. Управляет старшей частью 16-разрядного порта (биты 8..15).

Продолжение таблицы 1.16

Биты	Поле	Описание
2..4	SND2-SND0	Выход звукового ЦАП. Задает уровень напряжения на динамике. Позволяет формировать звуковые сигналы различной тональности и громкости.
2..4	SND2-SND0	Выход звукового ЦАП. Задает уровень напряжения на динамике. Позволяет формировать звуковые сигналы различной тональности и громкости.
5	INT0	При записи логического "0" в этот бит на вход INT0 ADuC812 также попадает логический "0". Бит можно использовать для формирования внешнего прерывания для микроконтроллера.

Регистр управления ЖКИ C_IND. Адрес 080006H. Значение после сброса xxxxx000B. В таблице 1.17 содержится информация о регистре управления ЖКИ C_IND. Таблица 1.18 дает сведения о назначении битов регистра C_IND.

Таблица 1.17 - Регистр управления ЖКИ C_IND

7	6	5	4	3	2	1	0
-	-	-	-	W	W	W	W
-	-	-	-	Reserved	RS	RW	E

Таблица 1.18 - Назначение битов регистра C_IND

Биты	Поле	Описание
0	E	Бит управления входом "E" ЖКИ. Наличие положительного импульса на входе "E" позволяет зафиксировать данные на шине ЖКИ (данные, сигналы RW и RS должны быть уже установлены).
1	RW	Бит переключения шины данных ЖКИ 0 - запись,
2	RS	Бит переключения режимов команды/данных ЖКИ. 1 - данные, 0 - команды.
3	Reserved	используется в технологических целях. Для корректной работы ЖКИ необходимо при каждой записи в данный регистр устанавливать бит в 1.

Регистр управления светодиодами SV. Адрес 080007H. Значение после сброса 00000000B. В таблице 1.19 содержится информация о регистре управления светодиодами. Назначение и описание битов регистра SV даны в таблице 1.20.

Таблица 1.19 - Регистр управления светодиодами SV

7	6	5	4	3	2	1	0
W	W	W	W	W	W	W	W
D7	D6	D5	D4	D3	D2	D1	D0

Таблица 1.20 - Назначение битов регистра SV

Биты	Поле	Описание
0..7	D0..D7	Биты управления светодиодами. Подача логической "1" зажигает светодиоды.

1.5 Доступ к регистрам ПЛИС MAX

Для доступа к регистрам ПЛИС нужно переключить страничный регистр DPP на восьмую страницу памяти. Адреса регистров внутри страницы находятся в диапазоне от 0 до 7.

Доступ к регистрам возможен через указатель: unsigned char xdata *regnum. Ниже приведен пример функций для доступа к регистрам ПЛИС.

```
#define MAXBASE 8 // Страница памяти, в которую отображаются регистры ПЛИС
/*
WriteMAX       - Запись байта в регистр ПЛИС
Regnum - адрес регистра ПЛИС
val        - записываемое значение
результат- нет
*/
void WriteMax (unsigned char xdata *regnum, unsigned char val)
{
   unsigned char oldDPP = DPP;
   DPP = MAXBASE;
   *regnum = val;
   DPP = oldDPP;
}
/*
ReadMAX        - Чтение байта из регистра ПЛИС
Regnum - адрес регистра ПЛИС
Результат        - прочитанное значение
```

```
*/
unsigned char ReadMax(unsigned char xdata *regnum)
{
   unsigned char oldDPP = DPP;
   unsigned char val = 0;
   DPP = MAXBASE;
   val = *regnum;
   DPP = oldDPP;
   return val;
}
```

Проблемы, часто возникающие при доступе к регистрам ПЛИС. Необходимо помнить, что при переключении страниц становятся недоступными все данные, размещенные в странице 0.

Для того, чтобы избежать проблем со страничным регистром DPP, нужно использовать специальные функции для доступа к ПЛИС, которые перед началом работы с регистрами ПЛИС будут запоминать старое значение страничного регистра, а по окончании работы возвращать его обратно.

Необходимо следить, чтобы передаваемые в регистры ПЛИС значения хранились во внутренней памяти микроконтроллера (DATA, IDATA). Убедиться, что передаваемая информация не содержится во внешней памяти контроллера (XDATA), достаточно легко: так как для доступа к внешней памяти в микроконтроллерах семейства C51 используется регистр DPTR, нужно просто просмотреть листинг программы и убедиться в том, что для доступа к переменным компилятор не использует DPTR.

1.6 Периферийные микросхемы

К периферийным микросхемам стенда SDK-1.1 относятся: жидкокристаллический индикатор, часы/календарь, таймер/счетчик. Далее дано развернутое описание каждой из этих микросхем.

1.6.1 Модуль ЖКИ

Описание функций модуля ЖКИ. Модуль ЖКИ встроен в контроллер (БИС) и имеет два 8-битовых регистра: регистр команд (IR) и регистр данных (DR).

Регистр команд хранит коды таких операций, как очистка дисплея, перемещение курсора, а также информацию об адресах памяти отображаемых данных (DDRAM) и генератора символов (CGRAM). В регистр команд можно только записывать информацию из микропроцессора. Регистр данных временно хранит данные, предназначенные для записи или чтения из DDRAM или

CGRAM. Когда адресная информация записывается в регистр команд, данные из DDRAM или CGRAM сохраняются в регистре данных. Эти два регистра можно выбрать с помощью регистрового переключателя (RS). В таблице 1.21 приведены коды регистра команд.

Таблица 1.21 – Коды регистра команд

RS	R/W	Команда
0	0	IR используется для внутренних команд (очистка дисплея и т.д.).
0	1	Считывание флага занятости (DB7) и счетчика адреса (от DB0 до DB7).
1	0	Запись данных в DDRAM или CGRAM (из регистра данных в DDRAM или CGRAM).
1	1	Чтение данных из DDRAM или CGRAM (из DDRAM или CGRAM в регистр данных).

Флаг занятости (BF). Если флаг занятости равен 1, это значит, что БИС занята выполнением внутренних операций и следующая команда не может быть принята. Если RS=0 и R/W=1, содержимое флага занятости передается в бит DB7. Следующая команда должна быть записана только при значении флага занятости, равном 0.

Счетчик адреса (AC). Счетчик адреса (AC) назначает адреса и DDRAM, и CGRAM.

Память данных ЖКИ (DDRAM). Эта память используется для хранения данных, выводимых на дисплей. Один символ представлен в виде 8-битного кода. Объем памяти составляет 80*8 битов или 80 символов. На рисунке 1.5 показана структура памяти данных ЖКИ (DDRAM).

Рисунок 1.5 – Структура памяти данных ЖКИ (DDRAM)

Ниже в таблице 1.22 приведена схема соответствия между адресами DDRAM и позициями ЖКИ.

Таблица 1.22 - Соответствие между адресами DDRAM и позициями ЖКИ

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Генератор символов, встроенный в ПЗУ (CGROM). CGROM генерирует символы размером 5*8 или 5*10 точек на основе 8-битных кодов символов.

Генератор символов ОЗУ (CGRAM). В CGRAM пользователь может программно генерировать символы. Команды CGRAM приведены в таблице 1.23. Можно определить 8 символов размером 5*8 точек и 4 символа размером 5*10 точек.

Таблица 1.23 – Таблица команд CGRAM

Команда	Код операции										Описание	Время выпол. (fosc=270 КГц)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Очистка экрана	0	0	0	0	0	0	0	0	0	1	Запись «00H» в DDRAM и установка адреса DDRAM на «00H» из AC	1.53 мс
Возврат в начало строки	0	0	0	0	0	0	0	0	1	-	Установка адреса DDRAM на «00H» из AC и возврат курсора в начало строки, если он был смещен. Содержимое DDRAM не меняется	1.53 мс
Начальные установки	0	0	0	0	0	0	0	1	1/D	SH	Задает направление перемещения курсора и разрешает сдвиг сразу всех символов.	39 мс
Дисплей ON/OFF	0	0	0	0	0	0	1	D	C	B	Устанавливает/отключает биты, отвечающие за режим дисплея (D), отображение курсора (C), мерцание курсора (B).	39 мс
Передвиж. курсора по экрану	0	0	0	0	0	1	S/C	R/L	-	-	Установка бита движения курсора и смещения всех символов, указание направления смещения без изменения данных в DDRAM.	39 мс
Функц. установки	0	0	0	0	1	DL	N	F	-	-	Установка длины данных (DL: 8-бит/4-бита), количества строк на дисплее (N:2-строки или 1) и размера символов (F:5x11 точек/5*8 точек).	39 мс
Установка адреса CGRAM	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Установка адреса CGRAM в счетчик адреса.	39 мс
Установка адреса DDRAM	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Установка адреса DDRAM в счетчик адреса.	39 мс

Продолжение таблицы 1.23

Команда	Код операции										Описание	Время выпол. (fosc=270 КГц)
Чтение флага занятости и адреса	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Прочитав флаг занятости, можно определить, занят ли контроллер выполнением внутренних операций. Также можно прочесть содержимое счетчика адреса.	0 мс
Записать данные в память	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Запись данных во внутреннюю память (DDRAM/CGRAM).	43 мс
Чтение данных из памяти	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Чтение данных из внутренней памяти (DDRAM/CGRAM).	43 μс

1.6.2 Часы / календарь

Часы / календарь с ОЗУ 240x8 бит PCF8583. Микросхема PCF8583, содержащая часы/календарь, имеет оперативную память на МОП-транзисторах объемом в 2048 бит, состоящую из 256 слов по 8 бит. Адреса и данные передаются последовательно через двунаправленную шину I2C. Встроенный регистр адреса автоматически наращивается после чтения или записи каждого байта данных.

Микросхема PCF8583 обладает следующими особенностями:

- I2C-интерфейс;
- диапазон рабочих напряжений питания в пределах от 1.0 В до 6.0 В (при температуре от 0 до +70 °С);
- низковольтная память объемом 240x8 бит;
- напряжение сохранения данных от 1.0 В до 6.0 В;
- рабочий ток (при частоте fscL = 0 Гц): максимум 50 мА;
- календарь на 4 года;
- универсальный таймер с сигналом и индикацией;
- 12-или 24-часовой формат времени;
- внешний генератор- 32.768 кГц или 50 Гц;
- последовательная шина ввода/вывода (I2C);
- автоматическое наращивание адреса при работе с памятью;
- программируемые динамик, таймер и функции прерывания;
- адреса Slave-устройств;
- чтение: A1 или A3;
- запись: A0 или A2.

Адресный вывод A0 используется для программирования адресов устройств, что позволяет подсоединять к шине два устройства без использования какого-либо дополнительного аппаратного обеспечения. На рисунке 1.6 показана структурная схема микросхемы PCF8583.

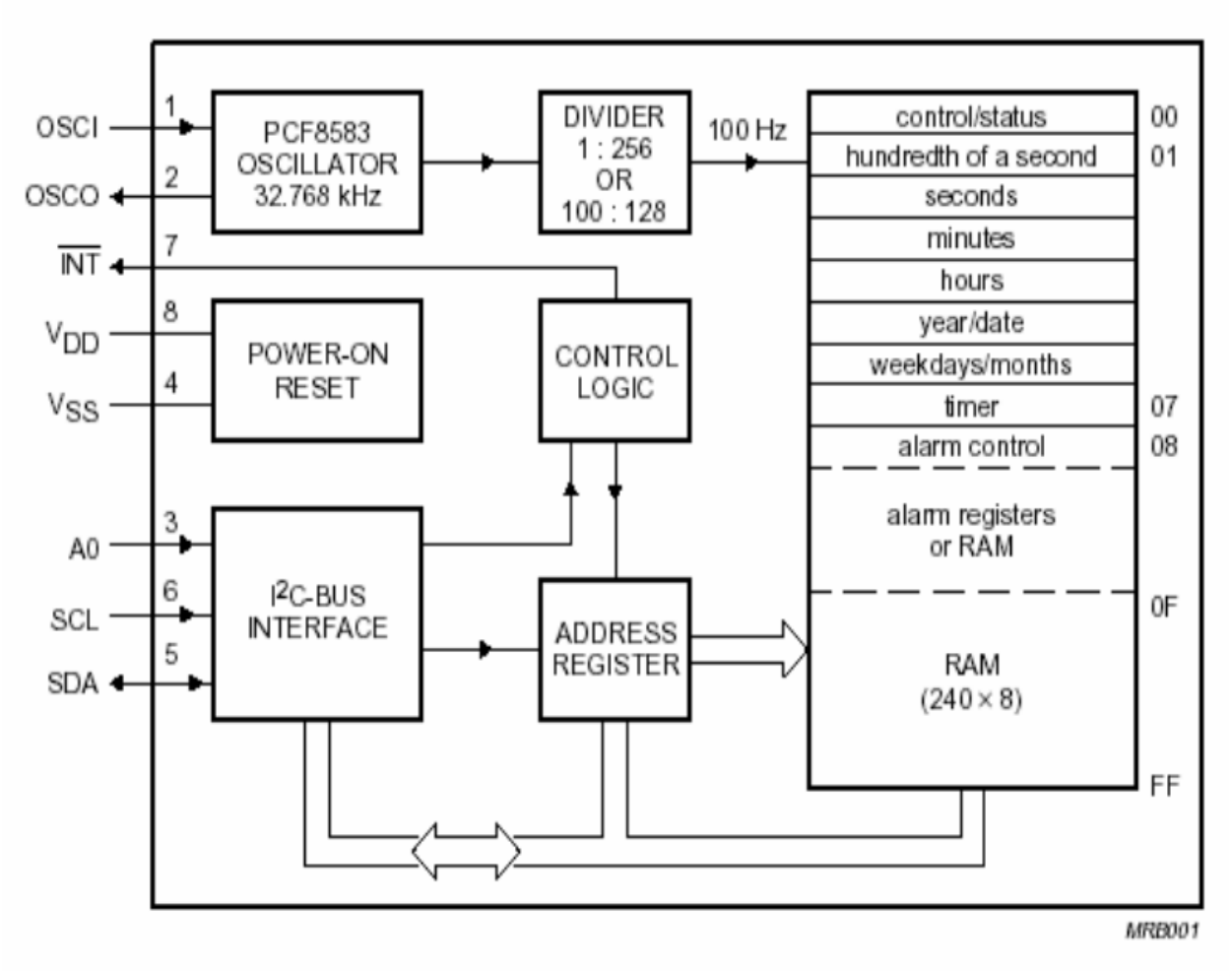


Рисунок 1.6 - Структурная схема микросхемы PCF8583

Обозначения:

– PCF8583 OSCILLATOR - тактовый генератор;
– POWER-ON RESET - сброс по включению питания;
– I2C-BUS INTERFACE - интерфейс шины I2C;
– DIVIDER-делитель;
– CONTROL LOGIC - логика управления;
– ADDRESS REGISTER - адресный регистр.

Встроенная микросхема генератора, работающая на частоте 32.768 кГц, и первые 8 байт оперативной памяти используются для часов, календаря и функций счетчика. Следующие 8 байт могут быть запрограммированы на использование в качестве регистров сигнализации, или же к ним можно обращаться как к свободным адресам памяти. Остальные 240 байт относятся к оперативной памяти.

В таблице 1.24 содержится информация об устройстве памяти микросхемы PCF8583.

Таблица 1.24 – Устройство памяти микросхемы PCF8583

Назначение ячеек	Адрес
Управление/состояние	00h
Одна сотая доля секунды	01h
Секунды	02h
Минуты	03h
Часы	04h
Год/дата	05h
Дни недели/месяцы	06h
Таймер	07h
Управление сигналом	08h
Регистры сигнала или ячейки памяти	0Fh
ОЗУ (240x8)	10h-FFh

Описание функций. Микросхема PCF8583 содержит 8-битную оперативную память объемом 256 байт с 8-битным адресным регистром, осуществляющим автоматическое инкрементирование адреса, встроенную микросхему генератора (частота 32.768 кГц), делитель частоты, последовательную двунаправленную шину I2C и схему, осуществляющую сброс по включению питания.

Первые 16 байт ОЗУ (адреса памяти от 00 до 0F) представляют собой адресуемые 8-битовые регистры специального назначения. Первый регистр (адрес 00) используется в качестве регистра управления/состояния. Регистры по адресам с 01 по 07 - счетчики для функций часов. Регистры, расположенные по адресам с 08 по 0F, могут быть запрограммированы в качестве регистров сигнала или использованы как обычные регистры памяти (когда сигналы отключены).

Режимы счетчика. При программировании регистра управления/состояния может быть установлен режим часов на частоте 32.768 кГц, режим часов на частоте 50 Гц или режим счетчика событий. В том случае, если выбран режим часов, сотые доли секунды, секунды, минуты, часы, дата, месяц (календарь на 4 года) и дни недели хранятся в двоично-десятичном формате. Режим счетчика используется для подсчета импульсов, выдаваемых на вход генератора (к выводу OSCO ничего не подключается). Счетчик событий хранит до 6 цифр данных.

При чтении одного из счетчиков (адреса с 01 по 07) содержимое всех счетчиков стробируется в регистры-защелки в начале цикла чтения. Таким об-

разом предотвращаются ошибки чтения счетчика. При записи в счетчик с другими счетчиками ничего не происходит.

Режим сигнализации. При установке в регистре управления/состояния бита, разрешающего сигнал, активируется регистр управления сигналом (адрес 08).

Используя настройки регистра управления сигналом, можно запрограммировать срабатывание сигнала при наступлении определенной даты, ежедневного сигнала, сигнала по дням недели и по времени. В режиме часов регистр таймера (адрес 07) может быть запрограммирован для подсчета сотых долей секунды, секунд, минут, часов и дней. Подсчет дней ведется, если не запрограммирован сигнал.

Каждый раз при наступлении "сигнального" события устанавливается соответствующий флаг регистра управления/состояния. Событие по таймеру устанавливает флаг сигнала, а в случае переполнения таймера устанавливается флаг таймера. В случае установки (разрешения) флага сигнала или таймера происходит включение вывода прерывания с открытым стоком (с активным низким уровнем выходного сигнала). Флаги остаются установленными до тех пор, пока они не будут сброшены напрямую в результате операции записи.

Если сигнал отключен (то есть бит 2 регистра управления/состояния равен 0), регистры сигналов (адреса с 08 по 0F) могут быть использованы как свободные ячейки памяти.

1.6.3 Таймеры/счетчики

Для работы с таймерами/счётчиками в микроконтроллерах семейства 8051 используются следующие регистры: TH0, TL0, TH1, TL1, TCON, TMOD.

Регистры TH0, TL0, TH1, TL1 (Timer/counter Low (High) byte). Исходное (текущее) состояние j-го таймера/счетчика T/Cj в микроконтроллере определяется программно доступными регистрами THj, TLj. Причем регистр THj - старшие, а регистр TLj - младшие 8 разрядов. Новая загрузка THj, TLj сразу же означает новую величину с которой будет начат счет в T/Cj, а старая теряется. Если загрузка произведена при включенном T/Cj, то счет продолжается с новой величины. Очередность загрузки регистров THj, TLj произвольная. Выключение T/Cj не искажает код, находящийся в THj, TLj. Таймер/счетчик T/Cj можно выключить, через произвольное время вновь включить и счет начнется с той величины, которая была в регистрах THj, TLj на момент выключения.

Регистр TCON (Timer/counter Control). Наименование и назначение разрядов регистра TCON приведены в таблице 1.25. Все разряды этого регистра доступны по записи и по чтению.

Таблица 1.25 – Разряды регистра TCON

Биты	Наименование	Назначение
7	TF1	Флаг переполнения T/C1.
6	TR1	Бит включения T/C1. TR1=1 - включен, TR1=0 - выключен.
5	TF0	Флаг переполнения T/C0.
4	TR0	Бит включения T/C0. TR0=1 - включен, TR0=0 - выключен.
3	IE1	Флаг запроса внешнего прерывания INT1.
2	IT1	Бит, определяющий вид прерывания INT1. IT1=0 - прерывание по уровню (низкому), IT1=1 - прерывание по фронту (переход из"1" в "0").
1	IE0	Флаг запроса внешнего прерывания INT0.
0	IT0	Бит, определяющий вид прерывания INT0. IT0=0 - прерывание по уровню (низкому), IT0=1 - прерывание по фронту (переход из"1" в "0").

Флаг TFj (j={0 | 1}) аппаратно устанавливается в "1" при переходе T/Cj из состояния "все единицы" в состояние "все нули". Если прерывание от T/Cj разрешено, то установка флага TFj вызовет прерывание. Бит TFj аппаратно сбрасывается в "0" при обращении к подпрограмме обработки прерывания. Флаг IEj аппаратно устанавливается в "1" от внешнего прерывания: от низкого уровня или перехода из "1" в "0" сигнала прерывания. Если при этом внешнее прерывание разрешено, то осуществляется переход к подпрограмме его обслуживания. Сброс флага IEj выполняется аппаратно при обслуживании прерывания только в том случае, когда ITj=1.

Регистр TMOD (Timer/counter Mode). Наименование и назначение разрядов регистра TMOD приведены в таблице 1.26. Все разряды этого регистра доступны по записи и по чтению.

Таблица 1.26 – Наименование и назначение разрядов регистра TMOD

Биты	Наименование	Назначение
7	GATE1	Бит разрешает (запрещает) управлять T/C1 от внешнего вывода. GATE1=1 - управление разрешено, GATE1=0 -

		управление запрещено.

Продолжение таблицы 1.26

<table>
<tr><td>Биты</td><td>Наимено-вание</td><td colspan="3">Назначение</td></tr>
<tr><td>6</td><td>C/T1</td><td colspan="3">Бит определяет работу T/C1 в качестве таймера (C/T1=0), счетчика внешних событий (C/T1=1).</td></tr>
<tr><td rowspan="5">5
4</td><td rowspan="5">M1.1
M0.1</td><td colspan="3">Биты определяют один из 4-х режимов работы T/C1.</td></tr>
<tr><td>M1.1</td><td>M1.0</td><td>Режим</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>2</td></tr>
<tr><td>3</td><td>GATE0</td><td colspan="3">Бит разрешает (запрещает) управлять T/C0 от внешнего вывода.
GATE0=1 - управление разрешено,
GATE0=0 - управление запрещено.</td></tr>
<tr><td>2</td><td>C/T0</td><td colspan="3">Бит определяет работу T/C0 в качестве таймера (C/T0=0), счетчика внешних событий (C/T0=1).</td></tr>
<tr><td rowspan="6">1</td><td rowspan="6">M1.0
M0.0</td><td colspan="3">Биты определяют один из 4-х режимов работы T/C0.</td></tr>
<tr><td>M1.1</td><td>M1.0</td><td>Режим</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>2</td></tr>
<tr><td>1</td><td>1</td><td>3</td></tr>
</table>

При работе в качестве таймера содержимое T/Cj (j={0 | 1}) инкрементируется с частотой f/12, где f есть частота синхронизации микроконтроллера. При работе T/Cj в качестве счетчика внешних событий, его содержимое инкрементируется в ответ на переход из "1" в "0" сигнала на j-ом счетном входе микроконтроллера. Для надежной работы T/Cj в режиме счетчика необходимо, чтобы максимальная частота указанного сигнала была не более f/24 , а уровень этого сигнала оставался неизменным в течение как минимум одного машинного цикла (12/f).

Таймер/счетчик T/Cj в режиме 0 (1) представляет собой устройство на основе 13- (16-) разрядного регистра, состоящего из 8-ми разрядов регистра THj и 5-ти младших разрядов (8-ми разрядов) регистра TLj. В режиме 2 T/Cj представляет собой устройство на основе 8-разрядного регистра TLj. При каждом переполнении TLj кроме установки в регистре TCON флага TFj происходит автозагрузка регистра TLj содержимым THj, причем указанная автозагрузка не

влияет на содержимое регистра THj. Таймер/счетчик T/C1 в режиме 3 заблокирован (значение кода в регистрах TH1, TL1 не изменяется). Эффект такой же, как при сбросе TR1 в "0". Таймер/счетчик T/C0 в режиме 3 представляет собой два независимых устройства на основе регистров TH0 и TL0. Устройство на основе TL0 может работать в режиме таймера или в режиме счетчика и при переполнении устанавливает флаг TF0. За этим устройством сохраняются биты управления TR0, GATE0, C/T0 . Устройство на основе регистра TH0 может работать только в режиме таймера. Оно использует бит включения TR1, при переполнении выставляет флаг TF1. Других битов управления устройство на основе TH0 не имеет.

2 Программное обеспечение стенда SDK-1.1

Для программирования стенда может использоваться любой транслятор ассемблера или С для ядра 8051, например, пакет µVision (Keil Software). До начала программирования на языке С рекомендуется внимательно ознакомиться с документацией по используемому компилятору, так как компиляторы для микроконтроллеров имеют нестандартные расширения.

Основные этапы программирования стенда:

- подготовка программы в текстовом редакторе или среде программирования;
- транслирование исходного текста и получение загрузочного HEX-модуля программы;
- подготовка и загрузка HEX-модуля в стенд через интерфейс RS232C с помощью поставляемых инструментальных систем. Под подготовкой понимается добавление в конец модуля строчки со стартовым адресом программы, то есть адреса, по которому передается управление после загрузки в стенд;
- прием и обработка HEX-модуля резидентным загрузчиком HEX202, передача управления загруженной программе.

На рисунке 2.1 показаны этапы программирования стенда SDK-1.1.

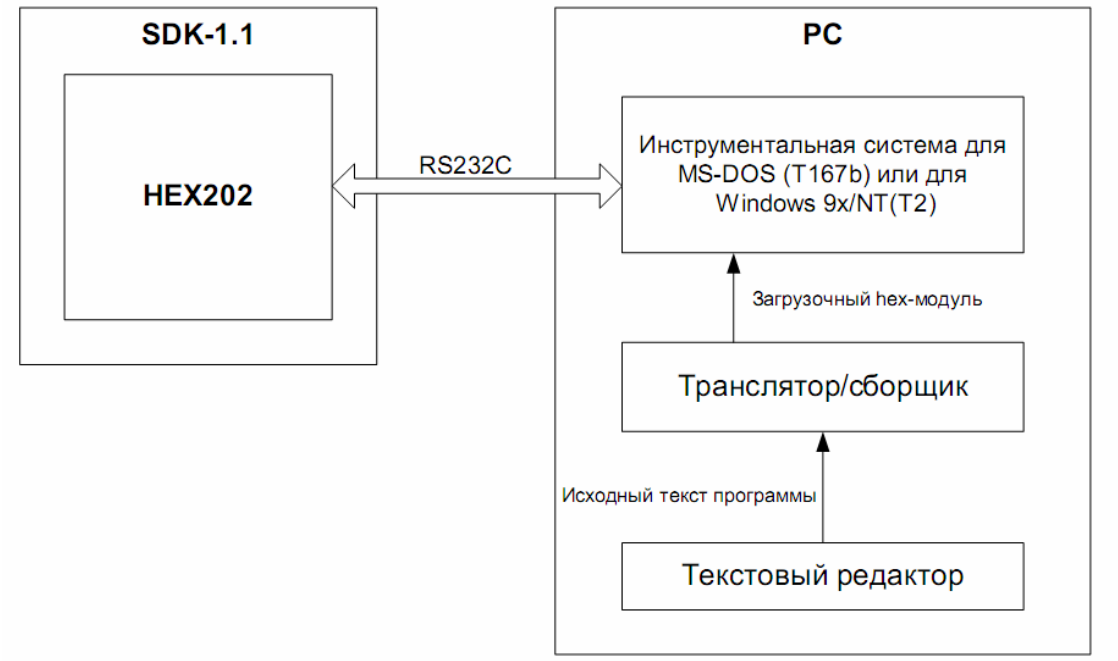


Рисунок 2.1 – Этапы программирования на стенде SDK-1.1

2.1 Инструментальные средства фирмы Keil Software

Keil Software поддерживает все стадии разработки приложения: создание исходного файла на С или Ассемблере, трансляцию, исправление ошибок, линкование объектных файлов, тестирование приложения.

В пакете Keil Software содержатся следующие средства разработки для микроконтроллера 8051:

- C51 - компилятор С;
- макроассемблер A51;
- динамический загрузчик/компоновщик BL51;
- конвертер объектных файлов OC51;
- конвертер объектных и HEX-файлов OH51;
- менеджер библиотек LIB51;
- симулятор dScope-51 (для Windows);
- отладчик/компилятор mVision/51 (для Windows);
- операционная система реального времени (Real-Time Operating System - RTX).

На рисунке 2.2 приведена схема средств разработки, входящих в пакет Keil Software.

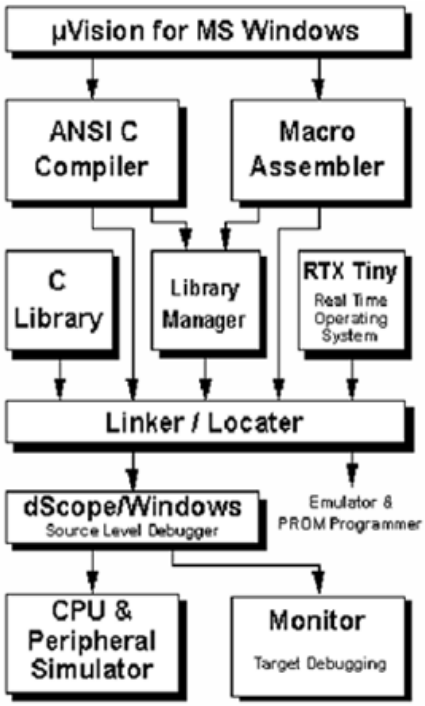


Рисунок 2.2 - Схема средств разработки, входящих в пакет Keil Software

Одним из наиболее удобных пакетов, предназначенных для разработки проектов для ядра 8051 является «uVision» выпускаемый фирмой Keil Software.

Данный пакет включает в себя средства управления проектом, транслятор Ассемблера, компилятор языка C51, и средства отладки приложения.
Для создания программы на языке C51 необходимо создать проект, используя функцию “New Project” в меню “Project”. На рисунке 2.3 показан скриншот окна uVision.

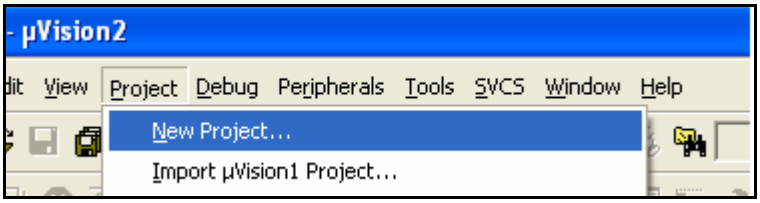


Рисунок 2.3 - Создание проекта

В появившемся диалоговом окне, необходимо ввести имя проекта и выбрать папку, в которой будет располагаться созданный проект. Далее последует диалоговое окно, в котором будет предложено выбрать целевое устройство. Как целевое устройство необходимо выбрать ADuC812 из раздела “Analog Device”. На рисунке 2.4 показан этап выбора целевого устройства.

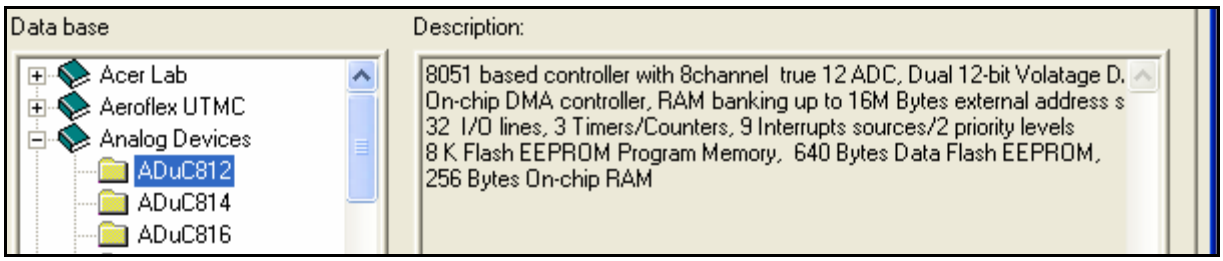


Рисунок 2.4 - Выбор целевого устройства

После создания проекта можно создавать в нём файлы, либо включать уже имеющиеся. Для создания файла необходимо выбрать команду “New” в меню ”File”. Чтобы создаваемый файл мог быть включён в менеджер проекта, при сохранении необходимо указать расширение файла xxx.c.

Для явного указания компилируемых файлов, файлы с исходным кодом нужно подключить к «целевой группе». Для этого в менеджере проекта в контекстном меню необходимо выбрать пункт “Add files to group”. На рисунке 2.5 показан этап добавления файла в группу. После включения файла, содержащего описание входной точки (функции void main(void)) необходимо произвести настройку линкования программы. В адресном пространстве микроконтроллера адреса с 0x0000 по 0x2000 отведены под резидентный загрузчик, и запись в них возможно только в режиме программирования FLESH/EE, поэтому запись пользовательских программ в эти адреса не допускается. Также не рекоменду-

ется писать исполняемый код программы в адреса 0x2000-0x2100 так как они используются в случае перехвата прерываний. Настроить компилятор на использование определённых адресов можно указав интервалы памяти для хранения кода в настройках «цели». На рисунке 2.6 показан процесс настройки компилятора HEX-файлов.

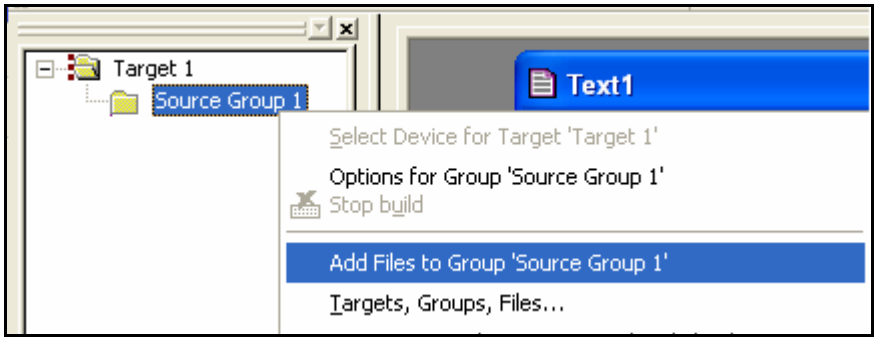


Рисунок 2.5 - Добавление файла в группу

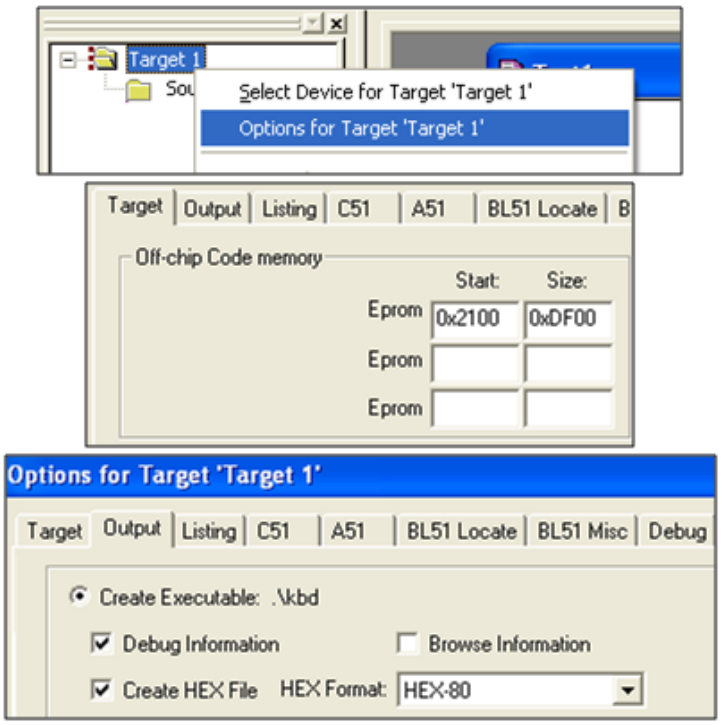


Рисунок 2.6 - Настройка компилятора HEX-файлов

Также для компиляции проекта в исполняемый модуль HEX-80 необходимо установить соответствующую галочку в закладке «output». Внимание: так как стенд SDK 1.1 не находится под управлением какой либо ОС то возвращаться из выполненной программы ему некуда, поэтому все программы необходимо строить по принципу бесконечного цикла, либо вставлять бесконечный цикл (while(1)) в конец кода.

Если в исходном тексте нет ошибок, компилятор создаст загрузочный HEX модуль в папке с проектом. Однако, этот файл не готов к загрузке в RAM стенда. Передача управления загруженной программе осуществляется по принятию строки вида: 02AAA060000SS<cr>. Где AAA- адрес точки входа. Данная строка должны быть добавлена в каждый загружаемый файл.

Для начала необходимо этот адрес узнать. При использовании программы, содержащей одну функцию main(), адрес точки входа совпадает с адресом указанным как начало банка кода. Для того чтобы узнать адрес входной точки, необходимо запустить режим отладки. На рисунке 2.7 показан скриншот выбора режима отладки.

Рисунок 2.7 - Режим отладки

Менеджер проекта заменится информацией о системных регистрах, в которых можно узнать информацию о состоянии указателя команды, который в начале сессии указывает на точку входа. Скриншот, содержащийся на рисунке 2.8, показывает состояние системных регистров.

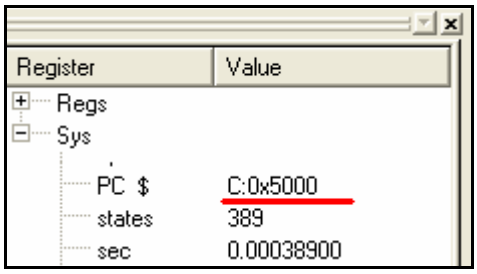


Рисунок 2.8 - Состояние системных регистров.

Загрузка HEX- модуля в память SDK1.1. Для добавления стартового адреса в HEX модуль и загрузки модуля в стенд используется утилита t2.exe. Скопируйте данную утилиту в папку с проектом. Подключите стенд к параллельному порту и включите питание стенда. Если стенд находится в тестовом режиме, выберите пункт загрузка HEX. Запустите утилиту t2.exe.

В появившейся консоли наберите команду «N DIV OPENCHANEL», где: N-номер COM-порта, DIV делитель частоты.
Пример: 1 12 openchanel.
Или opencom1, opencom2.

Далее необходимо добавить адрес точки входа к HEX- модулю: Адрес 0x0 addhexstart name

Пример: 0x2100 0x0 addhexstart 1.hex

После этого можно загружать программу.

Loadhex name. На рисунке 2.9 показана сессия консоли t2.

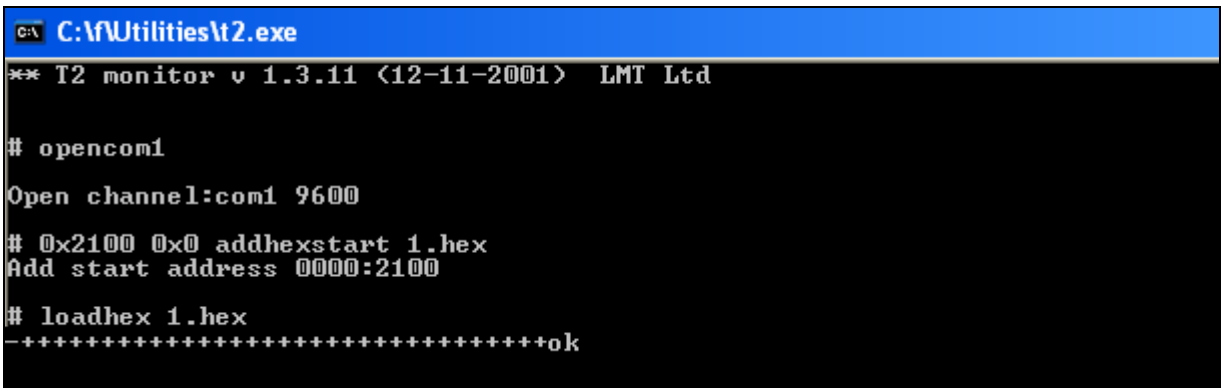


Рисунок 2.9 - Сессия в консоли t2

2.2 Расширение языка С

Компилятор языка Си фирмы Keil Software. Кросс-компилятор языка С предназначен для создания приложений для процессоров семейства 8051 на языке программирования С. Компилятор С51 поддерживает стандарт ANSI С, разработан специально для 8051 семейства и позволяет создавать программы на языке С, сохраняя эффективность и скорость оптимизации Ассемблера. Расширения, включенные в инструментальные средства Keil, обеспечивают полный доступ к ресурсам микроконтроллеров 8051. Для локализации ошибок полезно воспользоваться выходными данными С-компилятора или средствами PC-Link.

Отличительные черты компилятора:

1) Девять основных типов данных, включая быструю 32-разрядную IEEE-арифметику с плавающей точкой.
2) Разнообразие селекторов для различных областей памяти: code, data, bdata, idata, xdata и pdata, которые позволяют осуществить эффективный доступ к различным областям памяти и сгенерировать компактный код.

В таблице 2.1 содержатся селекторы для различных областей памяти.

Таблица 2.1 - Селекторы для различных областей памяти

Селектор	Область памяти
data	128 байт во встроенном ОЗУ — непосредственная байтовая адресация

Продолжение таблицы 2.1

Селектор	Область памяти
bit	128 бит во встроенном ОЗУ — непосредственная битовая адресация
bdata	128 бит/16 байт во встроенном ОЗУ — непосредственная битовая/байтовая адресация
idata	256 байт во встроенном ОЗУ — непосредственная адресация
pdata	256 байт в страничной внешней памяти XDATA
xdata	64 Кбайт памяти XDATA
code	64 байт памяти CODE

3) Область размещения переменных и функций и соответственно время доступа к переменной в памяти определяют модели памяти. Выбор модели зависит от размера области памяти: Small -128 байт, Compact - 256 байт, Large - 64 Кбайт.
4) Разнообразие общих и специальных указателей данных для различных областей памяти:

а) основные указатели (generic pointers) позволяют получить доступ ко всем областям памяти 8051 (code, data, idata, xdata и pdata). Эти указатели используются в функциях типа memcpy и printf для обеспечения максимальной совместимости;

б) специальные указатели (memory-specific pointers) указывают на определенную область памяти 8051 и позволяют осуществить эффективный доступ к различным областям и сгенерировать компактный код. Например, область data занимает только 128 байт и указатель data размещается в одном байте. При использовании модели Small с селектором data программа исполняется быстрее и генерируемый код короче, однако для редко используемых переменных вполне подходит модель Large с селектором xdata. Сочетание моделей достигается путем явного указания селектора при описании переменной.

5) Наиболее эффективное управление прерываниями при написании функций прерываний на С.
6) Прямое управление банками регистров и полное их использование, битовая адресация данных. Для доступа к регистрам специального назначения и их отдельным битам используются ключевые слова sfr, sfr16 и sbit. Доступ к отдельным битам побитноадресуемых регистров осуществляется с помощью ключевого слова bit.
7) Доступ на С к регистрам специальных функций для всех МК 8051, простой интерфейс с ассемблером. Полная отладочная информация для последующей высокоуровневой отладки с помощью эмулятора.
8) Использование эмуляторов AJMP и ACALL.

9) Встроенный интерфейс для операционной системы реального времени RTX51.
10) Сложная синтаксическая проверка и подробные предупреждающие со общения.
11) Более сотни библиотечных функций ANSI C, учитывающих особенности различных членов семейства 8051.
12) Эффективность и скорость С-программы за счет использования ряда методов оптимизации: регистровой оптимизации, общей оптимизации кода и специальной оптимизации для 8051.
13) В поставку C51 входят 6 оптимизированных библиотек, представляющих пользователю свыше 100 библиотечных функций, помимо стандартного набора из ANSI C. Эти функции приведены в таблице 2.2.

Таблица 2.2 – Библиотечные функции C51

chkfloat_	atan2	isalnum	memccpy	sinh	strrchr
crol	atof	isalpha	memchr	sprint	strrpbrk
cror	atoi	iscntrl	memcmp	sqrt	strrpos
_getkey	atol	isdigit	memcpy	srand	strspn
irol	cabs	isgraph	memmove	sscanf	tan
iror	calloc	islower	memset	strcat	Tanh
irol	ceil	isprint	modf	strchr	toascii
iror	cos	ispunct	pow	strcmp	toint
nop	cosh	isspace	printf	strcpy	tolower
testbit	exp	isupper	putchar	strcspn	toupper
_tolower	fabs	isxdigit	puts	strlen	uUngetchar
_toupper	floor	labs	pand	strncat	va_arg
abs	free	log	realloc	strncmp	va_end
acos	getchar	log10	scanf	strncpy	va_star
asin	gets	longjmp	setjmp	strpbrk	vprintf
atan	inin_mempool	malloc	sin	strpos	vsprintf

Использование инструкций AJMP и ACALL. Встроенный интерфейс для операционной системы реального времени RTX51. Сложная синтаксическая проверка и подробные предупреждающие сообщения. Более сотни библиотечных функций ANSI C, учитывающих особенности различных членов семейства 8051. Эффективность и скорость С-программы за счет использования ряда методов оптимизации: регистровой оптимизации, общей оптимизации кода и специальной оптимизации для 8051.

Расширение языка С до С51. Компилятор С51 очень гибок. Он поддерживает стандарт ANSI и все аспекты программирования на С, описанные этим стандартом. Все дополнения к стандарту предназначены для поддержки контроллера 8051.

Дополнения касаются следующих понятий:

- типы данных;
- типы памяти;
- модели памяти;
- указатели;
- реентерабельные функции;
- функции обработки прерываний;
- операционные системы реального времени;
- поддержка PL/M и A51.

Типы данных. Компилятор С51 поддерживает все типы данных, приведенные ниже в таблице 2.3. Скалярные переменные могут быть объединены в структуры, объединения и массивы. За некоторым исключением (что указано ниже) доступ к значениям переменных может быть получен с помощью указателей.

Таблица 2.3 – Типы данных поддерживаемых компилятором С51

Типы данных	Биты	Байты	Область значений
bit*	1		от 0 до 1
signed char	8	1	от –128 до +127
unsigned char	8	1	от 0 до 255
enum	16	2	от –32768 до +3276
signed short	16	2	от –32768 до +3276
unsigned shor	16	2	от 0 до 65535
signed int	16	2	от –32768 до +32767
unsigned int	16	2	от 0 до 65535
signed long	32	4	от –2147483648 до 2147483647
unsigned long	32	4	от 0 до 4294967295
float	32	4	от ±1.175494E-38 до ±3.402823E+38
sbit*	1	0	от 0 до 1
sfr*	8	1	от 0 до 255
sfr16*	16	2	от 0 до 65535

Типы bit, sbit, sfr и sfr16 являются специфичными для процессора 8051 и компилятора С51. Они не описаны стандартом ANSI, поэтому к ним нельзя обращаться через указатели.

Типы данных sbit, sfr и sfr16 используются для обращения к специальным регистрам процессора 8051. Например, строка sfr P0 = 0x80; объявляет переменную P0 и назначает ей адрес специального регистра 0x80. Это адрес порта P0. Компилятор C51 производит автоматическое преобразование типов в случае, если результат относится к другому типу. Кроме того, можно выполнить принудительное приведение типов. В процессе приведения расширение знака к переменным знаковых типов добавляется автоматически.

Модификаторы памяти. Компилятор C51 поддерживает контроллеры типа 8051 и обеспечивает доступ ко всем областям памяти контроллера. Для каждой переменной можно точно указать область ее размещения в памяти. В таблице 2.4 содержатся модификаторы памяти.

Обращение к внутренней памяти данных происходит гораздо быстрее чем к внешней. Поэтому переменные, которые используются чаще других, следует размещать во внутренней памяти, а остальные – во внешней.

Таблица 2.4 – Модификаторы памяти

Модификаторы памяти	Описание
code	Память программ (64 Кб); выполняется следующий код: MOVC @A+DPTR.
data	Внутренняя память данных с прямой адресацией; самая быстрая работа с переменными (128 байт).
idata	Внутренняя память данных с косвенной адресацией; доступ ко всему адресному пространству (256 байт).
bdata	Бит-адресуемая внутренняя память данных; возможность обращаться как к битам, так и к байтам (16 байт).
xdata	Внешняя память данных (64 Kbytes); выполняется код: MOVX @DPTR.
pdata	Внешняя память данных со страничной организацией (256 байт); выполняется код: MOVX @Rn.

Применяя модификаторы памяти при объявлении переменной, можно указать, где именно она будет размещена. Если в объявлении переменной модификатор памяти не указан, выбирается модель памяти, установленная по умолчанию. Аргументы функции и переменные класса памяти auto, которые не могут быть размещены в регистрах, также хранятся в области памяти, установленной по умолчанию.

Модель памяти, выбираемая в качестве модели по умолчанию, устанавливается с помощью директив компилятора SMALL, COMPACT и LARGE.

Модели памяти. Вы можете назначить модель памяти для тех аргументов функций, автопеременных и переменных, в объявлении которых не указана соответствующая модель. Это делается с помощью директив SMALL, COMPACT и LARGE. Явное задание модели отменяет использование модели памяти по умолчанию. В таблице 2.5 даны названия и описание моделей памяти.

Почти всегда следует использовать модель памяти SMALL: в этом случае вы получите самый быстрый, компактный и наиболее эффективный код. Однако у вас есть возможность явно указать нужную модель памяти. Модели, использующие внешнюю память, стоит использовать только в том случае, если ваше приложение никаким образом не может работать при модели памяти SMALL.

Таблица 2.5 – Модели памяти

Название	Описание
SMALL	В этой модели все переменные по умолчанию размещаются во внутренней памяти данных. Достигается тот же эффект, как при явном использовании модификатора data. При такой модели обращение к переменным оказывается очень эффективным. Но все объекты данных и стек должны размещаться во внутренней памяти. Размер стека имеет решающее значение, так как пространство, занимаемое стеком, зависит от глубины вложенности различных функций. Обычно, если компоновщик BL51 сконфигурирован для оверлейной загрузки переменных во внутреннюю память данных, лучше всего использовать малую модель памяти (small).
COMPACT	Если используется компактная модель памяти, все переменные по умолчанию загружаются во внешнюю память данных – точно так же, как при явном задании модификатора памяти pdata. В этой памяти может быть размещено до256 байтов. Ограничения появляются вследствие использования косвенной адресации, когда обращение происходит через регистры R0 и R1. Данная модель памяти не так эффективна, как малая, и обращение к переменным происходит медленнее. Но компактная модель все же быстрее, чем большая. Старший бит адреса обычно устанавливается через порт 2, причем делается это вручную программистом.

Продолжение таблицы 2.5

Название	Описание
LARGE	При использовании большой модели памяти все переменные по умолчанию размещаются во внешней памяти данных. Можно также явно указать модель памяти с помощью модификатора xdata. Для адресации используется указатель DPTR. Обращение к памяти через этот указатель не является эффективным, особенно если переменная имеет длину 2 или более байт. При использовании данной модели памяти код получается длиннее, чем при малой или компактной модели.

Функции – обработчики прерываний. Компилятор C51 предоставляет возможность вызова функций при возникновении прерываний. Это дает возможность написания на языке С собственных обработчиков прерываний. Однако следует соблюдать осторожность в выборе номера прерывания и банка регистров. Компилятор автоматически генерирует вектор прерывания, а также код входа в обработчик и выхода из него. Атрибут interrupt, включенный в объявление функции, указывает на то, что данная функция обрабатывает прерывание. Кроме того, можно указать банк регистров для данного прерывания с помощью атрибута using.

2.3 Инструментальная система для Win 9x/NT

Инструментальная система T167B призвана решать следующие задачи:

- преобразование HEX- и BIN- файлов;
- передача загрузочных модулей различных форматов в целевую систему с протоколами разного уровня сложности;
- получение информации из целевой системы;
- обеспечение элементарных операций с последовательным каналом (прием и передача байта, эмуляция терминала, настройка скорости);
- обеспечение быстрой адаптации к целевой системе.

Управляющие клавиши. CTRL+BREAK - аварийный выход; в большинстве случаев приводит к корректному завершению работы T167B при зависаниях. Alt+X - выход в DOS. Up, Down - перелистывание команд; командная строка в T167B имеет историю, записываемую в файл.

Инструментальная система T2 для Windows 9x/NT с точки зрения команд для работы с SDK-1. является аналогом системы T167B для MS-DOS (за исключением нескольких команд). В таблице 2.6 содержится перечень основных команд системы T2.

Таблица 2.6– Перечень основных команд системы Т2

Команда	Действие
OPENCHANNEL	Открытие COM-порта с установкой сигнала RTS.
TERM	Включение эмулятора терминала.
CLOSEHANNEL	Закрытие COM-порта.
OPENCOM1, OPENCOM2	Открытие портов COM1 или COM2 с установкой сигнала RTS.
+ECHO	Включение режима копирования консольного вывода в файл echo.txt.
-ECHO	Выключение режима копирования консольного вывода в файл echo.txt
LOADHEX	HEX-загрузка.
ADDHEXSTART	Добавление стартового адреса в конец HEX-файла.
BYE	Выход из Т2.
LFILE	Интерпретация командного файла.

OPENCHANNEL (baud -> com)

Открытие последовательного порта на заданной скорости. Числовой параметр baud определяет скорость в бодах, например, 19200. Параметр com может иметь два значения: «coml» или «сот2».

Пример:

9600 openchannel coml

OPENCOM1, OPENCOM2 (->)

Открытие COM1 или COM2 на скорости 9600 бод.

Пример:

opencoml

CLOSECHANNEL (->)

Закрытие ранее открытого COM-порта.

Пример:

closechannel

TERM (w->)

Включение эмулятора терминала:

0 - бинарный;

1 - HEX;

Пример:

1 term

+ECHO (->)

-ECHO (->)

Включение/выключение режима копирования консольного вывода в файл echo.txt.
Пример:
+echo

LOADHEX+ (->) filename.hex

Загрузка HEX-файла в целевую систему (SDK-1.1) по протоколу загрузчика HEX202. Этот протокол предполагает последовательную пересылку строк из HEX-файла filename.hex. После посылки очередной строки ожидается подтверждение со стороны HEX202 в виде символа '+' или запрос на повторную посылку в виде '-'. Далее ожидается символ '.'и производится либо посылка следующей строки (если были приняты '+' и '.'), либо повторная посылка данной строки. Если снова были приняты '-' и '.', то попытка повторяется во второй раз и т.д. - всего 9 раз, после чего производится аварийный выход.

Посылка HEX-файла производится только при наличии периодической индикации работоспособности HEX202, проявляющейся в посылке символа '.'.
Необходимо заметить, что перед посылкой HEX-файла, сгенерированного в какой-либо среде разработки (например, в ц Vision), необходимо добавить в его конец стартовый адрес (то есть адрес в памяти RAM, на который передается управление после загрузки в SDK-1.1) командой addhexstart.
Пример:
loadhex+ myfile.hex

ADDHEXSTART (addr,seg->) filename.hex

Добавление в конец файла filename.hex строчки, которая нужна для передачи управления загрузчиком HEX202 по адресу addr после загрузки в целевую систему (SDK-1.1). Поле seg необходимо указывать, но в данный момент оно не используется.
Пример:
0x9000 0x0 addhexstart myfile.hex

BYE (->)

Выход из T167B.
Пример:
bye

LFILE (->) filename.ext

Интерпретация командного файла filename.ext. Файл представляет собой набор строк текста, содержащих команды T167B в том же виде, в котором они представлены в командной строке T167B.
Пример:
lfile myfile.ini

2.4 Резидентный загрузчик HEX202

Резидентный загрузчик HEX202. Резидентный загрузчик HEX202 располагается во Flash-памяти ADuC812 начиная с адреса 0l00h. Он обеспечивает начальную инициализацию системы, загрузку программ в HEX-формате в память SDK-1.1 и передачу им управления.

Начальная инициализация. При включении питания или передаче управления на ячейку с адресом 0 происходит повторная инициализация всех регистров специального назначения их значениями по умолчанию. Это сделано для того, чтобы при случайной передаче управления на ячейку с адресом 0 вследствие возможной ошибки в пользовательской программе не происходило сбоя системы, а сама система вела себя так же, как при включении питания. Эта же процедура повторяется непосредственно перед передачей управления загруженной программе. В случае успешной инициализации на ЖКИ на мгновение выводится надпись «SDK-1.1, 2001 ©LMT Ltd» и на резонатор выдается короткий сигнал.

Загрузка программ в память SDK-1.1. После процедуры инициализации системы последовательный канал настраивается в режим 9600 бит/сек, 8 бит данных, 1 стоп-бит, без контроля четности и в него выдается строчка «HEX202-XX», где XX - номер версии загрузчика. Далее с интервалом примерно в 200 мс выдается символ '.' и ожидается появление символа со стороны инструментальной системы на PC. При появлении символа, если это первый символ строки в HEX-формате, то есть двоеточие (':'), выдача символа '.' прекращается и производится прием остальной части HEX-строки. После завершения приема очередной HEX-строки вычисляется ее контрольная сумма. Если она не совпадает с принятой, то в последовательный канал выдается символ '-', сигнализирующий об ошибке приема. В противном случае выдается '+' и принятая строка обрабатывается в соответствии с указанной в ней командой (запись данных в память, конец блока или передача управления). Далее, если не было команды передачи управления, вывод в последовательный канал символа '.' возобновляется и ожидается следующая HEX-строка.

Передача управления загруженной программе. Передача управления происходит по приему HEX-строки вида :02AAAA060000SS, где AAAA - это HEX-адрес, по которому необходимо передать управление, SS - контрольная сумма HEX-строки, <cr> -символ возврата каретки. Такая строчка должна быть добавлена в конец каждого HEX-файла, загружаемого в SDK-1.1. Для этого в поставляемых с SDK-1.1 инструментальных системах есть команда addhexstart.

Важное замечание. В процессе своей работы HEX202 использует область статической памяти в диапазоне адресов F000h-FFFFh, что следует учитывать при написании программ для стенда. Расположение части кода или статических

данных по этим адресам может привести к некорректной работе загрузчика и неработоспособности загружаемой программы.

Формат стандартного Intel Hex файла. Эта часть описывает формат стандартного Intel Hex файла. Шестнадцатеричный формат Intel или формат Intel Hex является стандартом для запоминающих машин в отображаемом и печатаемом форматом.

Стандартный Intel Hex формат генерируется ассемблером MicroConvertor (8051-совместимый код). Этот ассемблер (Metalink 2-pass) доступен как часть средств разработки для QuickStart или как свободно распространяемая копия, доступная на веб-сайте www.analog.com/microconverter. Intel Hex файл это последовательность строк или «шестнадцатеричных записей», содержащих следующие поля: Таблица 2.7 содержит формат записи Intel Hex.

Таблица 2.7 – Формат записи Intel Hex

Поле	Число байт	Описание
Начало записи	1	«:», характеризует начало записи
Длина записи	2	число байт в записи
Адрес загрузки	4	начальный адрес для размещения байтов данных
Тип записи	2	00 = данные, 01 = завершение
Байты данных	0-16	Данные
Контрольная сумма	2	сумма байтов данных в записи + контрольная сумма = 0

Эти поля показаны в раскрытом виде на рисунке 2.10, который иллюстрирует пример содержимого стандартного Intel Hex файла.

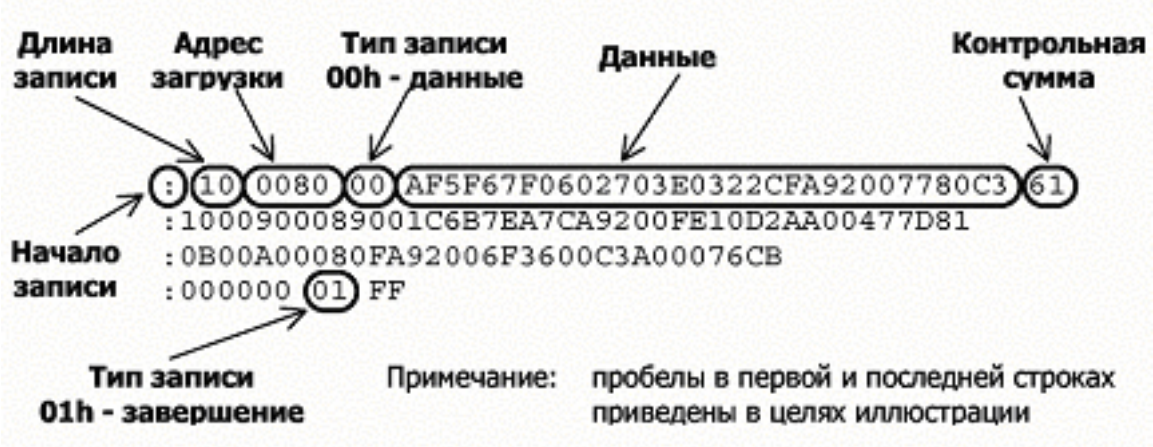


Рисунок 2.10 – Формат Intel Hex

3 Система ввода-вывода

Ввод-вывод данных в стенде SDK1.1 может осуществляться как непосредственно через порты процессора так и через консоль опосредованно ПЛИС микросхемой MAX8064.

3. 1 Программирование MAX8064

Стенд SDK1.1 имеет четыре устройства ввода-вывода, объединённых в «консоль управления». В состав консоли входят:

- 8 светодиодов;
- матричная клавиатура;
- динамик;
- LCD дисплей (16x2 символов).

Связь между консолью и процессором опосредована микросхемой MAX8064. Микросхема MAX имеет восемь однобайтных регистров, с помощью которых происходит управление вводом-выводом. Данные регистры отображаются во внешнем адресном пространстве микроконтроллера как интервал адресов от 0x080000-0x80007, то есть находятся в начале восьмой страницы расширенной памяти. Ниже, в таблице 3.1, дана информации об адресах размещения регистров MAX8064.

Таблица 3.1 - Размещение регистров MAX8064

Адрес в странице	Регистр	Доступ	Описание
0x0	KB	Чтение	Регистр клавиатуры
0x1	DATA_IND		Регистр шины данных LCD
0x2	EXT_LO	Запись	Регистр параллельного порта (младший байт)
0x3	EXT_HI		Регистр параллельного порта (старший байт)
0x4	ENA		Регистр разрешений
0x6	C_IND		Регистр управления LCD
0x7	SV		Регистр состояния светодиодов

Для записи в регистр необходимо создать указатель на байт во внешней памяти с помощью использования селектора xdata и указать в нём адрес регистра.

Пример:

```
unsigned char xdata ucxMaxReg = 0x7;
```

В данном примере создаётся указатель на регистр управления светодиодами. Перед записью по этому указателю, т.е. перед разыменованием необходимо переключить страницу внешней памяти на 8-ую, а после записи вернуться к исходной. Номер используемой страницы памяти находится в регистре DPP. Если модуль “ADuC812.H” не подключен, то переменную регистра можно определить вручную:

```
sfr   DPP = 0x84;
unsigned char oldPage = DPP;
DPP = 0x08;
*ucxMaxReg = 0xff;
DPP = oldDPP;
```

Данный пример записывает в регистр управления светодиодами.

3.2 Светодиоды

Самым простым средством вывода информации на стенде являются светодиоды. Регистр светодиодов представляет собой байт, в котором каждый бит отвечает за работу одного светодиода (младший разряд за нижний светодиод).

Нижеприведенный пример зажигает первый, третий и восьмой светодиоды:

```
unsigned char xdata *ucxMaxReg = 0x7;
sfr DPP = 0x84;
unsigned char oldPage = DPP;
DPP = 0x08;
*ucxMaxReg = (1 ) | (1 << 2) | (1<<7);
DPP = oldPage;
```

3.3 Динамик

Вывод звука производится путем подачи на динамик периодических сигналов. Сигнал формируется на основе содержимого регистра ENA. Ниже в таблице 3.2 содержится информация о регистре разрешений динамика.

Для генерации звука используются биты SND2..0, отвечающие за уровень напряжения на обмотке динамика, задавая различные уровни напряжения можно генерировать звуки разной громкости. Однако, некоторые стенды не поддерживают изменение уровня напряжения. На таких стендах сигнал формируется на основе бита SND2.

Алгоритм формирования звуковой волны:

– Подача сигнала на динамик.

– Пауза равная полупериоду волны.
– Отключение сигнала (изменения напряжения).
– Пауза равная полупериоду волны

Пример:

```
void Snd_Enable(unsigned char Value)
{
  unsigned char oldDPP=DPP;
  unsigned char xdata *reg = 0x4;
  unsigned char EReg=0;

  DPP = 0x8;
  if(Value)EReg = 0x10;
  else EReg     = 0x0;
  *reg = EReg;
  DPP=oldDPP;
  return;
}
void main ()
{
while(1)
{
Snd_Enable(1);
for(j = 0; j < 300; j++ );
Snd_Enable(0);
for(j = 0; j <  300; j++);
}
}
```

Таблица 3.2 – Регистр разрешений динамика

Номер бита	Название	Доступ	Описание
0x7	-	Запись	
0x6	-		
0x5	INT0		Бит прерывания
0x4	SND2		
0x3	SND1		Уровень напряжения на динамике
0x2	SND0		
0x1	EN_HI		Чтение/Запись из параллельного порта
0x0	EN_LO		

3.4 Матричная клавиатура

Ввод данных непосредственно от пользователя осуществляется с помощью матричной клавиатуры. Клавиатура представлена в виде матрицы 4x4, а доступ к колонкам и рядам клавиатуры осуществляется путём чтения и записи управляющего байта, располагающегося в регистре KB (0x0 offset) микросхемы MAX. Информация о регистре клавиатуры содержится в таблице 3.3.

Таблица 3.3 – Регистр клавиатуры

Бит	Поле	Доступ	Описание
0..3	Сканируемая колонка	Чтение/Запись	Указывает клавиатуре результат сканирования какой колонки необходимо поместить в старшие биты
4..7	Состояние колонки	Запись	Возвращает состояние указанной колонки: клавиша нажата клавиша не нажата

При указании более одного нуля приоритет имеет тот, который находится в младшем разряде. После указания номера колонки в старшие биты подается состояние ряда в данной колонке. При нажатии какой либо клавиши бит состояния устанавливается в «0». Ниже, в таблице 3.4, показана архитектура клавиатуры.

Сканирование происходит путем записи в младшие 4 бита номера сканируемой колонки, номер определяется логическим «0» установленным в соответствующем разряде: первая колонка - нулевой бит.

Таблица 3.4 - Архитектура клавиатуры

1	2	3	A	Ряд 1
4	5	6	B	Ряд 2
7	8	9	C	Ряд 3
*	0	#	D	Ряд 4
Колонка 1	Колонка 2	Колонка 3	Колонка 4	

Пример:

Байт в регистре		Результат
Старшие биты	Младшие биты	
1111	0111	Сканирование четвёртой колонки. Ни одна клавиша не нажата
1011	1011	Сканирование третьей колонки. Нажата клавиша во втором ряду- «9»
1001	1100	Сканирование первой колонки. Нажаты «7» и «4»

Во избежание ложного срабатывания при обнаружении нажатой клавиши рекомендуемся сделать задержку порядка 50 мс и повторить сканирование. Это поможет избежать ложных срабатываний из-за «дребезга» контактов и случайных «шумовых» помех.

3.5 LCD-дисплей

Для вывода текстовой информации в стенде SDK1.1 используется жидкокристаллический дисплей WH1602B-YGK-CP вместимостью 32 символа (2 строки по 16 символов). Модуль LCD (далее БИС) подключен к контроллеру, который имеет три регистра: регистр команд (IR), регистр данных (DR) и счетчик адреса (AC). Регистр IR хранит коды операций, и доступен только для записи, при чтении возвращается только один бит BF – флаг занятости, регистр DR предназначен для записи и чтения из DDRAM (память отображаемых символов) и CGRAM (память генератора символов). Запись и чтение из памяти происходит по адресу указанному в AC. В таблицах 3.5 и 3.6 дано описание регистра DATA_IND и C_IND.

Запись в регистры осуществляется через регистры MAX.

Таблица 3.5 – Регистр DATA_IND

Бит	Доступ	Описание
0..7	чтение/запись	Данные отправляемые БИС

Таблица 3.6 – Регистр C_IND

Бит	Название флага	Доступ	Описание
0	E	Запись	Бит управляет синхроимпульсом. При подаче «1» в этот бит происходит захват данных на шине БИС (соединена с DATA_IND) и выполнение команды.
1	RW		Переключатель чтение запись: 1- Чтение 0- Запись
2	RS		Переключатель команда/данные 1 – Данные 0 – Команда
3	Reserved		В некоторых экземплярах используется для питания и других тех. целей. Для корректной работы необходимо этот бит всегда устанавливать в «1»
4..7	Not used	-	-

Выполнение команд осуществляется подачей флага E в регистре C_IND. Перед подачей которого необходимо установить данные в регистр DATA_IND и флаги RW и RS. После подачи разрешающего сигнала БИС начинает выполнение команды, и до тех пор, пока команда не будет выполнена БИС будет игнорировать все запросы, поэтому необходимо либо обеспечивать необходимые задержки, либо проверять флаг занятости перед тем как отправить следующую команду. В таблице 3.7 представлены комбинации флагов RS и RW.

Контроллер обладает собственной памятью на 80 байт, представленной в виде строк длиной 16 символов. Для адресации внутренней памяти контроллер использует 7-ми разрядные адреса. В таблице 3.8 показана адресация памяти.

Таблица 3.7 – Комбинации флагов RS и RW

Комбинация		Описание
RS	RW	
0	0	Запись из DATA_IND в регистр команды IR и её выполнение
0	1	Чтение флага занятости. Если БИС занята то новые команды не принимаются
1	0	Запись в память DDRAM или CGRAM

1	1	Чтение в DATA_IND из DDRAM или CGRAM

Таблица 3.8 – Адресация памяти

Старшие биты (строка)			Младшие биты (символ в строке)			
A6	A5	A4	A3	A2	A1	A0

Для отображения на экране используются две строки (так называемая DDRAM):

- нулевая: верхняя строка на дисплее;
- четвёртая: нижняя строка на дисплее.

Адреса символов на экране представлены в таблице 3.9.

Таблица 3.9 - Адреса символов на экране

0x00	0x01	0x02	0x03	…	0x0E	0x0F
0x40	0x41	0x42	0x43	…	0x4E	0x4F

3.5.1 Инициализация LCD

Перед использованием дисплея его сначала нужно инициализировать в соответствии с протоколом указанным производителем. Протокол инициализации LCD показан на рисунке 3.1.

Текст программы для инициализации дисплея:

```
void SetLcdData(unsigned char ucNewData)  // помещает байт в регистр данных LCD
{
        unsigned char xdata * ucxLcdData = 0x01;
        unsigned char ucLastDPP = DPP;

        DPP = 0x08;
        *ucxLcdData   = ucNewData;
        DPP = ucLastDPP;
}
void LcdBisEnable(char cReadWrite /*1-read 0-write*/, char cDataCode /*1-data 0-command*/)
{
        unsigned char ucStrobe = 0;
        unsigned char xdata * ucxLcdData = 0x06;
        unsigned char ucLastDPP = DPP;
        int i = 0;
        DPP = 0x08;
        ucStrobe |= 0x9; // сигнал E = 1  и Reserved = 1;
        if( cReadWrite)ucStrobe |= 0x2; // бит RW
        if( cDataCode) ucStrobe |= 0x4; // бит RS
        *ucxLcdData   = ucStrobe;  // запись в регистр
```

```
	for( i = 0; i < 10; i++)continue;
	*ucxLcdData   = ucStrobe & 0xFE; // обнуление разрешающего сигнала E
	DPP = ucLastDPP;
	for( i = 0; i < 300; i++) continue; // задержка на исполнение команды
}
void LcdInit()
{
	int i = 0;
	for(i = 0; i < 4000; i++)continue; // задержка для ожидания конца процесса перехвата

	SetLcdData(0x30);    // 00110000 – установка битности дисплея 8bit
	LcdBisEnable(0x0,0x0);  //
	for(i = 0; i < 1500; i++)continue; // задержка по протоколу
	SetLcdData(0x30);    // 00110000 – установка битности дисплея 8bit
	LcdBisEnable(0x0,0x0);  //
	for(i = 0; i < 50; i++)continue; // задержка по протоколу
	SetLcdData(0x30);    // 00110000 – установка битности дисплея 8bit
	LcdBisEnable(0x0,0x0);  //
	for(i = 0; i < 150; i++)continue; // задержка по протоколу
	SetLcdData(0x38);    // 00111000 – установка битности дисплея
	LcdBisEnable(0x0,0x0);
	for(i = 0; i < 150; i++)continue; // задержка по протоколу
	SetLcdData(0x08);    // 00001000 – отключение дисплея
	LcdBisEnable(0x0,0x0);
	for(i = 0; i < 150; i++)continue; // задержка по протоколу
	SetLcdData(0x01);    // 00000001 – отключение дисплея
	LcdBisEnable(0x0,0x0);
	for(i = 0; i < 1500; i++)continue; // задержка по протоколу

	SetLcdData(0x06);      // 00000110 – начальные установки – направление текста  лев-
>прав
	LcdBisEnable(0x0,0x0);
	for(i = 0; i < 150; i++)continue; // задержка по протоколу

	SetLcdData(0x0c);    // 00001100 – включение дисплея
	LcdBisEnable(0x0,0x0);
	for(i = 0; i < 150; i++)continue; // задержка по протоколу
}
```

Рисунок 3.1 - Протокол инициализации LCD

3.6 Шина I2C

Для связи с внешними устройствами ввода-вывода процессор ADuC812BS может использовать два дискретных выхода, которые совместимы с интерфейсом I2C. Этот интерфейс представляет собой двунаправленную шину, состоящую из двух линий: линии синхронизации (SCL) и линии передачи данных (SDL). Схема шины данных I2C показана на рисунке 3.2.

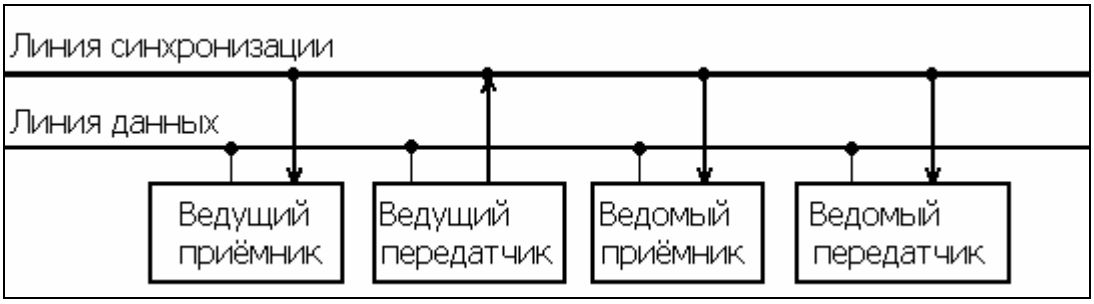


Рисунок 3.2 - Схема шины I2C

3.6.1 Управляющие сигналы

При обмене данными между двумя устройствами, устройство контролирующее процесс передачи называется ведущим (master), а второе устройство называется ведомым (slave). Для передачи бита данных ведущее устройство устанавливает «1» на линии синхронизации, если при этом уровень сигнала на линии данных не изменяется то происходит передача бита данных, а любое изменение сигнала воспринимается как контрольный сигнал. Управление шиной I2C осуществляется с помощью регистра I2CCON. Отдельные биты данного регистра могут быть адресованы как битовые регистры. В таблице 3.10 содержатся сведения о регистрах управления I2C.

Таблица 3.10 - Регистры управления шиной I2C

Бит	Битовый регистр	Описание
0x7	MDO	Выходной сигнал
0x6	MDE	Переключатель приёмник-передатчик.
0x5	MDC	Бит синхронизации, управляет линией SCL
0x4	MDI	Входной сигнал
0x3	I2CM	Переключатель ведущий-ведомый
0x2	I2CRS	Сброс последовательного порта
0x1	I2CTX	Состояние направления передачи
0x0	I2CI	Прерывание последовательного интерфейса

4 Система прерываний

Для обработки некоторых медленно работающих устройств можно использовать, аппаратные прерывания. Процессор ADuC812BS установленный на стенде обеспечивают восемь источников прерывания и два уровня приоритета. Как и во всех процессорах семейства Intel, таблица векторов прерываний расположена в начале памяти программ, однако, на стенде первые восемь килобайт адресного пространства отведены под сервисные программы и находятся под защитой от записи, и доступ к ним возможен только в режиме перепрограммирования FLASH-памяти. Для решения этой проблемы в начале памяти были размещены команды перехода на адрес равный 0x2000 + адрес данного вектора, то есть вектора программным путём отображаются на интервал памяти 0x2003-0x2043 названный пользовательской таблицей векторов. Именно поэтому рекомендуется записывать программы начиная с адреса 0x2100. В таблице 4.1 приведен список прерываний ADuC812.

Таблица 4.1 - Прерывания ADuC812

Прерывание	Наименование	Адрес вектора	Приоритет
PMSI	Источник питания ADuC812.	43H	1
IE0	Внешнее прерывание INT0.	03H	2
ADCI	Конец преобразования АЦП.	33H	3
TF0	Переполнение таймера 0.	0BH	4
IE1	Внешнее прерывание INT1.	13H	5
TF1	Переполнение таймера 1.	1BH	6
I2CI/ISPI	Прерывание последовательного интерфейса (I^2C, SPI).	3BH	7
R1/T1	Прерывание UART.	23H	8
TF2/EXF2	Переполнение таймера 2.	2BH	9

Ниже, на рисунке 4.1 проиллюстрировано использование прерываний в SDK-1.1.

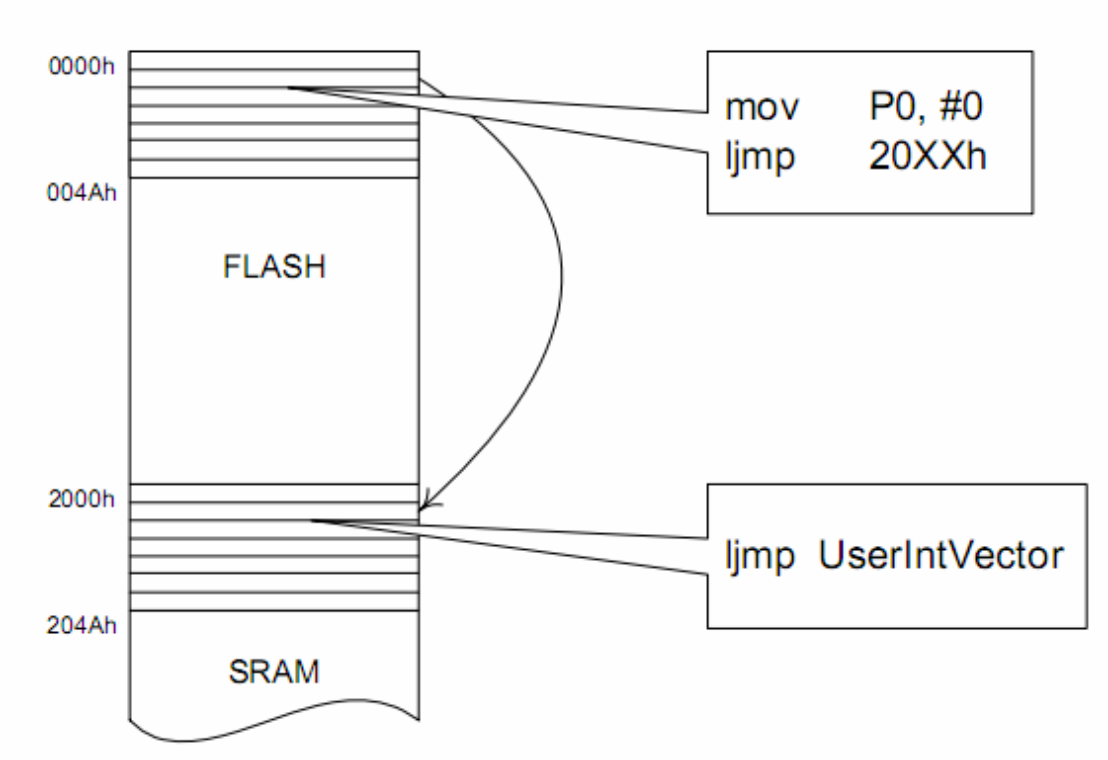


Рисунок 4.1 – Использование прерываний в SDK-1.1

Приведем пример помещения собственного вектора в пользовательскую таблицу. Пусть требуется осуществить обработку прерываний от таймера 0 (прерывание 0Bh). В программу на языке Си (компилятор фирмы Keil Software) можно вставить следующий код:

```
void TO_ISR(void) interrupt 1 // Обработчик прерывания от таймера 0
{
// Действия, выполняемые обработчиком
}
void SetVector(unsigned char xdata * Address, void * Vector)
// Функция, устанавливающая вектор прерывания Vector по адресу Address
{
unsigned short xdata * TmpVector;  // Временная переменная
*Address = 0x02;   // Первым байтом по указанному адресу записывается код команды
// передачи управления ljmp, равный 02h
TmpVector = (unsigned short xdata *) (Address+1) ;
*TmpVector = (unsigned short) Vector; // Далее записывается адрес перехода Vector
// Таким образом, по адресу Address теперь располагается инструкция ljmp Vector
}
void main(void)
{
/*...*/
SetVector(0x200B, (void *) TO_ISR); // Установка вектора в пользовательской таблице
ET0=1; EA=1;          // Разрешение прерываний от таймера 0
/*...*/
}
```

В блоке регистров специальных функций есть два регистра, предназначенных для управления режимом прерываний IE и уровнями приоритета IP. Возможность программной установки/сброса любого управляющего бита в этих двух регистрах делает систему прерываний 8051 исключительно гибкой. В таблицах 4.2 и 4.3 описаны назначения битов регистров IE и IP.

Таблица 4.2 – Регистр масок прерывания (IE)

Символ	Позиция	Имя и назначение
1	2	3
EA	IE.7	Снятие блокировки прерывания. Сбрасывается, программно для запрета всех прерываний независимо от состояний IE.4 - IE.0
	IE.6	Не используется
	lE.5	Не используется
ES	IE.4	Бит разрешения прерывания, от приемопередатчика Установка/сброс программой для разрешения/запрета прерываний от флагов TI или RI .
ET1	IE.3	Бит разрешения прерывания от таймера. Установка/сброс программой для разрешения/запрета прерываний от таймера 1
EX1	IE.2	Бит разрешения внешнего прерывания 1. Установка/сброс программой для разрешения/запрета прерывания 1
ET0	IE.1	Бит разрешения прерывания от таймера 0. Установка/сброс программой для разрешения/запрета прерываний от таймера 0
EX0	IE.0	Бит разрешения внешнего прерывания 0. Установка/сброс программой для разрешения/запрета прерывания 0

Таблица 4.3 – Регистр приоритетов прерываний (IP)

Символ	Позиция	Имя и назначение
-	IP.7 - IP.5	Не используется
PS	IP.4	Бит приоритета приемопередатчика. Установка/сброс программой для присваивания прерыванию от приемопередатчика высшего/низшего приоритета.
PT1	IP.3	Бит приоритета таймера 1. Установка/сброс программой для присваивания прерыванию от таймера 1 высшего/низшего приоритета.

Продолжение таблицы 4.3

Символ	Позиция	Имя и назначение
PX1	IP.2	Бит приоритета внешнего прерывания 1. Установка/сброс программой для присваивания высшего/низшего приоритета внешнему прерыванию INT1.
PT0	IP.1	Бит приоритета таймера 0. Установка/сброс программой для присваивания прерыванию от таймера 0 высшего/низшего приоритета.
PX0	IP.0	Бит приоритета внешнего прерывания 0. Установка/сброс программой для присваивания высшего/низшего приоритета внешнему прерыванию INT0.

Система прерываний формирует аппаратный вызов (LCALL) соответствующей подпрограммы обслуживания, если она не заблокирована одним из следующих условий:

- в данный момент обслуживается запрос прерывания равного или высокого уровня приоритета;
- текущий машинный цикл - не последний в цикле выполняемой команды;
- выполняется команда RETI или любая команда, связанная с обращением к регистрам IE или IP.

Отметим, что если флаг прерывания был установлен, но по одному из указанных выше условий не получил обслуживания и к моменту окончания блокировки уже сброшен, то запрос прерывания теряется и нигде не запоминается.

По аппаратно сформированному коду LCALL система прерывания помещает в стек только содержимое счетчика команд (PC) и загружает в него адрес вектора соответствующей подпрограммы обслуживания. По адресу вектора должна быть расположена команда безусловной передачи управления (JMP) к начальному адресу подпрограммы обслуживания прерывания. В случае необходимости она должна начинаться командами записи в стек (PUSH) слова состояния программы (PSW), аккумулятора, расширителя, указателя данных и т.д. и должна заканчиваться командами восстановления из стека (POP). Подпрограммы обслуживания прерывания должны завершаться командой RETI, по которой в счетчик команд перезагружается из стека сохраненный адрес возврата в основную программу. Команда RET также возвращает управление прерванной основной программе, но при этом не снимут блокировку прерываний, что приводит к необходимости иметь программный механизм анализа окончания процедуры обслуживания данного прерывания.

Существует два способа перехвата прерываний:

1) писать программу с адреса перехватываемого прерывания. Однако, данный подход означает что можно перехватить только одно прерывание;

2) запись команды безусловного перехода на функцию-обработчик в пользовательскую таблицу векторов.

После того как прерывание было перехвачено необходимо разрешить процессору вызывать эти прерывания. Список разрешенных прерываний, находящихся в регистре IE, приведен в таблице 4.4.

Таблица 4.4 – Список разрешенных прерываний

IE		
Бит	Битовый регистр	Описание
0x7	EA	Все прерывания
0x6	EADC	Прерывания от ЦАП.
0x5	ET2	Прерывание второго таймера
0x4	ES	Прерывание от последовательного порта
0x3	ET1	Прерывание первого таймера
0x2	EX1	Внешние прерывания 1
0x1	ET0	Прерывания нулевого таймера
0x0	EX0	Внешние прерывания

После того как прерывания будут разрешены, они будут обрабатываться назначенным обработчиком.

5 Многозадачность

Концепция ЭВМ с программным управлением подразумевает выполнение программы последовательно, шаг за шагом. Таким образом одно вычислительное ядро может выполнять только одну программу в один момент времени, однако часто требуется выполнение нескольких программ одновременно. До 80-х годов двадцатого века существовало два способа решения данной проблемы: либо наращивание количества вычислительных ядер, либо включение в основную программу нескольких функций и поочередное их выполнение. Каждый способ не лишён недостатков, наращивание числа вычислительных ядер приводило к большим финансовым затратам, а также к увеличению размеров ЭВМ, а они в то время и так были немаленькими. В случае последовательного выполнения все подпрограммы были зависимыми друг от друга, и в случае отказов одной из подпрограммы, например выхода в бесконечный цикл или фатальной ошибки, все остальные подпрограммы так же прекращали выполнение. В обоих случаях изменение количества одновременно выполняемых процессов было достаточно затруднено.

5.1 Исключающая многозадачность

В восьмидесятые годы была разработана так называемая концепция исключающей многозадачности. При таком построении основная программы (ядро ОС) оперировала с дескрипторами процессов, в которых описывалась среда выполнения каждого процесса. Под ресурсами в то время понимались процессорное время, область памяти и устройства ввода-вывода. Такие отдельно выполняемые программы получили название процесса, и стали независимыми друг от друга. При таком устройстве вычислительного процесса, добавление новых и завершение работающих процессов перестало представлять какие либо трудности. Необходимо было только добавить дескриптор процесса в очередь на использование ресурсов, либо удалить его из очереди.

Словесный алгоритм.

Загрузка ядра ОС.

Загрузка в память исполняемых процессов.

Создание дескрипторов.

Сортировка очереди дескрипторов.

Выполнение первого процесса в очереди.

Возврат в ядро (менеджер ресурсов).

Сохранение новой среды в дескриптор.

Перенос дескриптора в конец очереди.

Повтор пунктов, начиная с 5-ого.

5.2 Реализация многозадачности в среде uVision

В данном разделе будет описан способ реализации многозадачности на стенде SDK 1.1 с помощью перехвата прерывания таймера. Менеджер ресурсов будет рассчитан всего на два процесса, дабы не осложнять понимание самой концепции многозадачности.

Переменные с фиксированным адресом создаются с помощью директивы _AT_ Для простого менеджера ресурсов потребуется только три переменных: номер активного процесса, и программные счетчики каждого из процессов. А также одна внутренняя переменная – адрес возврата из прерывания. Их определение будет выглядеть так:

```
unsigned char  xdata  cThread    _at_ 0xff00;
unsigned short xdata ucCurAdr    _at_ 0xff10;
unsigned short xdata usRetAdr[2] _at_ 0xff12;
```

Данные переменные должны быть заполнены до того как будет разрешено выполнение прерывания. cThread необходимо приравнять к нулю как показано на примере *((unsigned char xdata*)(0x00ff00)) = 0x00;

Внутри менеджера ресурсов необходимо описать чередование активных процессов и сохранение программного счетчика для активного процесса. Также здесь необходимо установить таймер на необходимое время.

```
TH0 = 0xf0;
ET0 = 1;
if(cThread == 0)
{       usRetAdr[0] = ucCurAdr;
        cThread     = 0x1;
        ucCurAdr    = usRetAdr[1]; }
else
{       usRetAdr[1] = ucCurAdr;
        cThread = 0x0;
        ucCurAdr    = usRetAdr[0]; }
```

Данный код сначала сохраняет содержимое программного (ucCurAdr) счетчика в память и устанавливает новое значение для нового процесса.

Для сохранения значения программного счетчика в переменой ucCurAdr необходимо считать его из адреса возврата прерывания. Например, используя данный код Ассемблера:

```
#pragma asm
        MOV  DPTR,#0xff10
        mov A, SP;
        subb A, #0x5;  // глубина на которой находится младший байт адреса возврата
        MOV R1, A
```

```
        MOV A, @r1;
        MOVX @DPTR, A
        DEC R1;
        INC DPTR;
        MOV A, @R1;
        MOVX @DPTR, A
#pragma endasm
```

Для определения смещения адреса возврата относительно вершины стека необходимо откомпилировать проект и просмотреть сгенерированный SRC код и посчитать количество операций PUSH.

Для сохранения нового адреса возврата можно использовать такой код:

```
#pragma asm
MOV   DPTR,#ucCurAdr
mov A, SP;
subb A, #0x5;
MOV R1, A
MOVX A, @DPTR
MOV  @R1, A;
DEC R1;
INC DPTR;
MOVX A, @DPTR
MOV  @R1, A;
#pragma endasm
```

Регистр-указатель DPTR можно использовать только совместно с командой MOVX, обрабатывающей обращения к расширенной памяти.

После написания менеджера процессов необходимо указать стартовые адреса для процессов. Узнать их можно во время процесса отладки в окне дизассемблированного кода. На рисунке 5.1 показан процесс выбора окна дизассемблированного кода.

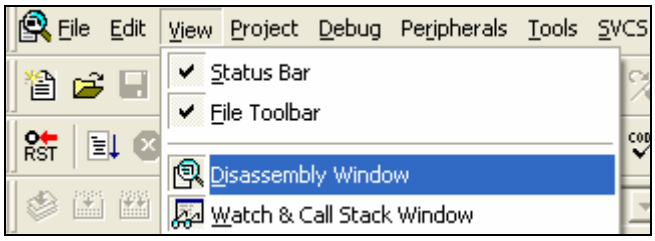


Рисунок 5.1 – Выбор команды дизассемблированного кода.

А записать в переменные, используемые менеджером ресурсов, с помощью обращения к ячейке памяти, записав адрес команды с которой начинается процесс: *((unsigned short xdata*)(0x00ff14)) = 0x21db;

Если все сделано правильно, то два процесса будут выполняться одновременно. Отдельные процессы лучше всего размещать в отдельных бесконечных циклах.

6 Лабораторный практикум по курсу «Организация ЭВМ и систем»

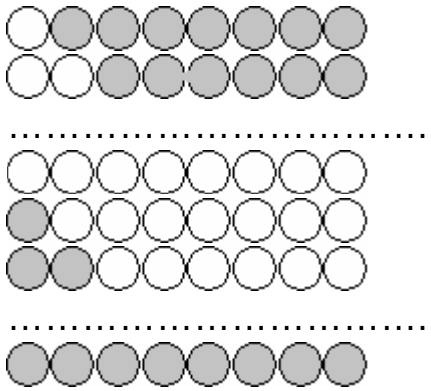
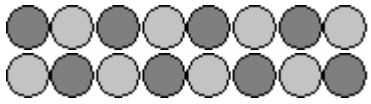
Архитектура учебного лабораторного комплекса SDK-1.1 построена в соответствии с основными принципами микро-ЭВМ. Выполнение лабораторных работ по курсу «Организация ЭВМ» с использованием SDK-1.1 позволит изучить теоретические особенности структуры и архитектуры ЭВМ, а также получить практические навыки создания и отладки программ.

6.1 Лабораторная работа №1. Светодиоды

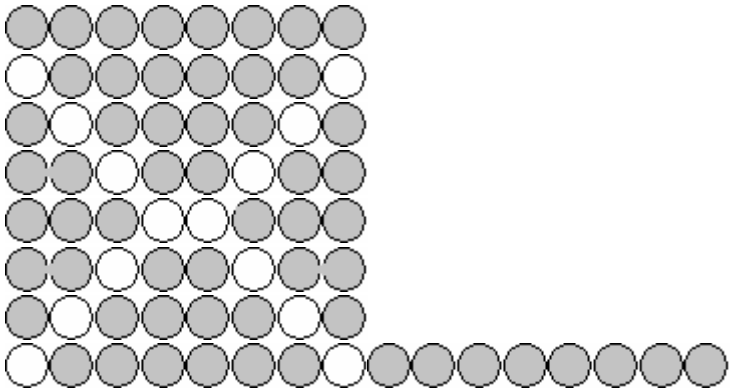
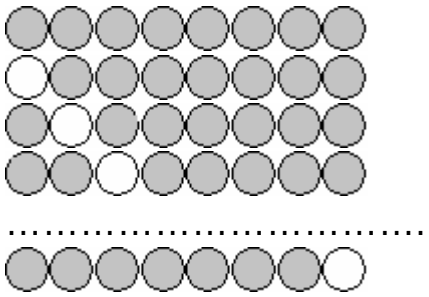
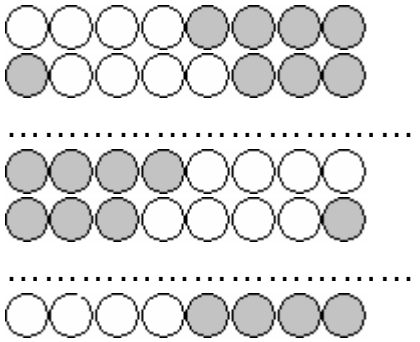
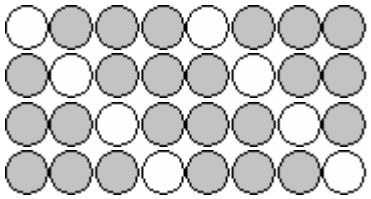
Цель: научиться управлять светодиодами стенда SDK 1.1 методом программирования регистров ПЛИС MAX3064. Написание библиотеки с функциями управления диодами для последующего использования

Задание: написать программу, позволяющую зажигать светодиоды в указанных последовательностях. В таблице 6.1 приведены варианты заданий.

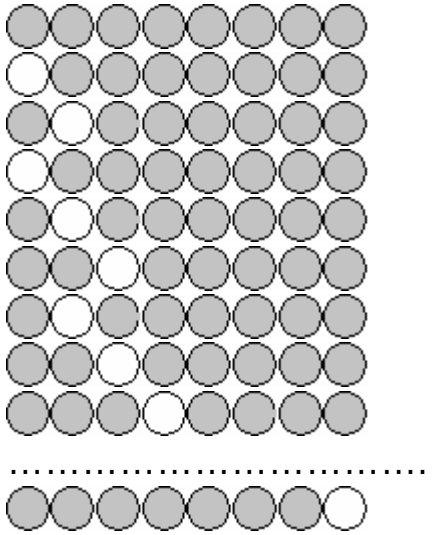
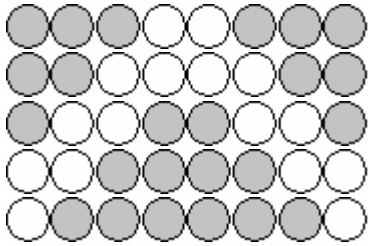
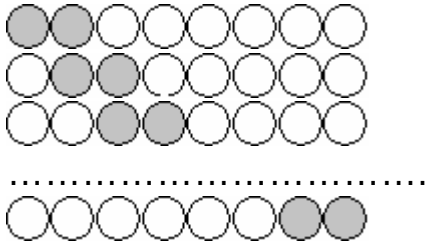
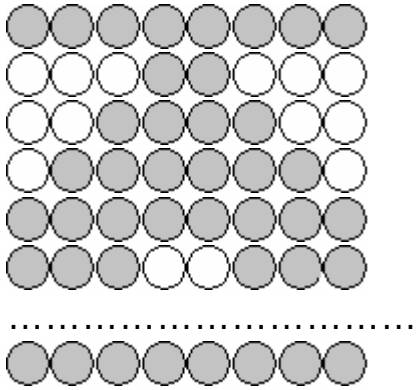
Таблица 6.1 – Варианты заданий

1	
2	

Продолжение таблицы 6.1

3	
4	
5	
6	

Продолжение таблицы 6.1

7	
8	
9	
10	

Листинг программы для варианта №4

Файл main.c

```
#include "ADuC812.h" // подключение модуля внутренней памяти
```

```
#include "LED.c" // подключение модуля светодиодов

void main(void)  // заголовок главной функции
{
// определение объектов
        unsigned int i;
        unsigned char Leds = 0;

        while(1)        //запуск бесконечного цикла
        {
                SetLed(Leds); // вызов функции зажигания светодиодов
                Leds <<=1; // сдвиг влево
                if(Leds & 0x80) // если зажжен последний светодиод
                {
                        for(i = 0; i < 6000; i++); // пустой цикл для задержки
                }
                else
                {
                        Leds |=1;       //  or 1
                        for(i = 0; i < 6000; i++); // пустой цикл для задержки
                }
        }
}
```

Файл LED.c

```
extern void SetLed(unsigned char val); // функция зажигания светодиодов
void SetLed(unsigned char LedStat)
{
  unsigned char oldDPP=DPP; // определение объектов
  unsigned char xdata *reg = 0x7;
  DPP = 0x8;

  *reg = LedStat;
  DPP=oldDPP;
}
```

6.2 Лабораторная работа №2. Матричная клавиатура

Цель: научиться работать с матричной клавиатурой методом программирования регистров ПЛИС MAX3064. Написать библиотеку с функциями обработки клавиатуры для последующего использования.

Задание: написать программу, позволяющую выполнить действия, соответствующие номеру варианта.

В таблице 6.2 приведены варианты заданий.

Таблица 6.2 – Варианты заданий

1	Зажечь светодиоды при вводе с клавиатуры указанной последовательности из 3 нажатий и погасить после повторного ввода последовательности
2	Мигать светодиодами, пока полностью зажат один ряд и не нажато ни одной другой клавиши
3	Мигать светодиодами, пока полностью зажат один столбец и не нажато ни одной другой клавиши
4	Зажечь один светодиод и управлять его перемещением с помощью клавиш 4-6.
5	Иметь функции сдвига влево вправо светодиодного ряда и установки/сброса состояния нулевого светодиода
6	Зажигать линию светодиодов, начинающуюся с младшего разряда и имеющую длину равную сумме столбца сканирования и строки нажатой клавиши
7	На светодиодном индикаторе отображать сумму всех нажатых цифровых клавиш
8	Зажечь светодиоды так, чтобы первые четыре соответствовали номеру сканируемого столбца, а последние - его состоянию
9	С помощью светодиодов вывести ASCII код нажатой клавиши
10	Зажигать первый и последний светодиод клавишами расположенными в шахматном порядке. (Клавиши на месте белых клеток зажигают один диод, клавиши на месте чёрных - другой)

Листинг программы для варианта №4

Файл main.c

```
#include "ADUC812.H" // подключение модуля внутренней памяти
#include "kbd.c" // подключение модуля клавиатуры
#include "led.c" // подключение модуля светодиодов

void main (void) // заголовок главной функции
{
	// описание типов данных
	unsigned char key;
	unsigned short inp;
```

```
        key = 0xf0; // адрес светодиода
        SetLed(key); // зажечь светодиод
        // цикл работы
        while(1)
        {
                if(KB_Hit(&key))SetLed(~key); // сканируем клавиатуру, зажигаем светодиод
                else SetLed(0x0); // потушить светодиоды
        }
}
unsigned char KB_Hit(char *pcBuff)
{
        // описание типов данных
        char i,j,col,row;
        unsigned short p;
        unsigned char xdata *reg = 0x0;
        unsigned char OldPage;
   // цикл нажатия клавиши по колонке
        for(i = 0; i < 4; i++ )
        {
                col  = 0x1 << i;
                OldPage = DPP; // сохранение значения DPP
                DPP     = 0x8; // переключение на 8 страницу
                *reg    =  ~col; // присвоение указателю значения переменной col
                DPP     = OldPage; // переключение на предыдущую страницу
                //  цикл нажатия клавиши по ряду
                        for(j = 0; j <4; j++) // запуск цикла
                {
                        DPP     = 0x8; // переключение на 8 страницу
                        row     = *reg; // присвоение переменной row значения переменной col
                        DPP     = OldPage; // сохранение значения DPP
                        row    &= 0x10 << j;
                        if(!row)
                        {
                                for(p = 0; p < 10000; p++); // запуск цикла
                                DPP     = 0x8; // переключение на 8 страницу
                                row     = *reg; // присвоение переменной row значения переменной
col
                                DPP     = OldPage; // переключение на предыдущую страницу
                                row    &= 0x10 << j;

                                if(!row)
                                {
                                        DPP     = 0x8; // переключение на 8 страницу памяти
                                        row     = *reg; // присвоение переменной row значения пе-
ременной col
                                        DPP     = OldPage; // переключение на предыдущую стра-
ницу
```

```
                                *pcBuff = row;
                                return 1;
                            }

                        }

                    }

        }
        return 0;
}
```

6.3 Лабораторная работа №3. Динамик

Цель: программирование регистров ПЛИС MAX3064 с целью получения звукового сигнала. Написание библиотеки с функциями управления динамиком для последующего использования.

Задание: написать программу, позволяющую выполнить действия, соответствующие номеру варианта.

В таблице 6.3 приведены варианты заданий.

Таблица 6.3 – Варианты заданий

1	Имитировать сигнал «внимание всем», частота которого медленно меняется по синусоиде с задержками в максимуме и минимуме на какое-то время
2	Имитировать стук колес вагона
3	Воспроизвести сигнал если нажаты все кнопки одной строки
4	Воспроизвести звуковую гамму, состоящую из звуков разной тональности, но одинаковой длительности
5	Воспроизводить звук, частота которого контролируется с помощью клавиатуры: повышение частоты нажатием одной клавиши, понижение - другой
6	Воспроизводить звук, частота которого зависит от нажатой клавиши

Продолжение таблицы 6.3

7	Имитировать сигнал о переполнении буфера клавиатуры. Выдавать короткий звуковой сигнал при нажатии более двух клавиш
8	Воспроизвести сигнал, если нажаты все кнопки одного столбца
9	Имитация импульсного набора номера
10	При правильном наборе пароля (последовательности из 3 клавиш) имитировать звук установки машины на сигнализацию

Листинг программы для варианта №4

```
#include <ADUC812.H> // подключение модуля внутренней памяти

#include "sound.c" // подключение модуля звука
#include "LED.c" // подключение модуля светодиодов

void main(void) // заголовок главной функции
{
// определение объектов
        unsigned int i,j;
        unsigned char Leds = 0, Val= 0;
        while(1)

// Создание задержек для мигания светодиодов и формирования звука
{               Leds = 0; // обнуление переменной leds
                for(Val = 0; Val<8; Val++) // запуск цикла
                {
                        Leds <<= 0x1; // сдвиг влево
                        Leds  |= 0x1; // or 1
                        SetLed(Leds); // зажигание светодиодов
                        for(i = 0; i < 60 + (Val > 4? -20: 0); i++) // запуск цикла
                        {       Snd_Enable(1); // включение динамика
                                for(j = 0; j < 300; j++ ); // задержка (время звучания)
                                Snd_Enable(0); // отключение динамика
                                for(j = 0; j <  300 - Val *40; j++); // задержка
                        }
                }

        }
}
```

6.4 Лабораторная работа №4. LCD дисплей

Цель: написание библиотеки с функциями управления дисплеем для последующего использования.

Задание: написать программу, позволяющую выполнить действия, соответствующие номеру варианта.

В таблице 6.4 приведены варианты заданий.

Таблица 6.4 – Варианты заданий

1	Заполнить экран случайными символами из латинского алфавита
2	Вывести символ «*» и управлять его передвижением с помощью клавиатуры.
3	Вывести на экран ASCIIZ строку из памяти
4	Написать программу, которая должна получить строку из символов и вывести её на экран дисплея
5	Вывести несколько ASCIIZ строк, используя вертикальную прокрутка с помощью клавиатуры
6	С помощью символа 0xFF и пробела заполнить дисплей в шахматном порядке
7	Заполнить экран символами, полученными с клавиатуры
8	Получить в программе число типа float (например, с помощью функции sin()) и вывести его в формате с фиксированной точкой (4-5 символов после точки)
9	Вывести случайное число в десятичной системе и продублировать его же в двоичной
10	Вывести количество нажатых кнопок на клавиатуре

Листинг программы для варианта №4

Файл main.c

```
#include <ADUC812.H> // подключение модуля внутренней памяти
#include "lcd.c" // подключение модуля дисплея
#include "LED.c"// подключение модуля светодиодов

void main(void)  // заголовок главной функции
{
// описание типов данных
        unsigned char ucCol = 1;
        unsigned char ucRow = 0;
        char cDir = 1;
```

```
        unsigned char ucComand;
        unsigned short i = 0;


    SetLed(0xfe); // зажигание светодиода
        LcdInit(); // инициализация LCD
        SetLed(0x00); // выключение светодиодов

        while(1) // запуск бесконечного цикла
        {
                ucCol   += cDir; // увеличение ucCol на значение cDir
                if(ucCol == 0) // сравнение ucCol с 0
                {
                        cDir = +1; // увеличение cDir
                        ucRow = 0; // обнуление ucRow
                }
                if(ucCol == 15) // если ucCol = 15
                {
                        cDir = -1; // уменьшение значения переменной cDir
                   ucRow = 1; // присвоение ucRow значения равного единице
                }

                ucComand = 0x80 | (ucCol & 0xf) | (ucRow ? 0x40:0);
                SetLed(ucComand); // зажечь светодиоды

                SetLcdData(ucComand);  // вывод на LCD
                LcdBisEnable(0,0);

                SetLcdData(0x43);
                LcdBisEnable(0,1);
                for(i = 0; i < 20000; i++)continue;
                SetLcdData(1);
                LcdBisEnable(0,0);
                for(i = 0; i < 20000; i++)continue;

        }
}
```

Файл lcd.c

```
void SetLcdData(unsigned char ucNewData)  // помещает байт в регистр данных LCD
{
        // определение типов данных
        unsigned char xdata * ucxLcdData = 0x01;
        unsigned char ucLastDPP = DPP;

        DPP = 0x08; // присвоение DPP адреса памяти 0x08
        *ucxLcdData   = ucNewData;
```

```
	DPP = ucLastDPP;
}

void LcdBisEnable(char cReadWrite /*1-read 0-write*/, char cDataCode /*1-data 0-command*/)
{
	// определение типов данных
	unsigned char ucStrobe = 0;
	unsigned char xdata * ucxLcdData = 0x06;
	unsigned char ucLastDPP = DPP;
	int i = 0;
	DPP = 0x08;

	ucStrobe |= 0x9; // сигнал E = 1  и Reserved = 1;
	if( cReadWrite)ucStrobe |= 0x2; // бит RW
	if( cDataCode) ucStrobe |= 0x4; // бит RS

	*ucxLcdData   = ucStrobe;  // запись в регистр
	for( i = 0; i < 10; i++)continue;
	*ucxLcdData   = ucStrobe & 0xFE;  //обнуление разрешающего сигнала E
	DPP = ucLastDPP;
	for( i = 0; i < 300; i++) continue; // задержка на исполнение команды
}

void LcdInit() // функция инициализации LCD
{
	int i = 0;
	for(i = 0; i < 4000; i++)continue; // задержка для ожидания конца переходных процессов


	SetLcdData(0x30);    // 00110000 – установка битности дисплея 8bit
	LcdBisEnable(0x0,0x0);  //
	for(i = 0; i < 1500; i++)continue; // задержка по протоколу

	SetLcdData(0x30);    // 00110000 – установка битности дисплея 8bit
	LcdBisEnable(0x0,0x0);  //
	for(i = 0; i < 50; i++)continue; // задержка по протоколу

	SetLcdData(0x30);    // 00110000 - установка битности дисплея 8bit     LcdBisEn-
able(0x0,0x0);  //
	for(i = 0; i < 150; i++)continue; // задержка по протоколу

	SetLcdData(0x38);    // 00111000 - установка битности дисплея 8bit и 2 линии
	LcdBisEnable(0x0,0x0);
	for(i = 0; i < 150; i++)continue; // задержка по протоколу

	SetLcdData(0x08);    // 00001000 – отключение дисплея
	LcdBisEnable(0x0,0x0);
	for(i = 0; i < 150; i++)continue; // задержка по протоколу
```

```
    SetLcdData(0x01);    // 00000001 - отключение дисплея
    LcdBisEnable(0x0,0x0);
    for(i = 0; i < 1500; i++)continue; // задержка по протоколу

    SetLcdData(0x06);    // 00000110 – начальные установки – направление
                                текста влево->вправо
    LcdBisEnable(0x0,0x0);
    for(i = 0; i < 150; i++)continue; // задержка по протоколу

    SetLcdData(0x0c);    // 00001100 – включение дисплея
    LcdBisEnable(0x0,0x0);
    for(i = 0; i < 150; i++)continue; // задержка по протоколу
}
```

6.5 Лабораторная работа №5. Арифметика

Цель: написание программы, выполняющей одно из арифметических действий. Операнды вводятся с клавиатуры. Результат отображается на ЖКИ.

Задание: написать программу, позволяющую выполнить действия, соответствующие номеру варианта.

В таблице 6.5 приведены варианты заданий.

Таблица 6.5 – Варианты заданий

1	Производить целочисленное деление 10-тизначного десятичного числа на 9-тизначное
2	Производить вычитание 10-значных десятичных чисел
3	Производить умножение 6-значных десятичных чисел
4	Производить сложение 10-значных десятичных чисел
5	Производить вычитание 10-значных десятичных чисел
6	Производить сложение 10-значных десятичных чисел
7	Производить умножение 6-значных десятичных чисел
8	Производить целочисленное деление 10-значного десятичного числа на 9-значное
9	Производить сложение 10-значных десятичных чисел
10	Производить умножение 6-значных десятичных чисел

Листинг программы для варианта №4

```
#include <ADUC812.H>  // подключение модуля внутренней памяти
#include <string.h>  // подключение модуля строк
#include<stdlib.h> подключение модуля библиотечной функции для формирования суммы
```

```
#include <stdio.h>// подключение модуля библиотечной функции для вывода информации

        // заголовок главной функции
        void main(void)
        // описание типов данных
        {
        unsigned short i = 0;
        unsigned short j = 0;
        unsigned char key;
        char buffer [33];
        int num=0;
        int rez;

        // инициализация дисплея
        LcdInit();
        while(1)
        // выбираем ключ
        {
        if(KB_Hit(&key))
        {
        switch(key)
        // формируем задержки, вводим значение, считаем и выводим на экран дисплея
        {
        case 0x7d: SetLcdData('0'); LcdBisEnable(0,1);for(i = 0; i < 20000; i++)continue;
num*=10; num+=0; break;
        case 0xee:  SetLcdData('1'); LcdBisEnable(0,1);for(i = 0; i < 20000; i++)continue;
num*=10; num+=1; break;
        case 0xed:  SetLcdData('2'); LcdBisEnable(0,1);for(i = 0; i < 20000; i++)continue;
num*=10; num+=2; break;
        case 0xeb:  SetLcdData('3'); LcdBisEnable(0,1);for(i = 0; i < 20000; i++)continue;
num*=10; num+=3; break;
        case 0xde:  SetLcdData('4'); LcdBisEnable(0,1);for(i = 0; i < 20000; i++)continue;
num*=10; num+=4; break;
        case 0xdd:  SetLcdData('5'); LcdBisEnable(0,1);for(i = 0; i < 20000; i++)continue;
num*=10; num+=5; break;
        case 0xdb:  SetLcdData('6'); LcdBisEnable(0,1);for(i = 0; i < 20000; i++)continue;
num*=10; num+=6; break;
        case 0xbe:  SetLcdData('7'); LcdBisEnable(0,1);for(i = 0; i < 20000; i++)continue;
num*=10; num+=7; break;
        case 0xbd:  SetLcdData('8'); LcdBisEnable(0,1);for(i = 0; i < 20000; i++)continue;
num*=10; num+=8; break;
        case 0xbb:  SetLcdData('9'); LcdBisEnable(0,1);for(i = 0; i < 20000; i++)continue;
num*=10; num+=9; break;
        case 0x7e: SetLed(0x10); SetLcdData(1); LcdBisEnable(0,0); rez=num; num=0; break;
//плюс
        case 0x7b: // переменные результата;
         rez+=num; // вывод строки
        sprintf(buffer,"%d",rez); // вывести на экран
```

```
          toLCD(&buffer,5,1,10000);
          break;  //равно
                        }
                }
        }
}
```

6.6 Лабораторная работа №6. Перехват прерываний

Цель: написание программу, в которой бы выполнялись одновременно задания для лабораторных работ № 1 и № 3 с помощью прерывания.

Задание: написать программу, в которой бы одно из заданий выполнялось в теле основной функции, а второе в обработчике прерывания.

Листинг программы для варианта №4

Файл main.c

```
#include <ADUC812.H> // подключение модуля внутренней памяти
#include "i2c.c""// подключение модуля шины I2C
#include "lcd.c " // подключение модуля дисплея

void main(void)  // заголовок главной функции
{
        // описание типов данных
        unsigned char ucComand;
        unsigned char ucRow;
        char *helStr = "secund";

        LcdInit(); // инициализация LCD
        while(*helStr) // запуск цикла
   {
                SetLcdData(0x80 | ucRow++ );
                LcdBisEnable(0,0);

                SetLcdData(*helStr); // вывод на LCD
                LcdBisEnable(0,1);

                helStr++; // инкремент количества секунд
        }

        while(1) // запуск бесконечного цикла
        {
                I2CM=1;

           i2cStartCondition(); // состояние «старт» для интерфейса I2C
                SendByte(0xA0); // посылка байтов по шине
```

```
            SendByte(0x02);
            wait(); // ожидание
            wait();

            i2cStartCondition(); // состояние «старт» для интерфейса I2C
            SendByte(0xA1); // посылка байтов по шине
            wait(); // ожидание
            ucComand = RecvByte(); // получение байтов с шины данных I2C
            Nack();

            i2cStopCondition(); // состояние стоп для интерфейса I2C

            SetLcdData(0xC0); // вывод на LCD
            LcdBisEnable(0,0);

            SetLcdData( (ucComand >> 4)  + '0');
            LcdBisEnable(0,1);

            SetLcdData(0xC1 );
            LcdBisEnable(0,0);
            SetLcdData( (ucComand & 0x0f)  + '0');
            LcdBisEnable(0,1);

        }
}
```

6.7 Лабораторная работа №7. Многозадачность

Цель: написание программы, в которой бы выполнялись одновременно задания для лабораторных работ № 1 и № 3, используя исключающую многозадачность.

Задание: расположить в двух бесконечных циклах соответствующие подпрограммы и добиться их одновременного выполнения.

Листинг программы для варианта №4

Файл main.c

```
// подключение необходимых модулей
#include <ADUC812.H>

extern void TimerIrp(void);   // вызов функции таймера

void SetVect(unsigned char xdata * addr, void * handler) // объявление функции

{
   unsigned short xdata *TmpjampDist;       // объявление типов данных
```

```
    addr += 0x2000;
   *addr = 0x2;

          TmpjampDist  =  ( unsigned short xdata *) (addr + 1);
          *TmpjampDist =(unsigned short) handler;

}

void SetLed(unsigned char LedStat) // функция зажигания светодиодов

{
// объявление типов данных
   unsigned char oldDPP=DPP;

   unsigned char xdata *reg = 0x7;

  DPP = 0x8; // переключение на восьмую страницу памяти
   *reg = LedStat; // зажигание светодиода
   DPP=oldDPP; // возвращение на прежнюю страницу памяти
}

void Snd_Enable(unsigned char Value) //подключение функции звука

{
   unsigned char oldDPP=DPP; // описание типов данных

   unsigned char xdata *reg = 0x4;

   unsigned char EReg=0;
   DPP = 0x8;  // переключение на восьмую страницу
   if(Value)EReg = 0x10; // включение звука
   else
   EReg     = 0x0;
   *reg = EReg;
   DPP=oldDPP; // возврат на предыдущую страницу
   return;

}
// описание переменных
char bLight = 0, cLig= 0;
unsigned short aret = 0;

extern unsigned char  xdata  cThread    _at_ 0xff00;
extern unsigned short xdata usRetAdr[2] _at_ 0xff12;

void main(void)
{
          int iVar = 0; // обнуление переменных
```

```
    int iJ   = 0;
     cThread = 0;

    *((unsigned short xdata*)(0x00ff12)) = 0x21db; // указатель на адрес памяти
    *((unsigned short xdata*)(0x00ff14)) = 0x21db;

    SetVect(0x0B, (void *)TimerIrp); // установка вектора прерываний

    ET0 = 1;
    EA  = 1;
    TR0 = 1;
    TMOD =0x11;
    TH0 = 0x00;
    TL0 = 0xc0;
    TF0 = 1;

    while(1) // запуск бесконечного цикла

    {
        ET0 = 0;        // остановка прерывания
        SetLed(iVar >> 8); // зажигание светодиодов
        iVar++; // инкремент переменной iVar
        ET0 = 1;        // разрешение прерывания
    }

    iJ = 0; // обнуление переменной iJ
    while(1) // запуск бесконечного цикла

    {
        ET0 = 0; // остановка прерывания
        Snd_Enable(1); // включение звука

        ET0 = 1; // разрешение прерывания
        for(iJ = 0; iJ < 3000; iJ++); // запуск цикла
        ET0 = 0; // остановка прерывания
        Snd_Enable(0); // выключение звука
        ET0 = 1; // разрешение прерывания
        for(iJ = 0; iJ < 3000; iJ++); // запуск цикла
    }
}
```

Файл mor.c

```
// подключение необходимых модулей
#include <ADUC812.H>

// определение типов данных
unsigned char  xdata  cThread    _at_ 0xff00;// _at_ 0x2ff0;
```

```
unsigned short xdata ucCurAdr    _at_ 0xff10;// _at_ 0x2ff2;
unsigned short xdata usRetAdr[2] _at_ 0xff12;//{0xff12,0xff14};// _at_ 0x2ff9;

void TimerIrp(void) interrupt 1 // функция таймера

{
        ET0 = 0; // остановка прерывания
#pragma asm // подключение ассемблера
        MOV  DPTR,#0xff10 // запись адреса в указатель внешней памяти
        MOV A, SP; // запись в A содержимого регистра SP
        SUBB A, #0x5;  // уменьшение значения A на 5
        MOV R1, A // запись значения аккумулятора в R1
        MOV A, @r1; //  запись в аккумулятор содержимого ячейки памяти по адресу r1
        MOVX @DPTR, A // запись значения аккумулятора по адресу, указанному в DPTR
        DEC R1; // декремент R1
        INC DPTR; // инкремент DPTR
        MOV A, @R1; // запись в аккумулятор содержимого ячейки памяти по адресу r1
        MOVX @DPTR, A// запись значения аккумулятора по адресу, указанному в DPTR

#pragma endasm // конец вставки

                TH0 = 0xf0;
                ET0 = 1; // включение таймера
                if(cThread == 0)

                {
                                usRetAdr[0] = ucCurAdr;
                                 cThread     = 0x1;
                                ucCurAdr    = usRetAdr[1];
                }

                else

                {
                        usRetAdr[1] = ucCurAdr;
                        cThread = 0x0;
                        ucCurAdr    = usRetAdr[0];
                }

                #pragma asm // подключение ассемблера

        MOV DPTR,#ucCurAdr // присвоение DPTR адреса, содержащегося в ucCurAdr

        MOV A, SP; // запись в A содержимого регистра SP
        SUBB A, #0x5; // уменьшение значения A на 5
        MOV R1, A // запись значения аккумулятора в R1
        MOVX A, @DPTR // запись в аккумулятор значения, содержащегося по адресу ука-
занному в DPTR
```

```
        MOV  @R1, A; // запись по адресу указанному в R1, значения указанного в аккумуляторе
        DEC R1;        // декремент R1
        INC DPTR; // инкремент DPTR
        MOVX A, @DPTR // запись в аккумулятор значения, записанного по адресу указанному в DPTR
        MOV  @R1, A; // запись по адресу, указанному в R1, значения содержащегося в аккумуляторе.

#pragma endasm // конец вставки на ассемблере
}
```

7 Лабораторный практикум по курсу «Микропроцессорные системы»

Учебный лабораторный комплекс SDK-1.1 построен на основе микроконтроллера семейства Intel 51 и включает все функциональные блоки, соответствующие законченной микропроцессорной системе: процессор, модули памяти, систему ввода-вывода, систему прерываний, периферийные микросхемы и т.д. Выполнение лабораторных работ позволит глубже понять принципы построения, программирования и функционирования микропроцессорной системы - SDK-1.1, получить навыки работы с реальным микропроцессорным устройством.

7.1 Лабораторная работа №1. Изучение архитектуры учебного стенда SDK 1.1. Работа со светодиодами

Цель работы:

- изучить структуру аппаратной части стенда;
- изучить распределение памяти SDK 1.1;
- изучить ПЛИС регистры, их адреса, назначение битов и доступ к регистрам;
- научиться использовать команды чтения и записи регистров ПЛИС MAX8064;
- научиться использовать команда управления светодиодами и динамиком.

Варианты заданий:

Вариант 1

Осуществить наглядный счёт (на светодиодах с задержкой) в двоичном коде от 0 до 64(10). По окончании счёта необходимо «зажечь» все светодиоды и воспроизвести звук любой тональности.

Вариант 2

Зажечь светодиоды поочередно. В результате горит один светодиод, затем два, три и т.д. После зажигания всех светодиодов необходимо воспроизвести звук любой тональности. Интервал времени между зажиганиями следующих светодиодов должен быть подобран исходя из соображений удобности (возможности) восприятия.

Вариант 3

Воспроизвести звуковой сигнал 8 раз. Каждый следующий звуковой сигнал должен иметь большую длительность, чем предыдущий. После воспроизведения звукового сигнала зажигать светодиод соответствующий его порядковому номеру. После зажигания светодиода организовать задержку примерно равную 0,5 секунды.

Вариант 4

Зажечь одновременно все светодиоды стенда, затем погасить их и воспроизвести звуковой сигнал любой тональности. Повторить эти действия 5 раз. Между повторениями организовать задержку примерно равную 0,5 секунды.

Вариант 5

Зажигать светодиоды, начиная с крайних и постепенно дойти до центральных светодиодов. После зажигания всех светодиодов воспроизвести звуковой сигнал любой тональности и погасить светодиоды. Повторить эти действия 5 раз. Между повторениями организовать задержку примерно равную 0,5 секунды.

Листинг программы для варианта №2

```
// подключение необходимых модулей

#include <ADUC812.H>
#include "led.h"
#include "sound.h"

void pause(unsigned int interval); // объявление функции pause
int main(void)

{
	char svet = 1;  // объявление и инициализация переменной svet
	int i;
	for (i=0; i<8; i++) // цикл поочередно зажигающий светодиоды

	{
		SetLed(svet); // зажигание светодиодов
		svet <<= 1; // сдвиг влево
		svet |= 1; // установка последнего бита в единицу
		pause(60000); // пауза между зажиганием в 60000 тактов

	}

	while (1) // запуск бесконечного цикла
```

```
{
        Snd_Enable(1); // включение динамика
        pause(1000); // задержка
        Snd_Enable(0); // выключение динамика
        pause(1000); // задержка
    }
}

void pause(unsigned int interval) // процедура задержки

{
    unsigned int i;
    for (i = 0; i < interval; i++);
}
```

7.2 Лабораторная работа №2. Приобретение навыков работы с матричной клавиатурой стенда

Цель работы:

- изучить организацию матричной клавиатуры AK1604-WWB;
- научиться использовать команды считывания данных с клавиатурной матрицы и использовать полученные результаты для решения поставленной задачи.

Варианты заданий

Вариант 1

Написать программу для стенда SDK1.1 осуществляющую инверсию соответствующего разряда байта при нажатии клавиши микроконтроллера от 1 до 8. Значение инвертируемого байта записано в программе и отображается на светодиодах.

Вариант 2

Ввести с клавиатуры последовательно два числа в диапазоне от 0 до 9. Осуществить перемножение этих чисел и по нажатию нецифровой клавиши стенда (одной по выбору) отобразить данное число в двоичном коде на светодиодах.

Вариант 3

Написать программу для стенда SDK1.1, которая отображает на светодиодах двоичный код цифры, введённой с клавиатуры микроконтроллера. Отображать код при каждом нажатии клавиши. При нажатии клавиши “D” инвертировать значение регистра светодиодов.

Вариант 4

Прибавлять к переменной, отражающей сумму, число соответствующее номеру нажатой клавиши микроконтроллера и отображать сумму на светодиодах. Начальное значение суммы равно 0. При значении суммы превышающее 255 погасить все светодиоды.

Вариант 5

Задать значение некой переменной, отражающей сумму, в диапазоне от 0 до 255. При нажатии клавиши “1” вычитать из суммы единицу, при нажатии “2” осуществить прибавление единицы. При выходе за диапазон 0-255 справа продолжать счёт с 0, при выходе за границу диапазона слева продолжать счёт с 255. Отображать двоичный код значения переменной суммы при каждом нажатии клавиши. При нажатии клавиши “D” инвертировать значение переменной суммы.

Листинг программы для варианта №2

```
// подключение необходимых модулей

#include <ADUC812.H>
#include "led.h"
#include "kbd.h"
#include "sound.h"
unsigned char KeyCode(unsigned char ikey);

int main(void)
{

	unsigned char key1,key2,key3; // определение типов данных

	while (1) // запуск бесконечного цикла
	{

		key1=key2=key3=0;  // обнуление переменных

		while (!KB_Hit(&key1)); // сканирование клавиатуры до тех пор пока не будет нажата клавиша
		SetLed(KeyCode(~key1)); // вывести значение клавиши на светодиоды
		pause(600*10); // задержка

		while (!KB_Hit(&key2));
		SetLed(KeyCode(~key2));
		pause(600*10000);
```

```
            key1 = KeyCode(~key1); // получение числового значение первой клавиши
            key2 = KeyCode(~key2); // получение числового значения второй клавиши

            while (1) // бесконечный цикл
            {
                  if (!KB_Hit(&key3))  // ожидание нажатия клавиши «A»
                        if ((~key3)==0x18)

                  {

                  SetLed(~(key1*key2)); // вывод значение на светодиоды
                  break;

                  }
            }
            pause(600*10000);
            SetLed(0);      // отключение светодиодов
      }
      while(1);
}

unsigned char KeyCode(unsigned char ikey) // функция перевода кода нажатой клавиши в цифровое значение
{
      switch(ikey)
      {

            case 0x82:      return 0;
            case 0x44:      return 9;
            case 0x42:      return 8;
            case 0x41:      return 7;
            case 0x24:      return 6;
            case 0x22:      return 5;
            case 0x21:      return 4;
            case 0x14:      return 3;
            case 0x12:      return 2;
            case 0x11:      return 1;

      }
      return 0;
}
```

7.3 Лабораторная работа №3. Работа с таймером/счетчиком и жидкокристаллическим индикатором учебного стенда

Цель работы:

- изучить организацию ЖКИ WH1602B-YGK-CP: структуру и функции;
- изучить устройство и функционирования счётчиков (TLx и THx) микроконтроллера;
- научиться использовать регистры команд и данных ЖКИ и счётчиков;
- изучить команды по работе с ЖКИ.

Описание работы

Данная лабораторная работа посвящена изучению таймера и жидкокристаллического индикатора (ЖКИ) стенда SDK-1.1.

Основными функциями системного таймера является измерение интервалов времени и выполнение периодических задач. В данной работе с помощью таймеров требуется управлять светодиодным индикатором или звуковым излучателем, входящими в состав контроллера SDK-1.1. Драйвер системного таймера должен включать функции, указанные в таблице 7.1.

Таблица 7.1 – Функции системного таймера

Функция	Описание
void InitTimer (void)	Инициализация таймера.
unsigned long GetMsCounter (void)	Получение текущей метки времени в миллисекундах
unsigned long DTimer (unsigned long t0)	Измерение количества миллисекунда, прошедших с временной метки t0 до текущего времени.
void DelayMs (unsigned long t)	Задержка на t миллисекунд.

Драйвер ЖКИ должен включать функции, указанные в таблице 7.2. Кроме того, может быть реализована функция вывода строки на ЖКИ, функция дополнительной настройки ЖКИ (отображение, мерцание курсора). Драйвер клавиатуры использует прерывание таймера, в котором производится опрос состояния кнопок. В данном случае можно применить автоматное программирование. В драйвер клавиатуры рекомендуется включить функцию чтения нажатых кнопок из буфера, связывающего ее с обработчиком прерывания. Реакции на нажатия кнопок клавиатуры должны формироваться в главной программе. Вся обработка нажатий кнопок не должна быть локализована в обработчике прерываний таймера.

Таблица 7.2 – Функции ЖКИ

Функция	Описание
void InitLCD (void)	Инициализация ЖКИ.
void WriteControlLCD (unsigned char ch)	Запись значения в регистр управления ЖКИ C_IND (ПЛИС): ch – значение, записываемое в C_IND.
bit ReadFLCD (void)	Чтение флага BF (флаг занятости контроллера ЖКИ).
unsigned char ClearLCD (void)	Очистка дисплея с возвратом результата выполения операции.
unsigned char GotoXYLCD (unsigned char x, bit y)	Переход в заданную позицию дисплея с возвратом результата выполнения операции: x, y – координаты позиции.
unsigned char PrintCharLCD (unsigned char symbol)	Вывод символа на дисплей с возвратом результата выполнения операции: symbol – выводимый символ.

Варианты заданий

Вариант 1

Разработать устройство, измеряющее интервал времени, в пределах от 0 до 10 секунд с точностью до 0,01 секунды. Начало и конец интервала задаётся нажатием клавиши на клавиатуре учебного стенда SDK 1.1. Результат измерения вывести на ЖКИ стенда в виде четырёхпозиционного десятичного кода. Измеренный интервал сравнить с эталоном, указанным преподавателем. Если измеренное значение интервала времени больше эталона, то зажечь светодиод 1, если меньше, то светодиод 8, а если равно, то все светодиоды.

Вариант 2

Разработать устройство, имитирующее работу секундомера. На клавиатуре должны быть отведены три клавиши для Старта секундомера, Стопа и Сброса. Точность отображения времени секундомером равна 0,1 секунды. Ход времени необходимо отображать на ЖКИ микроконтроллера. При достижении 99 секунд секундомер автоматически останавливается и микроконтроллером воспроизводится звук любой тональности.

Вариант 3

Разработать устройство, имитирующее работу таймера обратного отсчёта. На клавиатуре должны быть отведены три клавиши для Старта таймера, Стопа и Сброса. Точность отображения времени равна 0,1 секунды. Ход времени необходимо отображать на ЖКИ микроконтроллера. Перед началом работы вводится значение таймера в интервале от 00.1 до 99.0 секунд. При нажатии Сброса таймер должен возвращаться к первоначальному значению, а при достижении нуля микроконтроллером воспроизводится звук любой тональности.

Вариант 4

Разработать устройство, осуществляющее счёт времени в прямом или обратном порядке. На клавиатуре должны быть отведены три клавиши для Счёта, Обратного счёта и Стопа. Точность отображения времени равна 0,1 секунды, а интервал счёта равен от 00.0 до 99.0 секунд. Ход времени необходимо отображать на ЖКИ микроконтроллера. При достижении границ интервала счёта таймер останавливается.

Вариант 5

Разработать устройство, измеряющее интервал времени, в пределах от 0 до 10 секунд. Точность измерения интервала равна 0,01 секунды. Выполнить 3 измерения подряд.

В результате работы устройства на дисплее микроконтроллера должна быть отражена величина наибольшего из измеренных отрезков времени в виде четырёхпозиционного десятичного кода. А также номер измерения. Сигналом начала и окончания отсчёта служит нажатие клавиши на клавиатуре микроконтроллера, а индикатором счёта является мерцание светодиодов (1 раз в секунду или в полсекунды).

Листинг программы для варианта №2

```
// подключение необходимых модулей

#include <ADUC812.H>
#include "led.h"
#include "kbd.h"
#include "sound.h"
#include "interput.h"
#include "lcd.h"

void TimeToStr(void);          // функция перевода времени в строку

// объявление переменных
unsigned int uiCounter = 0;
unsigned int uiTime = 1;       // 1/10 seconds
```

```
char szTime[6]="-----";
unsigned char ucKey;

void TimerInt(void) interrupt 1 using 2 // функция, срабатывающая по прерыванию от таймера

{
	unsigned char ucOldPage;	// переменная хранит старое значение регистра DPP
	unsigned int i = 0;
	ET0 = 0;		// отключение таймера
	ucOldPage = DPP;
	uiCounter++;  // инкремент счетчика срабатывания прерываний
	if (uiCounter >= 0x15) // если количество прерываний больше 15h
	{
		SetLed(uiTime);		// вывод значения времени
		uiTime += 1;  // инкремент переменной времени
		if (uiTime == 10000) // если переменная времени = 10000
		{
			TimeToStr(); // преобразование времени в строку
		   LCD_GotoXY(5,0);		// установка курсора LCD
			LCD_Type(szTime); // вывод времени на LCD
			SetLed(0); // выключение светодиодов
			for (i=0; i<100000; i++) // воспроизвести звуковой сигнал
			{
				Snd_Enable(1); // включить динамик
				pause(100); // задержка
				Snd_Enable(0); // выключить динамик
				pause(100); // задержка
			}

			return;
		}
		uiCounter = 0;// обнуление счетчика
		TimeToStr();  // преобразование времени в строку

		if (uiTime%10 == 5) // вывод значения на экран через каждые 0,5 с
		{
			LCD_GotoXY(5,0);
			LCD_Type(szTime);
		}
	}
	else
	{

		TH0 = TL0 = 0; // обнуление таймера
		TR0 = 1; // включение прерывания

	}
	DPP = ucOldPage; // вернуть старое значение регистра
```

```
   ET0 = 1; // включить таймер
}

int main(void)
{
        SetVect(0x0B,(void*)TimerInt); // установить вектор прерывания таймера по адресу
0x0B
        TH0 = TL0 = 0; // обнуление таймера
        LCD_InitDefault(); // инициализация LCD

        while(1)
        {
                if (KB_Hit(&ucKey)) // ожидание нажатия клавиши
                        if (~ucKey==0x11) // если нажата единица
                        {
                                IE = 0x82; // разрешение прерывания
                                TMOD = 0x2;
                                TCON = 0x10;
                                uiTime = 0;
                                break;
                        }
        }

        EA = 1;         // включение прерывания
        while(1)
        {
                ET0 = 0; // выключить таймер
                if (KB_Hit(&ucKey)) // ожидание нажатия клавиши
                {
                        ET0 = 1; // включить таймер
                        switch(~ucKey) // выбор действия
                        {
                                case 0x11:  // включение секундомера
                                        TMOD = 0x2;
                                        TCON = 0x10;
                                        break;
                                case 0x12: // выключение секундомера
                                        TR0 = 0;
                                        break;
                                case 0x21: // сброс секундомера
                                        SetLed(2);
                                        uiTime = 0;
                                        TH0 = TL0 = 0;
                                        TR0 = 0;
                                        TimeToStr();
                                        LCD_GotoXY(5,0); // установка курсора LCD
                                        LCD_Type(szTime); // вывод на LCD
                                        SetLed(0); // выключение светодиодов
```

```
                                        break;
                                }
                        }
                        else
                                ET0 = 1;
                        pause(500);
                }
}

 void TimeToStr(void) // функция преобразования значения переменной времени в строку
 {
         szTime[0] = (uiTime/1000) % 10 + '0';
         szTime[1] = (uiTime/100) % 10 + '0';
         szTime[2] = ',';
         szTime[3] = (uiTime/10) % 10 + '0';
         szTime[4] = uiTime % 10 + '0';
         szTime[5] = 0;
 }
```

7.4 Лабораторная работа №4. Получение навыков работы с портами ввода вывода и звуком

Цель работы:

- изучить регистры управления портами ввода-вывода EXT_HI, EXT_LO и регистр управления звуком ENA;

- научиться использовать команды управления звуком: изменение тональности и длительности звукового сигнала.

Описание работы

Требуется написать драйвер звукового излучателя, позволяющий задавать частоту (в герцах) и длительность звука (в миллисекундах). Для управления звуковым излучателем используется регистр ПЛИС ENA (адрес 080004h). 2-4 биты регистра ENA управляют величиной напряжения на динамике, т.е. позволяют задавать громкость звука: чем больше “единиц” выставлено в этих битах, тем громче звук. Частота звука задается частотой смены нулей и единиц в управляющих битах регистра ENA. Ниже, в таблице 7.3, приведены частоты нот первой октавы.

Таблица 7.3 - Частоты музыкальных нот

Нота	Частота, Гц
До	261,63
Ре	293,67

Ми	329,63
Фа	349,22

Продолжение таблицы 7.3

Соль	391,99
Ля	440,00
Си	493,88

Варианты заданий

Вариант 1

Изменять тональность звучания динамика в зависимости от младшего байта, поступившего на параллельный порт. Старший разряд байта имеет высшую тональность, выбор тона происходит по приоритету: старшие разряды имеют наивысший приоритет.

Генерировать входной сигнал на параллельный порт при помощи набора переключателей SW3.

Вариант 2

С помощью драйверов системного таймера и звукового излучателя написать тестовую программу, которая в случае замыкания одного из DIP-переключателей циклически проигрывает восходящую гамму нот первой октавы (длительность каждой ноты – 1 секунда). В случае замыкания другого DIP-переключателя на линейку светодиодов должно выводиться количество замыканий входа T0 (счетный вход таймера 0; выведен на DIP-переключатель). Подсчет количества замыканий входа должен быть реализован с помощью таймера/счетчика.

Листинг программы для варианта №2

```
// подключение необходимых модулей
#include <ADUC812.H>
#include "led.h"
#include "sound.h"
#include "paral.h"
// объявление переменных
short iReciv;
char byte = 0xF;
void main()
{       bit b; // объявление и установка значения битовой переменной
        int i,a = 100; //объявление и установка значения переменных i, a
        EA = 0; // отключение прерывания
```

```
        while(1) // запуск бесконечного цикла
        {       iReciv = ReadPort(); // считывание порта в двухбайтовую переменную
                byte = iReciv; // присвоение переменной byte старшего разряда переменной IRe-
civ
                SetLed(byte); // вывод на светодиоды
                for (i=0; i<8; i++) // определение номера активного переключателя
                {       b = 1&(byte>>i); // сдвиг и сравнение
                        if (b==1) {a=i; break; }        // если значение совпало, то выход из цикла
                }
                for (i=0; i<10; i++) // подача звука
                {
                 Sound(a*5);
                }
        }
        while(1);
}
```

7.5 Лабораторная работа №5. Работа с прерываниями и часами реального времени

Цель работы:

- изучить систему прерываний микроконтроллера и источники возможных прерываний;
- научиться создавать собственные процедуры обработки прерываний;
- изучить работу микросхемы PCF8583 – часы/календарь;
- изучить команды для программирования регистров управления и состояния микросхемы часов времени.

Описание работы

Драйвер часов реального времени должен включать функции, приведенные в таблице 7.4.

Таблица 7.4 – Функции часов реального времени

Функция	Описание
void InitRTC (void)	Инициализация часов реального времени
unsigned char ReadRTC (TimeDate *td)	Чтение даты и времени из RTC с возвратом результата выполнения операции: td – буфер даты и/или времени в виде структуры специального формата
Unsigned char WriteRTC (TimeDate *td)	Запись даты и времени из RTC с возвратом результата выполнения операции: td – буфер для даты и/или времени в виде структуры специального формата.

Варианты заданий

Вариант 1

Перехватить внешнее прерывание INT1. Процедура обработки данного прерывания должна запускать таймер (секунды) встроенный в часы реального времени. На ЖКИ выводится значения таймера, а по окончании счёта микроконтроллер должен автоматически воспроизвести сигнал, т.е. должен быть разрешён сигнал по таймеру.

Вариант 2

Перехватить внешнее прерывание INT0. Процедура обработки данного прерывания должна устанавливать ежедневный сигнал для часов реального времени на 15 секунд больше, чем текущее время и отобразить текущее время в формате ЧАС:МИН:СЕК на ЖКИ.

Вариант 3

Создать свои процедуры обработки следующих прерываний: переполнение таймера; внешнее прерывание.

Перехватить внешнее прерывание INT1 или INT0 (по выбору), процедура обработки данного прерывания должна запускать таймер (один из таймеров по выбору).

Перехватить прерывание переполнения таймера, процедура обработки его должна вывести на ЖКИ текущее время в формате 12-ти часового отображения (AM/PM) с точностью до сотых секунды и дату (месяц и число). Время и дату брать с часов реального времени микроконтроллера.

Вариант 4

Перехватить прерывание переполнения таймера 0. Процедура обработки данного прерывания должна вывести на ЖКИ текущую дату в следующем формате: День (прописью) : Число : Месяц (прописью) : Год.

Вариант 5

Перехватить прерывание переполнения таймера 1. Процедура обработки данного прерывания должна запускать таймер (минуты) встроенный в часы реального времени а также отобразить текущее время в формате ЧАС:МИН:СЕК на ЖКИ.

Первоначальное значение таймера выбирается так, чтобы задержка была не более 2-х минут. При переполнении таймера микроконтроллер должен автоматически воспроизвести сигнал, т.е. должен быть разрешён сигнал по таймеру.

Листинг программы для варианта №2

```
// подключение необходимых модулей
#include <ADUC812.H>
#include "led.h"
#include "interput.h"
#include "rtc.h"
#include "sound.h"
#include "lcd.h"
#include "string.h"
// описание переменных
TIME xdata time,time2;
bit req = 0;
void OutTime(); // функция вывода времени
void Int0Handler(void) interrupt 0 using 2 // функция, срабатывающая по внешнему прерыванию int0
{ bit res = 0; // объявление и обнуление переменной res
  EX0 = 0; // отключение внешнего прерывания
  if (req == 1) return; // проверяем на рекурсию
  req = 1; // установка переменной рекурсии в 1
  EX0 = 1;   // включение внешнего прерывания
}
void main()
{       unsigned char n,r; // объявление переменных
        int res = 0;
        LCD_InitDefault(); // инициализация LCD
        EA  = 1; // отключение прерывания
        EX0 = 1; // включение прерывания
        SetVect(0x3,Int0Handler); // установка вектора внешнего прерывания
        time2.sec = 0; // обнуление секунд
        res = SetTime(&time2); // установка времени
        while(1)
        {       EA = 0;  // отключение прерывания
                OutTime(); // вывод времени
                EA = 1; // включение прерывания
                pause(1000); // задержка
                EA = 0;  // отключение прерывания
                if (req == 1)  // если прерывание уже сработало
                {       GetTime(&time); // получение значения времени
                        time.sec += 15; // прибавление 15 с к значению времени
                        n = time.sec / 60; // нормализация времени
                        r = time.sec % 60;
                        time.min += n;
                        time.sec = r;
                        SetTime(&time);       // установка времени
                        req = 0; // обнуление переменной req
```

```
                pause(1000); // задержка
                EX0 = 1; // включить внешнее прерывание
            }
            EA = 1;  // разрешить прерывание
        }
}
void OutTime() // функция вывода времени
{       char str[11];
        memset(str,'-',11);
        GetTime(&time2); // получение значения времени
        str[10] = 0;
        str[9] = ((time2.sec ) % 10)  + '0';
        str[8] = ((time2.sec / 10) % 10) + '0';
        str[7] = ':';
        str[6] = ((time2.min ) % 10)  + '0';
        str[5] = ((time2.min / 10) % 10) + '0';
        str[4] = ':';
        str[3] = ((time2.hour ) % 10)  + '0';
        str[2] = ((time2.hour / 10) % 10) + '0';
        str[1] = ':';
        str[0] = 'T';
        SetLed(time2.sec); // вывод секунд на светодиоды
         LCD_GotoXY(0,0); // установка курсора на LCD
        LCD_Type(str); // вывод строки со значение времени
}
```

7.6 Лабораторная работа №6. Разработка системы сбора и обработки информации

Цель работы:

- закрепление навыков работы со всеми узлами учебного стенда SDK-1.1;
- изучение команд передачи данных между узлами учебного стенда SDK-1.1.

Описание работы

В рамках лабораторной работы необходимо разработать программу для контроллера SDK-1.1 (ведомый), который обменивается данными с персональным компьютером (ведущий). В качестве канала связи используется последовательный канал RS232. Устройством вывода терминал персонального компьютера. Драйвер клавиатуры должен содержать: функцию инициализации, обработчик прерывания от таймера, циклический буфер клавиатуры (нажатые кнопки), При инициализации необходимо указать задержку перед повтором

символа (первый параметр) и скорость повтора символа (второй параметр). Рекомендуемая величина задержки перед повтором – 1 секунда (не меньше).

Варианты заданий

Вариант 1

Выполнить задачу сбора голосов и их подсчёт. На экран монитора компьютера необходимо вывести список кандидатов с указанием их порядкового номера. При голосовании запрашивается номер кандидата (запрос на ЖКИ). При вводе номера необходимо вывести на ЖКИ учебного стенда SDK-1.1 фамилию кандидата и получить подтверждение о правильности выбора. Количество голосующих задаётся программно. После сбора информации необходимо вывести на экран монитора компьютера результаты голосования и количество голосов отданных за каждого кандидата.

Вариант 2

Подсчёт рейтинга группы. Программно вводится список имён экзаменуемых студентов. После команды начала выставления оценок, на ЖКИ учебного стенда SDK-1.1 последовательно выводятся фамилии студентов. Напротив каждой фамилии надо поставить оценку 1..5 и подтвердить правильность ввода. После сбора информации обо всех студентах, на экран монитора компьютера необходимо вывести общее количество студентов, фамилии студентов с отображением оценки и средний балл (рейтинг) группы.

Вариант 3

Кодировка текста. На ЖКИ учебного стенда SDK-1.1 вводится числовая последовательность, которая представляет собой закодированную текстовую информацию. Разделителем кода символа служит нечисловой символ (*,#,A,B,C,D). Количество символов в последовательности не должно быть более 80, а при заполнении всей рабочей области ЖКИ (16X2), она очищается. При подаче сигнала на раскодирование (нажатие клавиши) на экране монитора компьютера должна быть отображена текстовая строка и количество символов в ней.

Вариант 4

Фильтрация информации. На экране монитора компьютера отобразить фамилии людей (не менее десяти), с указанием года, месяца (числом) и дня рождения. Предусмотреть три варианта фильтрации данных: по году, по месяцу и по дню рождения. На ЖКИ учебного стенда SDK-1.1 отобразить варианты фильтра. Необходимо осуществить выбор фильтра и указать значение для осуществления отбора, при выборе дать запрос на правильность ввода. На экране монитора должна быть отображена отсортированная информация. Необходимо

также предусмотреть клавишу возвращения в первоначальный список (неотфильтрованный).

Вариант 5

Тест. На экран монитора компьютера по порядку выводятся вопросы для теста (не менее десяти). На ЖКИ учебного стенда SDK-1.1 должны быть отражены варианты ответов. При вводе ответа должен инициироваться запрос на подтверждение правильности. При окончании тестирования на ЖКИ учебного стенда SDK-1.1 выводится средний бал, а на экране монитора статистика ответов, т.е. номера вопросов и правильность (неправильность) ответа на них, а также общее количество вопросов и правильных ответов.

Листинг программы для варианта №2

```
// подключение необходимых модулей

#include <ADUC812.H>
#include "sound.h"
#include "lcd.h"
#include "kbd.h"
#include "string.h"
#include "sio.h"

// объявление переменных

int N = 5;
char mas[5][10];
char* pMas = mas;
char xdata ball[5];
char xdata nCurrentLine;
char key;

void Output(); // функция вывода на LCD
void OutputCom(); // функция вывода на com-порт

void main()

{
        unsigned char iNum = 0; // определение переменной iNum
        nCurrentLine = 0;
        N = 5;  // размерность массива
        strcpy(mas[0],"mr.Ivanov"); // инициализация массива
        strcpy(mas[1],"mr.Petrov");
        strcpy(mas[2],"mr.Eprst");
        strcpy(mas[3],"mr.Andrew");
        strcpy(mas[4],"mr.Pink");
```

```
        LCD_InitDefault(); // инициализация LCD
        memset(ball,0,N); // обнуление массива оценок
        Output(); // вывод на экран
        while(1) // запуск бесконечного цикла

        {
          if(KB_Hit(&key)) // сканирование клавиатуры

                {
                        switch(~key) // определение нажатой клавиши

                        {
                        case 0x18:
                         if (nCurrentLine)
                         nCurrentLine--; // уменьшение значения текущей строки
                        break;

                        case 0x28:
                         if (nCurrentLine!=N-1) // проверка на допустимость значения текущей записи
                        nCurrentLine++; // увеличение значения текущей строки
                        break;

                        case 0x48: // вывод на com-порт
                        OutputCom();
                        break;

                        default: // если нажата клавиша от 0 до 5

                        iNum = KeyCode(~key); // получение текущего значения

                                if (iNum>5) iNum = 5;
                                if (iNum<1) iNum = 1;
                                ball[nCurrentLine] = iNum; // установить оценку текущему пользователю

                        }

                        Output(); // функция вывода
                }
        }

        while(1);
}

void Output() // функция вывода на LCD
{
        char str[16] = "               ";
        double avg = 0;
```

```
	int i = 0;
	strncpy(str,"Shtuk:",6); // функция копирования строки в строку
	str[6] = N + '0'; // преобразовать в символ
	for(i = 0; i < N; i++) // подсчет среднего значения оценки

	{
		avg += ball[i];
	}

	avg /= (double)(N);

	strncpy(str+8,"Avg:",4); // формирование строки для вывода
	str[12] = (int)avg % 10  + '0';
	str[13] = ',';
	str[14] = (int)(avg*10) %10  + '0';
	str[15] = (int)(avg*100) % 10  + '0';
	str[16] = 0;

	LCD_Clear(); // очистка LCD

	LCD_GotoXY(0,0); // установка курсора LCD
	LCD_Type(str);		// вывод на LCD

	LCD_GotoXY(0,1); // установка курсора LCD
	LCD_Type(&mas[nCurrentLine][0]); // вывод на LCD

	str[0] = ball[nCurrentLine] % 10  + '0'; // вывод оценок
	str[1] = 0;
	LCD_GotoXY(11,1); // установка курсора LCD
	LCD_Type(str);		// вывод на LCD

}
void OutputCom() // вывод на com-порт
{
	char str[16] = "               ";
	int j = 0;
	double avg = 0;
	int i = 0;
	strncpy(str,"Shtuk:",6); // формирование строки для вывода
	str[6] = N + '0';
	for(i = 0; i < N; i++)
	{
		avg += ball[i];
	}
	avg /= (double)(N);

	strncpy(str+8,"Avg:",4);
	str[12] = (int)avg % 10  + '0';
```

```
        str[13] = ',';
        str[14] = (int)(avg*10) %10  + '0';
        str[15] = (int)(avg*100) % 10  + '0';
        str[16] = 0;

        SIO_Init(S9600,0); // инициализация порта на скорости 9600
        Wsio('\n'); // команда перевода строки
        Wsio('\r'); // команда возврата каретки
        Type(str); // послать сформированную строку

        for (i=0; i<N; i++)
        {
           Wsio('\n');
           Wsio('\r');
                Type(&mas[i][0]);
                memset(str,' ',15);
                str[15] = 0;
                str[13] = ball[i] % 10  + '0';
                str[14] = 0;
                Type(str);
        }

}
```

8 Контрольные вопросы

1. Какие модули входят в состав учебного стенда SDK-1.1?
2. К какому семейству относится микроконтроллер ADuK812? Какой тип архитектуры? Основные характеристики этого микроконтроллера.
3. Показать визуально и рассказать о назначении всех узлов, переключателей и разъёмов стенда.
4. Какие виды памяти реализованы в стенде? Начертить карту памяти стенда и рассказать о каждом виде памяти стенда.
5. Опишите регистры ПЛИС: виды, назначение, адреса, примеры использования.
6. Как осуществляется доступ к регистрам ПЛИС?
7. Какие порты ввода-вывода есть в стенде? Каким образом осуществляется доступ к портам?
8. Сколько источников прерываний обеспечивает микроконтроллер ADuC812? Сколько уровней приоритетов прерываний? Как установить приоритет прерывания? Где хранятся адреса векторов прерываний?
9. Какие регистры предназначены для управления режимами прерываний?
10. Как используются прерывания в пользовательских программах?
11. Какие периферийные микросхемы входят в состав стенда?
12. ЖКИ – описание функций. Какие регистры имеет ЖКИ? Что хранится в этих регистрах?
13. Что представляет собой память данных ЖКИ? Что такое генератор символов ОЗУ?
14. Что представляет собой микросхема часы/календарь? Какая память в ней присутствует?
15. Какие основные узлы входят в состав микросхемы часов/календаря?
16. Что представляет собой устройство памяти часов/календаря?
17. Опишите основные функции, режим счета, режим сигнализации часов/календаря.
18. Что представляет собой регистр управления/состояния часов/календаря?
19. Опишите регистры-счетчики, регистр управления сигналом часов/календаря.
20. Какие вы знаете режимы работы таймера часов/календаря.
21. Сколько таймеров-счетчиков входит в состав стенда, опишите схемотехнику таймеров.
22. Какую информацию содержит регистр TCON?
23. Какую информацию содержит регистр TMOD?

24. Что такое резидентный загрузчик HEX202? Как осуществляется загрузка программы в память стенда SDK-1.1?
25. Что представляет собой формат записи Intel Hex?
26. Что такое инструментальная система T2? Назовите основные команды этой системы.
27. Что включает в себя набор средств тестирования стенда SDK-1.1?
28. Какие средства разработки для микроконтроллера 8051 включает в себя пакет Keil Software?
29. Назовите отличительные черты компилятора языка Си фирмы Keil Software.
30. Какие типы данных поддерживает компилятор C51?
31. Какие модификаторы памяти и модели памяти поддерживает компилятор C51?
32. Назовите основные функции, используемые при разработке программ.
33. Назовите этапы программирования и отладки стенда SDK-1.1.
34. Для чего используется память EEPROM в стенде SDK-1.1?
35. Что представляет собой интерфейс I2C?
36. Какие вы знаете средства ввода-вывода информации в стенде SDK-1.1?
37. Каким образом можно запрограммировать работу светодиодов?
38. Какие средства необходимо использовать для считывания клавиатуры стенда?
39. В чем заключается инициализация LCD стенда?
40. Как реализовать многозадачность в стенде SDK-1.1?
41. Что такое исключающая многозадачность?

Список использованных источников

1 Андреев, Д.В. Программирование микроконтроллеров MCS-51 : учебное пособие/ Д.Б. Андреев. - Ульяновск: УлГТУ, 2000. - 88с.
2 Бурькова, Е.В. Микропроцессорные системы: Электронное гиперссылочное учебное пособие / Е.В. Бурькова, А.Ю. Батов. Рег. № 90. - Оренбург: УФАП ГОУ ОГУ.-2005. – 112 с.
3 Бурькова, Е.В. Применение инструментального учебного микроконтроллера SDK-1.1 и программного симулятора при изучении дисциплины «Микропроцессорные системы» / Е.В. Бурькова // Современные информационные технологии в науке, образовании и практике: материалы Всероссийской научно-практической конференции. - Оренбург, ГОУ ОГУ, 2004. - С. 96-99.
4 Лукичев, А.Н. Расширение возможностей лабораторного комплекса SDK-1.1. Научно-технический вестник СПбГИТМО (ТУ). Выпуск 10. Информация и управление в технических системах. - СПб.: СПбГИТМО (ТУ), 2003.-С. 86-90.
5 Новиков, Ю.В. Основы микропроцессорной техники / Ю.В. Новиков, П.К. Скоробогатов - М.: ИНТУИТ.РУ. «Интернет-Университет Информационных технологий», 2003.-440с.
6 Новиков, Ю.В. Основы цифровой схемотехники. Базовые элементы и схемы. Методы проектирования / Ю.В. Новиков –М.: Мир 2001.–379 с.
7 Предко, М. Руководство по микроконтроллерам: в 2 т / М. Предко - М.: Постмаркет, 2001. – Т. 1 – 416 с.
8 Предко, М. Руководство по микроконтроллерам: в 2 т / М. Предко - М.: Постмаркет, 2001. – Т. 2 –488 с.
9 Пухальский, Г.И. Проектирование микропроцессорных устройств: учебное пособие для вузов / Г.И. Пухальский - СПб.: Политехника, 2001-544с.
10 Таненбаум, Э. Архитектура компьютера / Э. Таненбаум – СПб.: ПИТЕР, 2003. – 704 с.
11 Учебный стенд SDK-1.1 Руководство пользователя. - СПб.: ООО "ЛМТ", 2001.-99с.
12 Цилькер, Б.Я. Организация ЭВМ и систем: учебник для вузов / Б.Я. Цилькер, С.А. Орлов – СПб.: ПИТЕР, 2006. – 668 с.