

Кристиан Тавернье

РІС-МИКРОКОНТРОЛЛЕРЫ ПРАКТИКА ПРИМЕНЕНИЯ

**Архитектура и характеристики
микроконтроллеров**

**Средства разработки
приложений разного типа**

**Коллекция программных
и схемных решений**

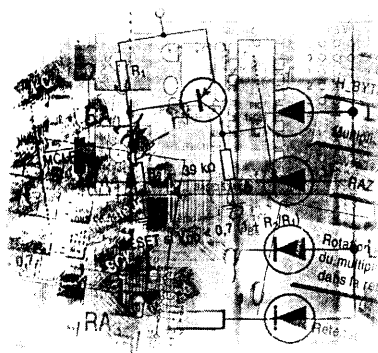
**Использование платы
Stamp фирмы Parallax**



CHRISTIAN TAVERNIER

LES MICROCONTRÔLEURS PIC

APPLICATIONS

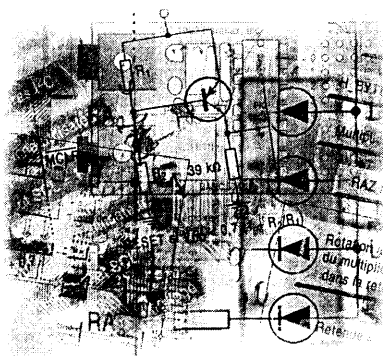


Серия «Справочник»

КРИСТИАН ТАВЕРНЬЕ

РIS- МИКРОКОНТРОЛЛЕРЫ

ПРАКТИКА ПРИМЕНЕНИЯ



ДМК
ИЗДАТЕЛЬСТВО

Москва, 2002

УДК 621.3.049.77

ББК 32.852

T13

Тавернье К.

T13 ПИС-микроконтроллеры. Практика применения: Пер. с фр. – М.: ДМК Пресс, 2002. – 272 с.: ил. (Серия «Справочник»).

ISBN 5-94074-115-0

В книге представлена информация о технических и программных средствах разработки приложений на базе ПИС-микроконтроллеров. Приведена коллекция схемных и программных решений, касающихся взаимодействия ПИС-микроконтроллеров с популярной периферией, реализации типовых интерфейсов, с которыми вы можете столкнуться в своих разработках. Рассмотрены многочисленные примеры программной реализации самых различных функций: организация прерываний, подпрограммы расширенной арифметики, арифметики с плавающей запятой и т.д.

В качестве примеров предлагаются несколько конкретных устройств, в том числе часы-будильник и многоканальный цифровой вольтметр.

Отдельная глава книги посвящена описанию и возможностям использования платы STAMP фирмы Parallax, которая построена на базе ПИС-микроконтроллера, программируемого на Basic и ориентированного на устройства автоматизации.

Книга адресована студентам, специалистам и любителям электроники, занимающимся разработкой микроконтроллерных устройств.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 2-10-002866-9 (фр.)

ISBN 5-94074-115-0 (рус.)

© DUNOD, Paris

© Перевод на русский язык, оформление. ДМК Пресс, 2002

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ	9
-------------------	---

ГЛАВА 1

РІС-МИКРОКОНТРОЛЛЕРЫ 16СХХ	11
МИКРОКОНТРОЛЛЕРЫ ПОДСЕМЕЙСТВА РІС 16СХХ	12
ОБЩИЕ ХАРАКТЕРИСТИКИ	13
Питание РІС-микроконтроллеров	13
Тактирование РІС-микроконтроллеров	13
Схемы сброса	17
Порты ввода/вывода	18
БАЗОВЫЕ СХЕМЫ	19

ГЛАВА 2

РАЗРАБОТКА ПРИЛОЖЕНИЙ	21
ВЫБОР МИКРОКОНТРОЛЛЕРА	22
АСЕМБЛЕР ИЛИ ЯЗЫКИ ВЫСОКОГО УРОВНЯ	23
СИСТЕМА РАЗРАБОТКИ	24
Ассемблер и компилятор языка высокого уровня	24
Эмулятор и симулятор	25
РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ПРОМЫШЛЕННОГО ПРИМЕНЕНИЯ	27
СРЕДСТВА РАЗРАБОТКИ ФИРМЫ MICROCHIP	28
Системы Picstart-16b и Picstart-16c	29
Программное обеспечение разработки	30

СИНТАКСИС АССЕМБЛЕРА MPALC	35
АССЕМБЛЕР PASM	37
СРЕДСТВА РАЗРАБОТКИ ФИРМЫ PARALLAX	39
Псевдоэмулятор Reflection-5X	46
Эмуляторы ClearView 5X и ClearView XX	48

ГЛАВА 3

СХЕМНЫЕ РЕШЕНИЯ

ИНТЕРФЕЙСОВ. МИКРОКОНТРОЛЛЕРОВ	49
ПАРАЛЛЕЛЬНЫЕ ВЫХОДЫ	50
Управление светодиодами и оптронами	50
Управление реле	54
Прямое управление нагрузкой, питающейся от источника постоянного напряжения	55
Управление светодиодным цифровым индикатором	56
Управление индикаторами на жидких кристаллах	69
ПАРАЛЛЕЛЬНЫЕ ВХОДЫ	86
Кнопки и переключатели	86
Гальваническая развязка входов	88
Клавиатуры	89
Вывод из sleep-режима с помощью клавиатуры	93
КОМБИНИРОВАННОЕ ИСПОЛЬЗОВАНИЕ ПОРТОВ	101
ВНЕШНЯЯ ПЕРИФЕРИЯ	110
Стандартная периферия	111
Взаимодействие с периферией по последовательному интерфейсу	113
ЭНЕРГОНЕЗАВИСИМАЯ ПАМЯТЬ С ПОСЛЕДОВАТЕЛЬНЫМ ИНТЕРФЕЙСОМ	114
УПРАВЛЕНИЕ АНАЛОГО-ЦИФРОВЫМ ПРЕОБРАЗОВАТЕЛЕМ	120
ЗАКЛЮЧЕНИЕ	122

ГЛАВА 4

БИБЛИОТЕКА ПРОГРАММ	123
АРИФМЕТИЧЕСКИЕ ПОДПРОГРАММЫ	124
Беззнаковое умножение 8-разрядных чисел	125
Знаковое и беззнаковое умножение 16-разрядных чисел	129

Деление 16-разрядных чисел	136
Сложение и вычитание 16-разрядных чисел	144
Операции с плавающей запятой	146
Преобразование двоично-десятичных кодов в двоичные	156
Преобразование двоичных кодов в двоично-десятичные	159
Сложение и вычитание чисел в двоично-десятичных кодах	163
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРЕРЫВАНИЙ МИКРОКОНТРОЛЛЕРОВ 16C5X	168
ПРИНЦИП МНОГОЗАДАЧНОСТИ	172
РАСШИРЕНИЕ СТЕКОВОЙ ПАМЯТИ МИКРОКОНТРОЛЛЕРОВ 16C5X	175
ПЕРЕДАЧА АСИНХРОННОЙ ПОСЛЕДОВАТЕЛЬНОСТИ ПРИ ОТСУТСТВИИ ПОСЛЕДОВАТЕЛЬНОГО ПОРТА	178

ГЛАВА 5

ГОТОВЫЕ РЕШЕНИЯ	185
ЧАСЫ С БУДИЛЬНИКОМ	186
РЕАЛИЗАЦИЯ ШИНЫ I ² C	215
Общие положения	216
Принцип обмена данными	216
Форматы передачи	219
Схема реализации интерфейса I ² C	220
ЧЕТЫРЕХКАНАЛЬНЫЙ ВОЛЬТМЕТР СО СВЕТОДИОДНОЙ ИНДИКАЦИЕЙ ...	232
МИКРОКОМПЬЮТЕР, ПРОГРАММИРУЕМЫЙ НА BASIC	244

ГЛАВА 6

МИКРОКОНТРОЛЛЕР STAMP	245
ОБЩИЕ ПОЛОЖЕНИЯ	246
СИСТЕМА РАЗРАБОТКИ	247
Схема платы Stamp	247
Программная среда разработки	250
ЯЗЫК ПРОГРАММИРОВАНИЯ PBASIC	250
Операторы управления портами ввода/вывода	252
Операторы для управления последовательным вводом/выводом	253
Управление аналоговыми величинами	253
Прочие операторы	253

ПРИМЕРЫ ПРИМЕНЕНИЙ	255
Аналого-цифровое преобразование	255
Управление шаговыми двигателями	259
Интерфейс клавиатуры и индикатора на жидких кристаллах	264
ПРИЛОЖЕНИЕ	269
ФАЙЛ ИДЕНТИФИКАТОРОВ	269

ПРЕДИСЛОВИЕ

Желание написать книгу, посвященную применению целого класса микроконтроллеров, может показаться высокомерным. Действительно, поскольку эти компоненты программируемы, они способны выполнять «все пожелания» пользователя, благодаря чему широко применяются в самых разных сферах. Какой бы объемной ни была книга, в ней невозможно рассказать обо всех реализуемых приложениях с такими микроконтроллерами. Поэтому мы решили написать не сборник готовых приложений, которые, как показывает опыт, редко отвечают потребностям большинства пользователей, а пособие, помогающее читателю самостоятельно разрабатывать собственные проекты.

Для того чтобы знакомство с предлагаемой информацией было наиболее эффективным, мы структурировали книгу следующим образом:

- ♦ в главе 1 коротко рассказано об архитектуре PIC-микроконтроллеров фирмы Microchip. Даны сравнительные характеристики микроконтроллеров семейства, приводятся схемные решения, касающиеся подключения питания, тактирования работы микроконтроллера внешними и внутренними генераторами, а также различных способов сброса;
- ♦ в главе 2 говорится о системах разработки приложений, о том, какой вариант выбрать в зависимости от имеющихся средств. Приводится описание ассемблера PIC-контроллеров, который необходимо знать, чтобы разобраться в текстах программ, представленных в последующих главах;
- ♦ в главе 3 на многочисленных примерах показано, как обеспечить взаимодействие микроконтроллеров с периферией (причем даны не только описания схем, но и тексты программ). Конечно, предлагаемые решения охватывают лишь малую часть возможных областей применения PIC-микроконтроллеров, однако знакомство с ними позволит вам быстро реализовать собственные приложения;

- ◆ в главе 4 представлена библиотека стандартных программ. Приведенные арифметические алгоритмы широко используются во многих приложениях. Их реализация, особенно на ассемблере, по плечу только опытным пользователям микроконтроллеров;
- ◆ в главе 5 описаны три актуальных приложения. Используемые в них решения не рассматривались в предыдущих главах;
- ◆ наконец, в главе 6 рассказывается об оригинальном продукте Stamp фирмы Parallax – программируемом PIC-микроконтроллере 16C56, интерпретирующем язык Basic, и дано несколько примеров его применения. Эта глава будет особенно интересна тем, кто предпочитает не пользоваться машинным языком.

ПРИНЯТЫЕ СОКРАЩЕНИЯ

Для удобства работы с книгой приняты следующие обозначения:

- ◆ наиболее важные понятия и основные термины выделены *курсивом*;
- ◆ листинги, а также названия регистров, переменных и т. д., встречающихся в тексте, набраны моноширинным шрифтом.

ГЛАВА 1

РІС-МИКРОКОНТРОЛЛЕРЫ 16СХХ

В этой главе:

Микроконтроллеры подсемейства PIC 16СХХ

Общие характеристики

Базовые схемы

Эта книга посвящена, главным образом, применению PIC-микроконтроллеров подсемейства 16CXX. Но чтобы дальнейшее изложение было понятно всем читателям, имеет смысл коротко напомнить об общих особенностях архитектуры семейства.

Детальное описание всех микросхем семейства вы сможете найти в каталогах фирмы Microchip и на ее сайте.

МИКРОКОНТРОЛЛЕРЫ ПОДСЕМЕЙСТВА PIC 16CX

В настоящее время существуют несколько подсемейств PIC-микроконтроллеров, каждое из которых включает множество различных типов. В данной книге рассматриваются два подсемейства, их состав и основные характеристики даны в табл. 1.1. Наиболее известно младшее подсемейство 16C5X, включающее архитектурно самые простые модели PIC-микроконтроллеров, которые используются в недорогих приборах, где они с успехом заменяют множество микросхем средней степени интеграции.

В состав данного подсемейства входят микроконтроллеры пяти типов, имеющие варианты по тактовой частоте и типу программного запоминающего устройства. Все микросхемы 16C5X имеют одну и ту же внутреннюю архитектуру, но отличаются объемом оперативной и программной памяти (*ОЗУ* и *ПЗУ*) и количеством портов ввода/вывода.

Второе подсемейство, занимающее верхнюю часть табл. 1.1, квалифицируется фирмой Microchip как среднее. В настоящее время в него входят четыре типа микросхем, но подсемейство развивается, и сейчас готовятся к выпуску новые типы.

По архитектурным концепциям и системе команд данное подсемейство очень близко к подсемейству 16C5X, но располагает более широкими внутренними аппаратными ресурсами. Микроконтроллеры данного подсемейства имеют, в частности, несколько последовательных интерфейсов, а также содержат многоканальные аналого-цифровые преобразователи.

Оба подсемейства имеют примерно одинаковую область применения. Однако каждое приложение, описанное в данной книге, ориентировано только на одно из подсемейств. Так, программная реализация последовательного асинхронного интерфейса более актуальна для подсемейства PIC 16C5X, которое не располагает данным аппаратным ресурсом, и совершенно не нужна, например, для PIC 16C74, имеющего встроенный высокоскоростной последовательный интерфейс.

ОБЩИЕ ХАРАКТЕРИСТИКИ

ПИС-микроконтроллеры подсемейства 16СХХ имеют сходные схемы подключения к питанию, тактирования и сброса.

Питание ПИС-микроконтроллеров

У всех ПИС-микроконтроллеров есть два вывода питания V_{SS} и V_{DD} :

- ♦ V_{SS} – общий («минус» питания);
- ♦ V_{DD} – «плюс» питания.

Для микроконтроллеров 16С5Х величина напряжения питания может составлять от +3 до +6,25 В. Для ПИС 16СХХ диапазон питающих напряжений не так широк: от +4 до +6 В.

Питание ПИС-микроконтроллеров любых типов, тактируемых от встроенного высокочастотного генератора (вариант HS), осуществляется напряжением от +4,5 до +5,5 В. В sleep-режиме внутреннее ОЗУ сохраняет свое содержимое при напряжении питания 1,5 В.

Чтобы устранить высокочастотные пульсации по питанию, применяют многослойный керамический конденсатор емкостью не менее 0,1 мкФ, который монтируется как можно ближе к корпусу микросхемы.

Если напряжение питания находится в пределах норм, установленных для ИС ТТЛ (от 4,75 до 5,25 В), то и уровни на входах/выходах ПИС-микроконтроллеров будут соответствовать уровням ТТЛ.

Тактирование ПИС-микроконтроллеров

У всех микросхем подсемейства ПИС 16СХХ имеются два вывода, предназначенные для тактирования их работы: OSC1 (CLKIN) и OSC2 (CLKOUT). Для тактирования могут использоваться генераторы нескольких типов.

В версиях микросхем ПИС 16СХХ с ультрафиолетовым и электрическим стиранием тип тактового генератора задается программированием соответствующих битов конфигурационного слова (особого регистра, доступного только во время программирования микросхемы). Для одной и той же микросхемы может быть задан или кварцевый, или RC-генератор.

Что касается однократно программируемых (ОТР – One-Time-Programmable) версий микросхем, то для них тип тактового генератора выбирается при заказе или покупке. В маркировке ПИС-микроконтроллеров 16СХХ версий ОТР тип генератора указывается

Таблица 1.1

Микроконтроллеры подсемейства PIC 16CXX

	Тактирование	Память					Периферия					Характеристики						
	Максимальная тактовая частота (МГц)	EPROM	Память программ		Память данных (байт)	EPROM данных (байт)	Таймеры	Захват/сравнение/ШИМ (PWM)	Последовательные интерфейсы (SPI/I ² C, SCI)	Параллельный служебный порт	Каналы АЦП (8 бит)	Внешнее прерывание	Источники прерывания	Входы/выходы	Напряжение питания (В)	Количество команд	Корпус	
ПЗУ			EEPROM															
PIC16C64	20	2К	-	-		128	TMR0, TMR1, TMR2	1	SPI/I ² C	ДА	-	ДА	8	33	2,5–6,0	35	40-pin DIP, 44-pin QFP	
PIC16C71	16	1К	-	-	36	-	TMR0	-	-	-	4	ДА	4	13	3,0–6,0	35	18-pin DIP, 18-pin SOIC	
PIC16C74	20	4К	-	-	192	-	TMR0, TMR1, TMR2	2	SPI/I ² C, SCI	ДА	8	ДА	12	33	2,5–6,0	35	40-pin DIP, 44-pin PLCC, 44-pin QFP	
PIC16C84	10	-	-	1К	36	64	TMR0	-	-	-	-	ДА	4	13	2,0–6,0	35	18-pin DIP, 18-pin SOIC, 20-pin SSOP	
PIC16C54	20	512	-	-	25	-	RTCC	-	-	-	-	-	0	12	2,5–6,25	33	18-pin DIP, 18-pin SOIC, 20-pin SSOP	

Таблица 1.1

Микроконтроллеры подсемейства PIC 16CXX (окончание)

	Тактирование	Память					Периферия					Характеристики						
		Макс. тактовая частота (МГц)	Память программ			Память данных (байт)	EPROM данных (байт)	Таймеры	Захват/сравнение/ШИМ (PWM)	Последовательные интерфейсы (SPI/I ² C, SCI)	Параллельный служебный порт	Каналы АЦП (8 бит)	Внешнее прерывание	Источники прерывания	Входы/выходы	Напряжение питания (В)	Количество команд	Корпус
			EPROM	ПЗУ	EEPROM													
PIC16C54A	20	512	-	-	25	-	RTCC	-	-	-	-	-	0	12	2,5-6,25	33	18-pin DIP, 18-pin SOIC, 20-pin QFP	
PIC16CR54	20	-	512	-	25	-	RTCC	-	-	-	-	-	0	12	2,5-6,25	33	18-pin DIP, 18-pin SOIC, 20-pin SSOP	
PIC16C55	20	1K	-	-	25	-	RTCC	-	-	-	-	-	0	20	2,5-6,25	33	28-pin DIP, 28-pin SOIC, 28-pin SSOP	
PIC16C56	20	1K	-	-	25	-	RTCC	-	-	-	-	-	0	12	2,5-6,25	33	28-pin DIP, 28-pin SOIC, 28-pin SSOP	
PIC16C57	20	2K	-	-	72	-	RTCC	-	-	-	-	-	0	20	2,5-6,25	33	28-pin DIP, 28-pin SOIC, 28-pin SSOP	
PIC16CR57A	20	-	2K	-	72	-	RTCC	-	-	-	-	-	0	20	2,5-6,25	33	28-pin DIP, 28-pin SOIC, 28-pin SSOP	
PIC16C58A	20	2K	-	-	72	-	RTCC	-	-	-	-	-	0	12	2,5-6,25	33	18-pin DIP, 18-pin SOIC, 20-pin SSOP	

в суффиксе. В зависимости от типа тактового генератора выделяют следующие версии PIC 16CXX:

- ◆ XT – с внутренним генератором на стандартном кварцевом резонаторе (согласно терминологии фирмы Microchip). Эта версия работает до максимальной частоты 4 МГц;
- ◆ HS – высокоскоростная версия (High Speed), которая также тактируется внутренним генератором с кварцевым резонатором, но способна работать на частотах до 20 МГц;
- ◆ RC – тактируется внутренним RC-генератором и способна работать на частотах до 4 МГц, но с меньшей стабильностью по частоте, чем версии с кварцевыми генераторами;
- ◆ LP – версия с малым потреблением (Low Power), тактирующаяся низкочастотным кварцевым генератором. У этой версии максимальная рабочая частота не превышает 200 кГц.

Для стабилизации частоты наряду с кварцевыми резонаторами возможно использование керамических резонаторов.

Существуют три схемы тактирования PIC-микроконтроллеров (рис. 1.1).

Для версий с кварцевым или керамическим резонатором используют схему, изображенную на рис. 1.1а. Значение конденсаторов C1 и C2 выбирается в зависимости от типа резонатора (кварцевый или керамический) и частоты (табл. 1.2). Для версии XT резистор R1 не нужен, однако иногда он требуется для микроконтроллеров версии HS. Только точное знание характеристик кварцевого резонатора позволяет определить, есть ли необходимость в резисторе R₁ и каким должно быть его значение.

Схема на рис. 1.1б представляет реализацию RC-генератора.

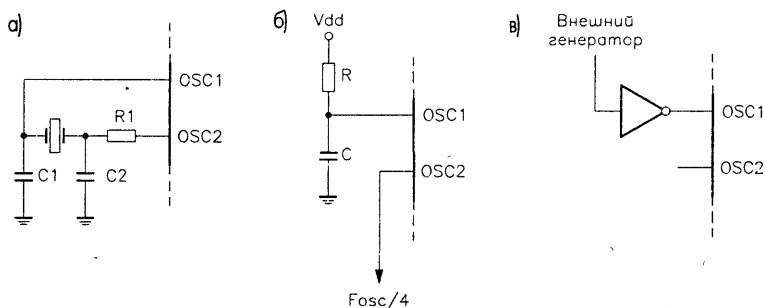


Рис. 1.1

Схемы тактовых генераторов для PIC 16CXX

Таблица 1.2

Номиналы компонентов для схем тактовых генераторов

Тип резонатора	Частота, кГц	C1, пФ	C2, пФ
LP	32*	15	15
	100	15	15
	200	0–15	0–15
XT	100	15–30	200–300
	200	15–30	100–200
	455	15–30	15–100
	1000	15–30	15–30
	2000	15	15
	4000	15	15
HS	4000	15	15
	8000	15	15
	20000	15	15

* Для $V_{cc} > 4,5$ В, $C1 = C2 = 30$ пФ

В этом случае для собственно генерации используется вывод OSC1. Вывод OSC2 является выходом внутренней рабочей частоты микроконтроллера (частоты командных циклов), которая в четыре раза меньше, чем частота генератора.

Стабильность RC-генератора не столь высока, как у кварцевого. Чтобы на нее не оказывали сильного влияния внешние факторы и внутренние характеристики самой микросхемы, фирма Microchip рекомендует применять резистор R с сопротивлением от 5 до 100 кОм и конденсатор емкостью не менее 20 пФ.

И наконец, схема на рис. 1.1в показывает способ тактирования PIC 16CXX внешним генератором. Очевидно, что формируемые внешним генератором уровни должны соответствовать напряжению питания микроконтроллера.

Схемы сброса

Все микроконтроллеры, в том числе и микроконтроллеры подсемейства 16CXX, имеют вывод сброса, называемый в данном случае MCLR. У PIC-микроконтроллеров предусмотрена внутренняя схема автоматического сброса при включении напряжения, она устойчиво работает, если скорость роста напряжения питания достаточно высока (обычно выше 0,05 В/мсек). Если напряжение питания при включении растет медленно, требуется внешняя схема сброса (так называемый ручной сброс), один из вариантов которой представлен на рис. 1.2а.

Внешняя схема сброса может потребоваться, если вы используете кварцевый резонатор относительно низкой частоты, с достаточно большим временем «разгона». В таком случае применяется схема, приведенная на рис. 1.2б. Эта схема известна пользователям и может применяться для микроконтроллеров, выпускаемых не только компанией Microchip, но и другими фирмами. Обратите внимание на резистор R1, значение которого может варьироваться от 100 Ом до 1кОм. Он служит для защиты входа MCLR микроконтроллера от положительного напряжения на конденсаторе C при выключении питания.

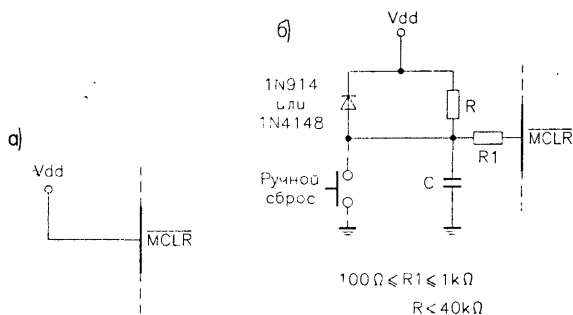


Рис. 1.2

Автоматический сброс
при включении питания
и ручной сброс

Наконец, если напряжение питания может снижаться до уровней, способных нарушить нормальную работу микроконтроллера, лучше использовать схему, инициирующую сброс, когда напряжение падает ниже определенного порога. Последние версии PIC-микроконтроллеров в отличие от ИС подсемейства 16CXX имеют для этого специальные аппаратные узлы. Две схемы внешнего сброса, предлагаемые фирмой Microchip, представлены на рис. 1.3. Схема со стабилитроном (рис. 1.3а), обладающая большей точностью, обеспечивает сброс, как только питание опускается ниже $V_Z + 0,7\text{ В}$, в то время как схема без стабилитрона (рис. 1.3б) — при снижении напряжения до уровня $0,7 (1 + R_2/R_1)$.

Порты ввода/вывода

Другие выводы микроконтроллеров 16CXX выполняют функции портов ввода/вывода, с помощью которых обеспечивается взаимодействие внутренних узлов с периферией. Особое внимание следует

обратить на согласование уровней сигналов микроконтроллера и внешних схем. Этой проблеме посвящена глава 3.

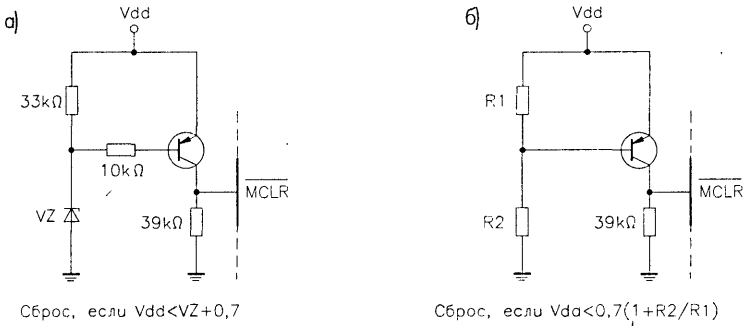


Рис. 1.3

Схемы автоматического сброса в случае снижения напряжения питания

БАЗОВЫЕ СХЕМЫ

Существует множество различных схем подключения PIC-микроконтроллеров, обеспечивающих их нормальное функционирование. Здесь приведены две базовые, которые используют схемные решения, представленные на рис. 1.1–1.3. На их основе вы сможете самостоятельно построить любые другие.

Первая из предлагаемых схем (рис. 1.4) встречается в недорогих переносных приборах. Она использует внутреннюю схему автоматического сброса и типовой RC-генератор.

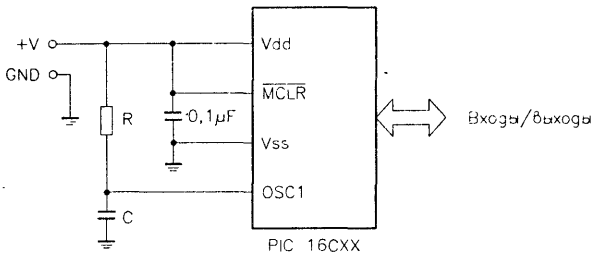


Рис. 1.4

Экономичный вариант подключения PIC-контроллеров

Вторую схему (рис. 1.5) можно рекомендовать, если вопросы цены и размеров не первостепенны. В ней используется кварцевый тактовый генератор, а схему сброса образует RC-цепочка, к которой при необходимости добавляется кнопка ручного сброса. Возможны и другие варианты в зависимости от предъявляемых к устройству требований. Например, если в вашем приложении нужен очень точный отсчет времени, не используйте тактовый RC-генератор. При нестабильном питании микроконтроллера лучше выбрать схему сброса, представленную на рис. 1.3. Это вопрос не техники, а элементарного здравого смысла.

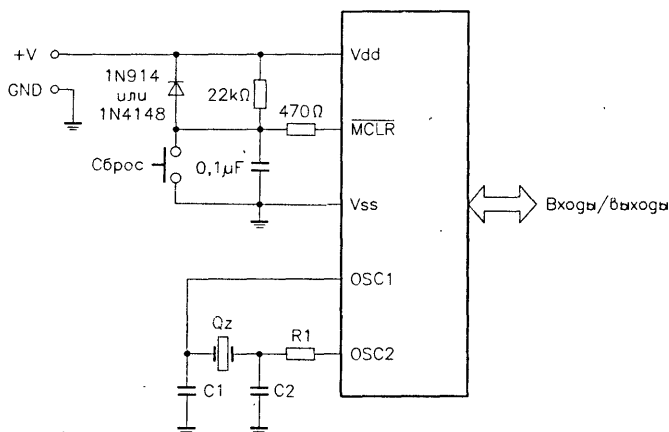


Рис. 1.5

Рекомендуемая схема подключения

Разработка приложений на базе микроконтроллеров мало отличается от создания приложений на основе микропроцессоров вообще.

При разработке таких приложений одним из главных вопросов является оптимальное разделение функций между аппаратурой и программным обеспечением.

В качестве классического примера, который позволяет хорошо понять суть этой проблемы, можно привести устройство управления цифровой индикацией. Практически всегда управление индикацией осуществляется с помощью техники мультиплексирования (*динамическая индикация*) или через последовательные интерфейсы, что позволяет сократить число линий связи, обеспечивающих управление индикатором. Чтобы упростить взаимодействие микроконтроллера с индикаторными устройствами, можно либо использовать внешние специализированные микросхемы, либо программно реализовать соответствующие интерфейсы.

Очевидно, что в первом случае усложняется аппаратура и упрощается программное обеспечение, так как управлять уплотнением каналов связи будет именно внешняя схема. Во втором случае оборудование, напротив, потребуется самое элементарное, зато программному обеспечению придется уделить гораздо больше внимания, причем программа связи должна обеспечивать необходимую скорость обмена.

Таким образом, прежде чем начинать разработку приложения с применением микроконтроллера, важно хорошо продумать варианты решений. Когда выбор сделан, можно переходить к подбору собственно микроконтроллера.

ВЫБОР МИКРОКОНТРОЛЛЕРА

При выборе микроконтроллера надо исходить из его функциональных возможностей и внутренней архитектуры (см. табл. 1.1).

Выбор зависит от наличия или отсутствия системы обеспечения разработки (программной среды разработки и соответствующего оборудования), а также финансовой базы, которой вы будете располагать – это может быть и масштаб промышленного производства, допускающий значительные затраты, и разработка единичных образцов, требующая минимальных средств.

Иногда лучше выбрать микроконтроллер, не содержащий всех необходимых элементов, рискнув добавить к нему затем одну или две внешние микросхемы. Нецелесообразно покупать один «супер-микроконтроллер», если нет средств на приобретение среды

разработки приложения или если капиталовложения, необходимые для создания приложения, не оправдываются его применением.

Когда выбор сделан, можно приступать к написанию и тестированию программного обеспечения. В этой главе речь идет о различных типах средств разработки: какими возможностями они обладают и каких результатов позволяют достичь. Кроме того, рассматривается синтаксис языков программирования. Знакомство с ними необходимо, чтобы свободно читать многочисленные листинги, приведенные в данной книге.

Сначала будет рассказано о наиболее популярных средствах разработки, обеспечивающих эффективную работу с PIC-микроконтроллерами. Описать все средства невозможно, поскольку PIC-микроконтроллеры используются практически повсеместно и множество фирм, иногда очень маленьких и мало знакомых широкой публике, предлагают продукты в этой области. Поэтому обзор ограничивается системами, предложенными фирмами Microchip и Rallex, которые располагают многочисленными, иногда весьма оригинальными продуктами.

АСЕМБЛЕР ИЛИ ЯЗЫКИ ВЫСОКОГО УРОВНЯ

Разработка любой компьютерной программы может быть осуществлена с помощью языка низкого уровня (машинного языка, ассемблера) или языка высокого уровня (C, Basic, Pascal). Для PIC-микроконтроллеров, и особенно входящих в подсемейство PIC 16CXX, ассемблер – самое лучшее решение, даже если он кажется более трудным в применении.

Действительно, размер памяти, доступный программе в микроконтроллерах этого подсемейства, относительно мал (от 512 до 4 К слов), что делает невозможным размещение большой программы. Программа, написанная на ассемблере, требует минимального объема памяти и позволяет реализовать максимальное количество функций. Напротив, если она создана на языке высокого уровня, то в памяти микроконтроллера может храниться код, соответствующий всего лишь нескольким десяткам или сотням операторов языка.

В таких языках, как C, Basic или Pascal, имеется множество типов переменных и обрабатывающих их операторов, что позволяет писать очень компактные исходные тексты программ. Но даже самый простой оператор на языке высокого уровня при компиляции

генерирует несколько десятков команд машинного языка, из-за чего доступная память заполняется очень быстро.

Язык высокого уровня целесообразно использовать для микроконтроллеров с большим объемом памяти, или в тех случаях, когда время, отведенное на разработку приложения, очень ограничено. Конечно, для этого необходимо знание соответствующего языка. Но лучше обратитесь к главам 3 и 4 настоящей книги, где описаны 35 команд PIC-микроконтроллеров и основы ассемблера. Вы убедитесь, что этих знаний вполне достаточно для написания программы.

СИСТЕМА РАЗРАБОТКИ

Некоторые понятия, относящиеся к системам разработки и соответствующему программному обеспечению, воспринимаются достаточно трудно, поэтому имеет смысл напомнить читателям основные термины.

Ассемблер и компилятор языка высокого уровня

Комплекс разработки всегда включает как минимум ассемблер и иногда один или несколько компиляторов языков высокого, которые используются для программирования.

Ассемблер переводит мнемокоды команд машинного языка в двоичные (бинарные) коды, исполняемые PIC-микроконтроллером. Текст с последовательностью мнемокодов называется листингом, или *исходным кодом* программы, в то время как бинарный код называется *объектным*, или *исполняемым*.

Компилятор переводит операции, операторы и другие конструкции языка высокого уровня, образующие исходный текст программы, в исполняемый бинарный код PIC-микроконтроллера.

В хорошо продуманной системе (среде) разработки обе программы, ассемблер и компилятор, могут сосуществовать отдельно или использоваться вместе. Это позволяет строить сложные алгоритмы на языке высокого уровня, а те программные модули, которые требуют высокой производительности или предназначены для управления периферией через порты ввода/вывода (*драйверы*), писать на ассемблере. Очень важно, чтобы компилятор языка высокого уровня допускал включение программных сегментов на ассемблере.

Эти обе программы, ассемблер и/или компилятор, должны обязательно «прогоняться» на так называемом *хост-компьютере*. В таком

качестве может выступать практически любая машина: специализированная система производителя PС-микроконтроллеров (все более и более редкий случай, так как это очень дорого), мощная ЭВМ (VAX, рабочая станция Sun, Apollo или HP и т.п.) или просто, и это становится правилом, IBM совместимый ПК. Последний вариант позволяет уменьшить капиталовложения, поскольку наверняка уже имеется IBM совместимый компьютер, используемый для других целей.

Написав программу на ассемблере или языке высокого уровня и проведя компиляцию, вы получаете исполняемый бинарный код. Однако прежде чем тиражировать созданное приложение, надо тщательно его протестировать. Чтобы обеспечить полный контроль, программа должна тестироваться в условиях, максимально близких к условиям будущего реального использования. Для этого существует несколько возможных решений.

Эмулятор и симулятор

Первое решение, которое является самым эффективным, но также, увы, и наиболее дорогостоящим, – приобрести специальную аппаратуру. На рис. 2.1 показано одно из таких устройств. Аппаратный *эмулятор*, составляющий «сердце» этой системы, фактически представляет собой специальное устройство, иногда очень сложное, которое выполняет все функции PС-микроконтроллера и заменяет его. На рисунке представлена «разрозненная» версия, то есть версия, в которой все внутренние компоненты PС-микроконтроллера реализованы в виде отдельных схем.

Этот эмулятор снабжен специальным соединительным шлейфом, называемым *зондом эмуляции*, на конце которого есть разъем, аналогичный выводам корпуса PС-микроконтроллера. С помощью шлейфа эмулятор подключается к макету приложения вместо настоящего PС-микроконтроллера.

Для подключения эмулятора к компьютеру обычно используется последовательный интерфейс RS232. Через него в эмулятор загружается программа микроконтроллера и осуществляется управление ее выполнением с помощью специальной компьютерной программы. Так как эмулятор – это «разрозненная» версия PС-микроконтроллера, вы имеете доступ к его различным внутренним узлам и шинам, в частности, можете узнать, по каким адресам «проходит» программа, какие коды записаны в управляющие регистры и т.п. В случае

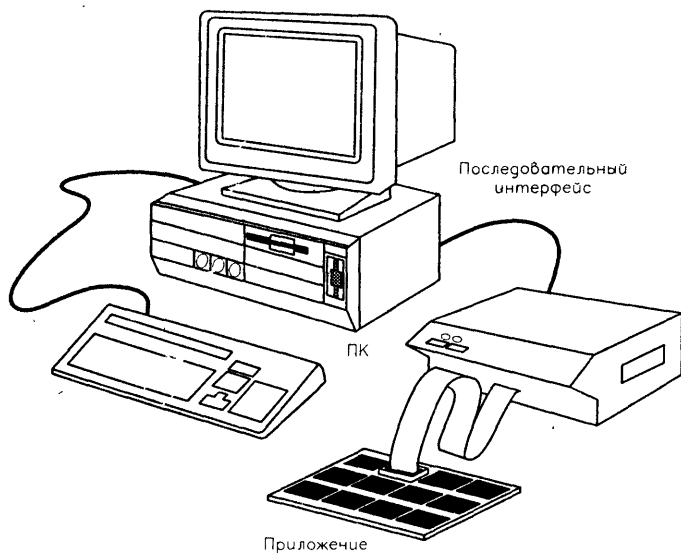


Рис. 2.1

Общий вид эмулятора

необходимости можно остановить исполняемую программу, чтобы считать состояние памяти или некоторых регистров.

Эмулятор обеспечивает работу приложения в реальном времени, поскольку способен функционировать на частоте микроконтроллера. Эмулятор – дорогостоящее промышленное средство, поэтому вряд ли имеет смысл покупать его для создания мелкосерийных или единичных приложений.

Второе возможное решение – использовать *симулятор*, но он не позволяет провести все необходимые тесты. Симулятор – это программа, написанная специально для микроконтроллера, работу которого она будет имитировать. Симулятор обычно функционирует на той же машине, где писалась программа приложения. На ее вход подается объектный (или исполняемый) код, который необходимо протестировать, и программа ведет себя так, как вел бы себя имитируемый микроконтроллер.

Симулятор значительно дешевле, чем эмулятор, ведь это только программа, сложность которой пропорциональна сложности микроконтроллера. Стоит он в десятки раз меньше, чем эмулятор.

Регистры и порты ввода/вывода микроконтроллера представляются переменными, хранящимися в памяти компьютера. Так,

8-разрядный параллельный порт представлен байтом данных. Достаточно считать эти данные, чтобы в любой момент знать о ходе выполнения программы и состоянии выходов. Ввод данных через порт можно имитировать записью соответствующих данных в память. Процедура моделирования операций обмена через порты ввода/вывода «растягивается» во времени, но если она хорошо проведена, то позволяет проверить 80% функций программы микроконтроллера.

Однако положение усложняется, когда необходимо учитывать временные характеристики сигналов, передаваемых через порты ввода/вывода. Симулятор – это программа, воспроизводящая лишь алгоритм работы микроконтроллера. Даже если использован высокопроизводительный компьютер, симулятор работает в десятки, а иногда и в сотни раз медленнее, чем та же программа, непосредственно выполняемая микроконтроллером.

Некоторые функции, связанные со временем, нельзя полностью смоделировать на симуляторе. Тем не менее, хорошее владение симулятором позволяет достаточно быстро и с минимальными капиталовложениями тестировать приложения.

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ПРОМЫШЛЕННОГО ПРИМЕНЕНИЯ

Разработку микроконтроллерных приложений для промышленного использования (о приложениях *ограниченного* применения будет сказано чуть ниже), целесообразно проводить следующим образом.

Сначала необходимо выбрать элементную базу разрабатываемого устройства, поскольку от этого зависит и качество аппаратуры (надежность, стабильность, быстродействие), и объем разрабатываемого программного обеспечения. Выбор элементной базы должен быть тщательно продуман и обоснован до начала разработки программного обеспечения, так как в некоторых случаях дальнейшие замены компонентов могут привести к полной переработке программы.

Как только выбор сделан, нужно определить структуру программного обеспечения. В результате этой работы может оказаться, что в элементную базу требуется внести какие-либо изменения. Допустим, составляя структуру и оценивая время, необходимое для выполнения различных программных функций, вы поняли, что выбранные компоненты (каналы передачи данных, например) не обеспечат необходимой скорости функционирования. Придется отступить на шаг назад, чтобы изменить некоторые компоненты.

Когда структура программного обеспечения составлена, приходит время написания программы. Чтобы облегчить последующее тестирование и отладку, нужно разделить ее на несколько модулей, каждый из которых выполняет определенные функции. Эти модули нужно писать поочередно и тестировать один за другим, что особенно важно для модулей драйверов, управляющих портами ввода/вывода. В идеале, каждым портом должен управлять соответствующий программный модуль.

Когда программное обеспечение написано, его необходимо протестировать. В зависимости от обстоятельств и специфики разрабатываемого устройства применяют симулятор или эмулятор. По завершении тестирования можно выполнить *натурное моделирование* работы устройства. Для этого предназначены перепрограммируемые версии микроконтроллера (EEPROM – с электрическим стиранием или UVPROM – с ультрафиолетовым стиранием), идентичные версии ОТР, которую вы сможете использовать позже, когда программное обеспечение будет доработано.

Именно перепрограммируемые версии следует использовать для проведения последних тестов устройства на макете или опытном образце. Это позволит в случае необходимости внести нужные изменения. Если все окажется правильным, переходите к программированию микросхем ОТР.

Как видите, этапы создания микроконтроллерных приложений и программного обеспечения классической информационной системы практически совпадают, но в первом случае некоторые из них оказываются более критичными из-за тесной взаимосвязи между элементной базой и программным обеспечением.

Общий алгоритм разработки приложения приведен на рис. 2.2.

СРЕДСТВА РАЗРАБОТКИ ФИРМЫ MICROCHIP

Тому, кто разрабатывает приложения на основе микроконтроллеров только от случая к случаю, не имеет смысла инвестировать средства в дорогостоящий эмулятор. Если речь идет о создании небольшого количества приложений с использованием микроконтроллеров PIC 16CXX, можно обойтись и без него благодаря комплектам для разработки, выпускаемым фирмами Microchip и Parallax. Такие комплекты содержат очень хороший ассемблер и достаточно эффективный симулятор.

Конечно, симулятор не имеет смысла полностью заменить эмулятор. Однако он позволяет любому внимательному разработчику

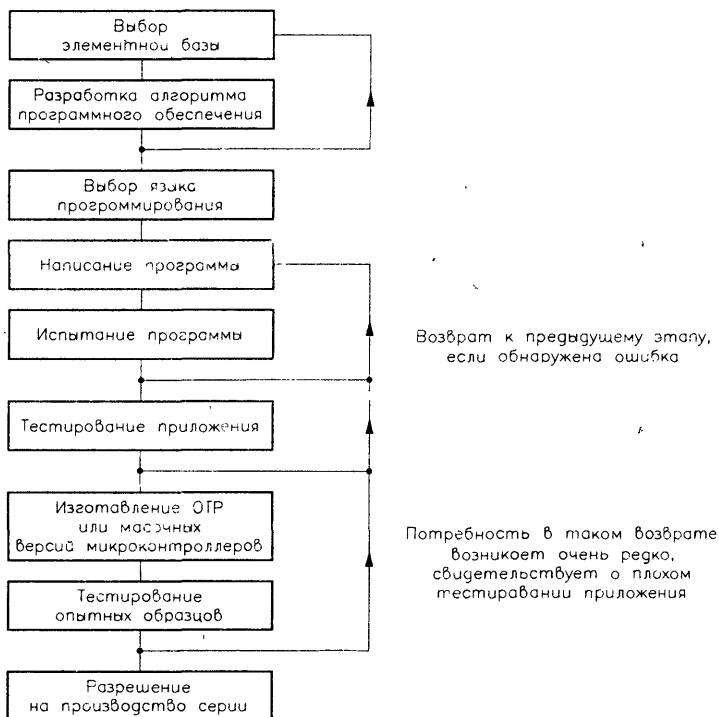


Рис. 2.2

Последовательность разработки приложения

достаточно точно протестировать свое приложение, особенно если окончательную доводку проводить с использованием микросхем версии UVPROМ (с ультрафиолетовым стиранием).

Системы Picstart-16b и Picstart-16c

Фирма Microchip предлагает различные средства разработки, среди которых – внутрисхемный эмулятор PICMASTER, способный поддерживать все микросхемы PIC 16CXX. Программное обеспечение эмулятора работает под ОС Windows. Этот продукт можно порекомендовать любому серьезному разработчику, который располагает достаточными средствами для его приобретения.

Есть и более приемлемые по цене продукты, которые позволяют успешно разрабатывать приложения на основе PIC 16CXX. Речь идет о системе разработки микроконтроллерных приложений, известной под названием PICSTART-16.

Когда писалась книга, этот продукт существовал в двух версиях:

- ♦ PICSTART-16B1 – позволяет осуществлять разработку приложений на основе PIC 16C5х, 16C61, 16C71 и 16C84;
- ♦ PICSTART-16C – поддерживает микроконтроллеры 16C64, 16C65, 16C73 и 16C74.

Эти системы отличаются только перечнем поддерживаемых микросхем, общие же их характеристики идентичны.

Несмотря на низкую цену, возможности PICSTART-16 способны удовлетворить самого требовательного пользователя. Обе системы включают:

- ♦ макроассемблер MPALC* для микроконтроллеров подсемейства PIC 16CXX;
- ♦ симулятор для указанных микросхем;
- ♦ плату программатора с соответствующим программным обеспечением;
- ♦ сетевой источник питания и кабель для подключения программатора к последовательному порту компьютера;
- ♦ образцы PIC-контроллеров подсемейства 16CXX (количество и типы меняются согласно комплекту и дате приобретения).

Все программное обеспечение может работать на любом IBM совместимом ПК и не очень требовательно к его ресурсам: достаточно 640 Кб оперативной памяти и операционной системы DOS версии 3.3 или выше.

В очень подробной инструкции по эксплуатации PICSTART-16 содержатся рекомендации по установке данного продукта на компьютере, а также описание программного обеспечения разработки, включающего макроассемблер и симулятор.

Что касается инструкций по применению макроассемблера и симулятора, вам надо будет их распечатать на принтере, так как они находятся в текстовом файле на прилагаемом диске.

Программное обеспечение разработки

Минимально необходимые возможности для разработки приложений предоставляют макроассемблер и симулятор, которые позволяют соответственно писать программы и тестировать их.

* По каталогу фирмы Microchip данный макроассемблер называется MPASM. – *Прим. ред.*

Ассемблер MPALC

Данный ассемблер – продукт очень высокого уровня, и это комплимент не новичка, поскольку автор практикует микропроцессорную технику уже более 15 лет! Помимо классических возможностей любого ассемблера MPALC поддерживает ряд макрокоманд и директив, например *условное ассемблирование*.

В ассемблере заложены модели всех микросхем серии PIC 16CXX, а также традиционные имена регистров, констант, переменных величин и т.д. Цифровые данные могут быть выражены во всех классических представлениях: бинарном, десятичном, восьмеричном, шестнадцатеричном и символьном (ASCII).

Он разрешает запись арифметических и логических выражений над данными и адресами, а также позволяет записывать операции отношения, сдвига вправо и влево.

Файлы с программами могут быть написаны в любом текстовом редакторе, единственное требование – они должны быть представлены в текстовом формате ASCII.

Используемый в MPALC синтаксис – классический для ассемблера, и даже менее консервативный, чем у аналогов. Допускается форматировать исходный текст с помощью табуляции. Идентификаторы способны включать до 32 знаков, причем различаются прописные и строчные буквы, но по желанию эту опцию можно отключить.

На выходе ассемблер создает различные файлы: файл для программирования микроконтроллера, файл листинга, файл сообщений об ошибках и предупреждений и, наконец, файл для симулятора.

Различные форматы файлов для программирования микроконтроллеров позволяют использовать различные программаторы, что соответствует требованиям рынка.

Возможности ассемблера расширяет поддержка макрокоманд. Что такое макрокоманда? Это выбранная вами часть программы, которая выполняет обычно одну часто используемую и хорошо отлаженную функцию, определенная как макрокоманда, названная оригинальным именем и включающая переменные величины. Однажды определенная, эта макрокоманда может быть помещена в любое место программы и интерпретирована ассемблером.

Наконец, условное ассемблирование позволяет, в зависимости от значения некоторых определенных вами параметров, откомпилировать ту или иную часть программы. Это очень интересная функция, хотя и недооцененная многими программистами. Благодаря

ей можно написать программу, общую для нескольких приложений, отличающихся только деталями, например для таймера с жидкокристаллической индикацией и таймера на светодиодах. В ходе компиляции параметры, определяющие условия ассемблирования, показываются в начале листинга, чтобы было ясно, для какого приложения предназначена программа.

Листинги некоторых программ, которые вы найдете в главах 3 и 4, были написаны именно для компоновки с помощью ассемблера MPALC.

Симулятор MPSIM

Симулятор помогает проверить на IBM совместимом ПК «поведение» вашей программы. Он имитирует работу выбранного микроконтроллера подсемейства PIC 16CXX и позволяет моделировать состояние его памяти, управляющих регистров и портов ввода/вывода в процессе выполнения микропрограммы.

Несмотря на то, что MPSIM работает в среде DOS и информация, которую он представляет разработчику в текстовой форме (рис. 2.3), выглядит малопривлекательно по сравнению с документами Windows, эффективность симулятора сомнений не вызывает. В верхней части окна выводится текущее содержимое регистровой памяти микроконтроллера, в том числе управляющих регистров. Эти данные можно редактировать.

Идентификация регистров осуществляется либо по их адресам, либо по именам, которые задаются в тексте программы. Последнее значительно упрощает отладку программы, поскольку определить назначение регистра проще по его имени, чем по обезличенному адресу. Можно отображать не только состояние регистров, но и отдельных битов. В рассматриваемом примере это показано для порта В, представленного бит за битом с RB7 по RB0. Более того, отдельные биты регистров могут идентифицироваться по именам.

Содержимое регистров будет меняться по мере трассировки имитируемой программы.

Нижняя часть экрана – это рабочее окно, где отображается листинг трассируемой (то есть выполняемой в пошаговом режиме) программы. Здесь же пользователь может набирать команды, необходимые для управления процессом отладки, и наблюдать их выполнение. Пример работы в режиме трассировки представлен на рис. 2.4. В правом верхнем углу экрана показывается теоретическое время выполнения программы и номер шага.

Симулятор MPSIM включает все функции, обычные для таких устройств: запуск симулируемой программы, трассировку программы с заданным шагом, установку точек прерывания, выполнение программ в режиме трассировки с прерываниями по условию, отображение содержимого памяти и управляющих регистров и т.д.

Он обеспечивает возможность редактирования отлаживаемой программы, оперативного изменения содержимого памяти, располагает функцией, позволяющей запоминать только что сделанные изменения таким образом, чтобы их можно было легко переносить потом в исходную программу.

Симулятор имитирует внешние сигналы, подаваемые на входы микроконтроллера. Осуществляется это с помощью специального файла, где хранится информация о том, какие сигналы должны подаваться на различные входы и в какие моменты времени. Данный файл можно сформировать во время работы симулятора. MPSIM поддерживает командный файл, в котором содержатся различные функции (команды) управления процессом симулирования, начальные условия, адреса точек прерываний и т.п. Наличие командного файла избавляет пользователя от необходимости при каждом новом цикле отладки программы повторно набирать все команды и условия. Вместо этого достаточно просто обратиться к командному файлу, и все загрузится автоматически. Напомним, что все файлы, с которыми работает симулятор, – стандартные текстовые файлы формата ASCII, и они могут быть прочитаны и отредактированы с помощью команд DOS или любого текстового редактора.

В заключение добавим, что результаты моделирования можно выводить в файл или на печать, можно вести журнал симулирования, дезассемблировать код, вести поиск данных в памяти и т.д.

Надеемся, что несмотря на краткость обзора возможностей симулятора MPSIM, вы убедились в его эффективности и целесообразности использования при отладке программного обеспечения.

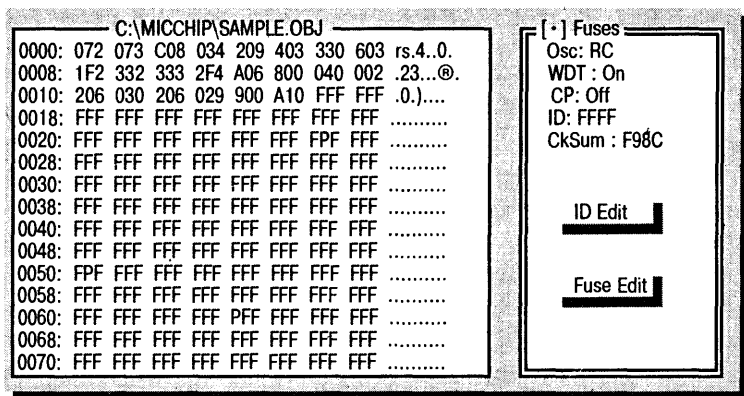
Программатор

Программатор – это последний элемент набора разработки программного обеспечения для микроконтроллеров, наиболее простой из трех представленных. Поддержка его работы осуществляется специальной программой, обеспечивающей развитый диалог в псевдографическом режиме, с развертывающимися меню и диалоговыми окнами (рис. 2.5).

Он обладает набором типичных для всех микроконтроллерных программаторов функций: выбор типа программируемой микросхемы, тест микросхемы «на чистоту», программирование, считывание и некоторые другие менее важные функции.

Аппаратную часть программатора образует плата, подключаемая к одному из четырех последовательных портов IBM, совместимого ПК. Связь с компьютером через порт тестируется до начала программирования.

PicStart File Windows Options Device Transfer Config P16C54 21:22:37



F1 Help Alt-X Exit F4 Edit F5 Program F6 Verify F7 Blank F8 Read

Рис. 2.5

Экран программатора фирмы Microchip

Работа данного программатора не требует особых комментариев. Отметим лишь, что он не предназначен для массового тиражирования микроконтроллеров.

СИНТАКСИС АССЕМБЛЕРА MPALC

Синтаксис, используемый ассемблером MPALC, достаточно традиционный и не должен вызывать трудностей у пользователей, уже работавших с машинным языком. Но, чтобы вы могли легко читать листинги, приведенные в следующих главах, ниже даются основные сведения об этом ассемблере. Детальное описание языка вы найдете в учебнике по MPALC.

Как и большинство ассемблеров, MPALC нечувствителен к управляющему символу «табуляция», и поэтому листинги можно табулировать как вам нравится. Ассемблер не различает прописные и строчные

буквы в командах и директивах. Они различаются только в идентификаторах (именах), если не отключена соответствующая опция.

Комментарии допускается размещать в любых строках программы. Им предшествует точка с запятой.

Идентификаторы могут включать от 1 до 32 знаков, но не должны использовать зарезервированные слова, являющиеся директивами и командами.

Цифровые данные могут быть выражены в десятичном, шестнадцатеричном, восьмеричном, двоичном и ASCII кодах. Типы используемых числовых данных указаны в табл. 2.1. В табл. 2.2 приведены математические операторы.

Таблица 2.1

Типы числовых данных, допускаемые MPALC

Формат данных	Способ записи	Пример
Десятичный	Данные	100
	.Данные	.100
Шестнадцатеричный	H'Данные'	H'9f'
	Данные h	900h
	0xДанные	0x9ff
Восьмеричный	O'Данные'	O'777'
	Q'Данные'	Q'777'
	Данные O	777O
	Данные Q	777Q
	\ Данные	\ 777
Двоичный	B'Данные'	B'0011101'
	Данные B	0011101B
ASCII	'Данные'	'C'
	A'Данные'	A'C'

Чаще других употребляются следующие директивы ассемблера:

- ◆ DATA – инициализирует несколько слов, хранящихся в памяти;
- ◆ ZERO – заполняет нулями область памяти, размер которой задается числом, следующим за директивой;
- ◆ SET – определяет значение переменной;
- ◆ RES – резервирует память, объем которой указан следующим за директивой числом;
- ◆ EQU – определяет константу в программе;
- ◆ INCLUDE – позволяет вставить файл внутрь текущей программы;
- ◆ IF – определяет начало кодового раздела условного ассемблирования;
- ◆ ELSE – определяет начало альтернативного кодового раздела условного ассемблирования;

- ◆ **ENDIF** – заканчивает кодовый раздел условного ассемблирования;
- ◆ **ORG** – определяет абсолютный адрес кода;
- ◆ **END** – заканчивает исходный текст;
- ◆ **MACRO** – определяет начало кода макрокоманды;
- ◆ **ENDM** – определяет конец кода макрокоманды;
- ◆ **LOCAL** – определяет локальную переменную в макрокоманде;
- ◆ **EXITM** – выход из макрокоманды.

Есть и другие директивы, относящиеся к форматированию или нумерации страниц. Но поскольку они не влияют на понимание листингов программ, представленных в следующих главах, в этой книге описываться не будут.

Таблица 2.2

Математические операторы

Оператор	Функция
+	Сложение
-	Вычитание
-	Изменение знака
%	Деление по модулю
*	Умножение
/	Деление
<<	Сдвиг влево
>>	Сдвиг вправо
()	Скобки
=	Равно
!=	Не равно
<=	Меньше или равно
>=	Больше или равно
!	Логическое отрицание
1	Поразрядное ИЛИ
&	Поразрядное И
	Логическое ИЛИ
&&	Логическое И
^	Поразрядное исключающее ИЛИ
\$	Текущее значение программного счетчика

АСЕМБЛЕР PASM

Микроконтроллеры семейства MCS51 фирмы Intel очень распространены, и их ассемблер известен многим программистам. В связи с этим американская фирма Parallax создала ассемблер под названием PASM, который поддерживает набор команд PIC-микроконтроллера и псевдоинструкций, аналогичных инструкциям 8051.

Так, вы можете записать в PASM:

```
ADD A, B
```

Что будет переведено им в:

```
MOVF B, 0
```

```
ADDWF A, 1
```

Для тех, кто уже работал с ассемблером 8051, первая строка будет более привычной, чем две эквивалентные команды для PIC-микроконтроллеров.

Этот ассемблер и его псевдоинструкции микроконтроллера 8051 используются в листингах нескольких приложений, представленных в данной книге. Поэтому в табл. 2.3 приведены правила перевода псевдоинструкций ИС 8051 в инструкции PIC-микроконтроллеров. Как можно заметить, эти правила естественно вытекают из функций, соответствующих инструкций.

Синтаксис этого ассемблера подобен синтаксису MPALC, но имеет несколько отличий.

PASM не поддерживает ни макрокоманды, ни условное ассемблирование, и соответствующих директив, естественно, не существует. Отсутствует восьмеричное представление данных. Что касается операторов, то они не столь многочисленны, как видно из табл. 2.4.

Директив ассемблера также меньше:

- ◆ DS – заполняет заданное число ячеек памяти указанным значением;
- ◆ EQU – определяет константу в программе;
- ◆ = – выполняет ту же функцию, что и EQU;
- ◆ INCLUDE – позволяет включить файл в текущую программу;
- ◆ ORG – определяет абсолютный адрес для кода;
- ◆ END – заканчивает текст программы;
- ◆ DEVICE – определяет тип используемого PIC-микроконтроллера;
- ◆ RESET – определяет адрес сброса.

Однако есть у ассемблера PASM и преимущества – он может определять идентификаторы для битов. Так, запись PORTC.3 будет интерпретирована как «бит 3 регистра PORTC».

Кроме того, в PASM имеется несколько стандартных *логических* имен регистров PIC-микроконтроллера и возможность вызывать файл со стандартными именами регистров посредством директивы INCLUDE. Таким образом, RA соответствует порту A, RB – порту B и т. д. Чтение листингов, представленных в этой книге и написанных на данном ассемблере, не вызовет у вас трудностей.

СРЕДСТВА РАЗРАБОТКИ ФИРМЫ PARALLAX

Фирма Parallax много и успешно занимается выпуском средств разработки микроконтроллеров. В этом разделе будет рассказано о наиболее интересных из предлагаемых продуктов.

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера

Псевдокоманда 8051	Эквивалент PIC
ADD fr, #literal	MOVLWF literal ADDWF fr,1
ADD fra,frb	MOVF frb,0
ADD fr,W	ADDWF fra,1
ADD W,Fr	ADDWF fr,0
ADDB fr,bit	BTFSC bit incf fr,1
AND fr, #literal	MOVLW literal ANDWF fr,1
AND fra,frb	MOVF frb,0 ANDWF fra,1
AND fr,W	ANDWF fr,1
AND W, #literal	ANDLW literal
AND W,fr	ANDWF fr,0
CALL addr	CALL adr8
CJA fr, #literal, addr9	MOVLW literal^0FFh ADDWF fr,0 BTFSC 3,0 GOTO addr9
CJA fr1, fr2, addr9	MOVF fr1,0 SUBWF fr2,0 BTFSC 3,0 GOTO addr9
CJAE fr, #literal, addr9	MOVLW literal SUBWF fr,0 BTFSC 3,0 GOTO addr9

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (продолжение)

Псевдокоманда 8051	Эквивалент PIC
CJAE fr1,fr2,addr9	MOVF fr2,0 SUBWF fr1,0 BTFSC 3,0 GOTO addr9
CJB fr,#literal,addr9	MOVLW literal SUBWF fr,0 BTFSS 3,0 GOTO addr9
CJB fr1,fr2,addr9	MOVF fr2,0 SUBWF fr1,0 BTFSS 3,0 GOTO addr9
CJBE fr,#literal,addr9	MOVLW /literal ADDWF fr,0 BTFSS 3,0 GOTO addr9
CJBE fr1,fr2,addr9	MOVF fr1,0 SUBWF fr2,0 BTFSC 3,0 GOTO addr9
CJE fr,#literal,addr9	MOVLW literal SUBWF fr,0 BTFSC 3,2 GOTO addr9
CJE fr1,fr2,addr9	MOVF fr2,0 SUBWF fr1,0 BTFSS 3,2 GOTO addr9
CJNE fr,#literal,addr9	MOVLW literal SUBWF fr,0 BTFSS 3,2 GOTO addr9

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (продолжение)

Псевдокоманда 8051	Эквивалент PIC
CJNE fr1,fr2,addr9	MOVF fr2,0 SUBWF fr1,0 BTFSS 3,2 GOTO addr9
CLC	BCF 3,0
CLR fr	CLRF fr
CLR W	CLRW
CLR WDT	CLRWDT
CLRB bit	BCF bit
CLZ	BCF 3,2
CSA fr,#literal	MOVLW /literal ADDWF fr,0 BTFSS 3,0
CSA fr1,fr2	MOVF fr1,0 SUBWF fr2,0 BTFSC 3,0
CSAE fr,#literal	MOVLW literal SUBWF fr,0 BTFSS 3,0
CSAE fr1,fr2	MOVF fr2,0 SUBWF fr1,0 BTFSS 3,0
CSB fr,#literal	MOVLW literal SUBWF fr,0 BTFSC 3,0
CSB fr1,fr2	MOVF fr2,0 SUBWF fr1,0 BTFSC 3,0
CSBE fr,#literal	MOVLW /literal ADDWF fr,0 BTFSC 3,0

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (продолжение)

Псевдокоманда 8051	Эквивалент PIC
CSBE fr1,fr2	MOVF fr1,0 SUBWF fr2,0 BTFSS 3,0
CSE fr,#literal	MOVLW literal SUBWF fr,0 BTFSS 3,2
CSE fr1,fr2	MOVF fr2,0 SUBWF fr1,0 BTFSS 3,2
CSNE fr,#literal	MOVLW literal SUBWF fr,0 BTFSC 3,2
CSNE fr1,fr2	MOVF fr2,0 SUBWF fr1,0 BTFSC 3,2
DEC fr	DECf fr,1
DECSZ fr	DECFSZ fr,1
✓ DJNZ fr,addr9	DECFSZ fr,1 GOTO addr9
IJNZ fr,addr9	INCFSZ fr,1 GOTO addr9
INC fr	INCF fr,1
INCSZ fr	INCFSZ fr,1
JB bit,addr9	BTFSC bit GOTO addr9
JC addr9	BTFSC 3,0 GOTO addr9
JMP addr9	GOTO addr9
JMP PC+W	ADDWF 2,1
JMP W	MOVWF 2
JNB bit,addr9	BTFSS bit GOTO addr9

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (продолжение)

Псевдокоманда 8051	Эквивалент PIC
JNC addr9	BTFSS 3,0 GOTO addr9
JNZ addr9	BTFSS 3,2 GOTO addr9
JZ addr9	BTFSC 3,2 GOTO addr9
MOV fr,#literal	MOVLW literal MOVWF fr
MOV fr1,fr2	MOVF fr2,0 MOVWF fr1
MOV fr,W	MOVWF fr
MOV OPTION,#literal	MOVLW literal OPTION
MOV OPTION,fr	MOVF fr,0 OPTION MOV OPTION,W
MOV !port_fr,#literal	MOVLW literal OPTION TRIS port_fr
MOV !port_fr,fr	MOVF fr,0 TRIS port_fr
MOV !port_fr,W	TRIS port_fr
MOV W,#literal	MOVLW literal
MOV W,fr	MOVF fr,0
MOV W,/fr	COMF fr,0
MOV W,fr-W	SUBWF fr,0
MOV W,++fr	INCF fr,0
MOV W,-fr	DECF fr,0
MOV W,<<fr	RLF fr,0
MOV W,>>fr	RRF fr,0
MOV W,<>fr	SWAPF fr,0

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (продолжение)

Псевдокоманда 8051	Эквивалент PIC
MOVB bit1,bit2	BTFSS bit2 BCF bit1 BTFSC bit2 BSF bit1
MOVB bit1,/bit2	BTFSC bit2 BCF bit1 BTFSS bit2 BSF bit1
MOVSZ W,++fr	INCFSZ fr,0
MOVSZ W,-fr	DECFSZ fr,0
NEG fr	COMF fr,1 INCF fr,1
NOP	NOP
NOT fr	COMF fr,1
NOT W	XORLW OFFh
OR fr,#literal	MOVLW literal IORWF fr,1
OR fr1,fr2	MOVF fr2,0 IORWF fr1,1
OR fr,W	IORWF fr,1
OR W,#literal	IORLW literal
OR W,fr	IORWF fr,0
RET	RETLW 0
RETW literal1, literal2,...	RETLW literal1 RETLW literal2 ...
RL fr	RLF fr,1
RR fr	RRF fr,1
SB bit	BTFSS bit
SC	BTFSS 3,0
SETB bit	BSF bit

Таблица 2.3

(соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (окончание))

Псевдокоманда 8051	Эквивалент PIC
SKIP	BTFSS 4,7
SLEEP	SLEEP
SNB bit	BTFSC bit
SNC	BTFSC 3,0
SNZ	BTFSC 3,2
STC	BSF 3,0
STZ	BSF 3,2
SUB fr,#literal	MOVLW literal SUBWF fr,1
SUB fr1,fr2	MOVF fr2,0 SUBWF fr1,1
SUB fr,W	SUBWF fr,1
SUBB fr,bit	BTFSS 3,0 DECF fr,1
SWAP fr	SWAPF fr,1 BTFSS 3,2
TEST fr	MOVF fr,1
TEST W	IORLW 0
XOR fr,#literal	MOVLW literal XORWF fr,1
XOR fr1,fr2	MOVF fr2,0 XORWF fr1,1
XOR fr,W	XORWF fr,1
XOR W,#literal	XORLW literal
XOR w,fr	XORWF fr,0

Таблица 2.4

Операторы ассемблера фирмы Parallax

Оператор	Функция
+	Сложение
-	Вычитание
-	Изменение знака
*	Умножение
/	Деление
<<	Сдвиг влево
>>	Сдвиг вправо
&	Логическое И
	Логическое ИЛИ
^	Исключающее ИЛИ
<	Старший байт
>	Младший байт
.	Адрес бита

Псевдоэмулятор Reflection-5X

В каталоге фирмы Parallax представлен очень интересный симулятор Reflection-5x для микроконтроллеров семейства PIC 16CXX, который способен имитировать работу портов ввода/вывода.

Это устройство, как показано на рис. 2.6, подсоединяется к отлаживаемой схеме как эмулятор, то есть заменяет микроконтроллер в вашем приложении. Псевдоэмулятор управляется через последовательный порт IBM совместимого ПК (COM1 или COM2) и поддерживается специальным программным обеспечением.

Необходимо помнить, что Reflection-5x все же симулятор и не может работать в реальном времени. Тем не менее он позволяет успешно проводить тестирование аппаратуры приложения.

Симулятор Reflection-5x питается от сетевого адаптера, дающего постоянное стабилизированное напряжение от 9 до 12 В при токе в 250 мА. Два плоских кабеля, с 18- и 28-контактными разъемами, позволяют подключать его вместо PIC-микроконтроллера. Благодаря DIP-переключателю устройства можно подключить выводы MCLR и RTCC либо к положительному полюсу питания вашего приложения, либо к аналогичной цепи симулятора Reflection-5x. Связь с ПК осуществляется через выбираемый вами последовательный порт с помощью входящего в комплект устройства кабеля. При этом программное обеспечение Reflection-5x автоматически обнаруживает

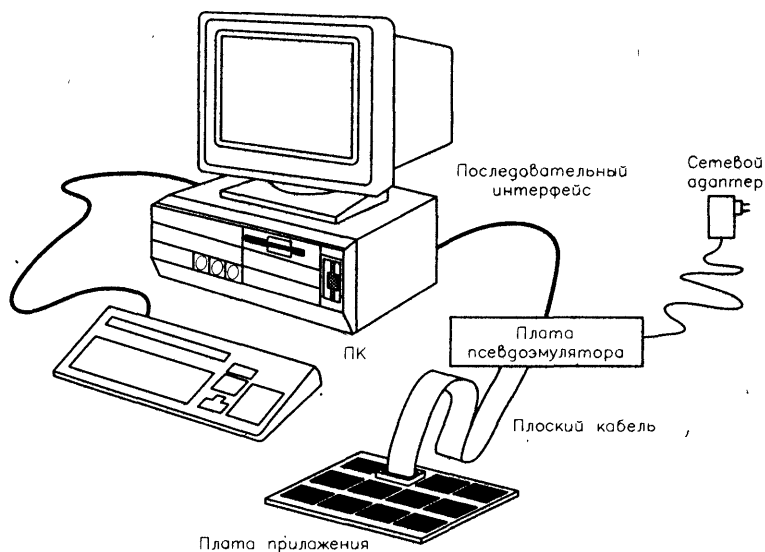


Рис. 2.6

Подключение симулятора Reflection-5x

и настраивает порт. Обмен осуществляется на максимальной скорости передачи порта.

Программное обеспечение симулятора использует для диалога трехоконную конфигурацию экрана. Верхнюю правую часть занимает фрейм, где показано содержимое оперативной памяти микроконтроллера; в центре и справа отображается содержимое управляющих регистров; в нижней части окна вы можете контролировать ход моделирования по исходному листингу.

Симулятор позволяет вводить неограниченное число точек прерывания, допускает пошаговое выполнение программы, прогон от одного прерывания до следующего. Он разрешает оперативно изменять содержимое любого управляющего регистра или ячейки памяти для того, чтобы наблюдать влияние этого на ход программы. Особо стоит рассказать о двух опциях.

Одна из них – выполнение программы до следующей строки программы. Она позволяет при работе в пошаговом режиме, например, за один шаг выполнять всю подпрограмму.

В любой момент можно увидеть точное реальное время выполнения всей программы или любой ее части.

Вы можете задать частоту тактового генератора вашего приложения. Это, конечно, не оказывает никакого влияния на скорость

моделирования, но учитывается программным обеспечением. Необходимо отметить гибкость установки точек прерываний и модификации регистров. Достаточно навести курсор на строку, которую вы хотите определить как точку прерывания, или на регистр, который собираетесь изменить, и впечатать новое значение. Не требуется запоминать адрес прерывания, все видно на экране.

Наконец, довольно редкая, но очень полезная опция: симулятор может переходить назад на величину от одного до ста шагов. Это позволяет, например, увидеть, каким было состояние приложения до остановки программы!

Эмуляторы ClearView 5X и ClearView XX

Эмулятор ClearView 5X предназначен для группы микроконтроллеров PIC 16C5X, в то время как ClearView XX – для остальной части подсемейства PIC 16CXX. Такое деление неудобно, так как вынуждает приобретать два отдельных устройства, чтобы обеспечить работу со всеми микроконтроллерами подсемейства PIC 16CXX.

В том, что касается аппаратуры, данные эмуляторы сходны с симулятором Reflection-5x. Отличие в том, что у эмуляторов ClearView 5X и XX имеется тактовый генератор, способный заменить, в случае необходимости, тактовый генератор приложения. Включается он двумя DIP-переключателями. Сетевой адаптер обоих устройств имеет небольшие размеры и обеспечивает напряжение питания от 7 до 9 В при токе до 1 А. Подключение к приложению осуществляется в соответствии с рис. 2.6, поскольку это настоящие эмуляторы.

Что касается их программного обеспечения, то все сказанное по поводу Reflection-5x действительно и для них. Но поскольку ClearView 5X и XX являются эмуляторами, они работают в реальном времени. Поэтому, если вы, например, тестируете свою программу и в ходе теста осуществляете прогон программы от одного прерывания до другого, он будет происходить с той же скоростью, как если бы на вашем приложении стоял микроконтроллер.

Это остается действительным до частоты тактового генератора 20 МГц для ClearView 5X и до 10 МГц – для ClearView XX.

Естественно, в реальном времени можно имитировать и функционирование портов ввода/вывода, что свойственно только настоящему эмулятору.

Большинство устройств на основе микроконтроллеров соединяется с внешними устройствами при помощи портов ввода/вывода. Их назначение может быть различным и определяется конкретным применением микроконтроллера. Так, если ваш микроконтроллер – автомат, управляющий посудомоечной машиной, на его входы будут подаваться сигналы с различных датчиков, а его выходы – управлять двигателями и электромагнитными клапанами. Если же речь идет о программируемом термостате, то порты ввода/вывода микросхемы будут взаимодействовать с клавиатурой, цифровым индикатором, температурным зондом и системой управления нагревательного устройства.

Разнообразие применений портов ввода/вывода не должно вас пугать, поскольку в большинстве случаев в их основе лежит несколько базовых схем, которые и описаны в этой главе.

Ни одно из представленных здесь решений не претендует на звание единственно возможного или самого лучшего. Преимущество микроконтроллеров именно в том и проявляется, что они позволяют получить один и тот же результат разными способами. Отталкиваясь от представленных здесь примеров, вы должны сами определить, какой вариант лучше всего подходит для того или иного приложения.

ПАРАЛЛЕЛЬНЫЕ ВЫХОДЫ

Речь идет о наиболее простом способе использования портов ввода/вывода, как по числу необходимых компонентов, так и по сложности соответствующего программного обеспечения. И именно в устройствах с микроконтроллерами такой вид взаимодействия аппаратурных узлов применяют чаще всего. Поэтому мы с него и начнем.

В качестве выходов параллельные порты могут применяться для управления реле, симисторами, светодиодными или иными цифровыми индикаторами.

Управление светодиодами и оптронами

Управление светодиодами – самое простое, что может встретиться на практике. Схемно оно может немного изменяться в зависимости от используемого микроконтроллера. У некоторых микроконтроллеров выходы рассчитаны на ток большой силы, а поэтому светодиод может быть подключен к ним непосредственно через ограничивающий ток резистор. В качестве примера приведем схему управления четырьмя светодиодами через порт APIC-микроконтроллера 16C5X

Г Л А В А 2

РАЗРАБОТКА ПРИЛОЖЕНИЙ

В этой главе:

Выбор микроконтроллера

Ассемблер или языки высокого уровня

Система разработки

Разработка программного обеспечения
для промышленного применения

Средства разработки фирмы Microchip

Синтаксис ассемблера MPALC.

Ассемблер PASM

Средства разработки фирмы Parallax

Разработка приложений на базе микроконтроллеров мало отличается от создания приложений на основе микропроцессоров вообще.

При разработке таких приложений одним из главных вопросов является оптимальное разделение функций между аппаратурой и программным обеспечением.

В качестве классического примера, который позволяет хорошо понять суть этой проблемы, можно привести устройство управления цифровой индикацией. Практически всегда управление индикацией осуществляется с помощью техники мультиплексирования (*динамическая индикация*) или через последовательные интерфейсы, что позволяет сократить число линий связи, обеспечивающих управление индикатором. Чтобы упростить взаимодействие микроконтроллера с индикаторными устройствами, можно либо использовать внешние специализированные микросхемы, либо программно реализовать соответствующие интерфейсы.

Очевидно, что в первом случае усложняется аппаратура и упрощается программное обеспечение, так как управлять уплотнением каналов связи будет именно внешняя схема. Во втором случае оборудование, напротив, потребуется самое элементарное, зато программному обеспечению придется уделить гораздо больше внимания, причем программа связи должна обеспечивать необходимую скорость обмена.

Таким образом, прежде чем начинать разработку приложения с применением микроконтроллера, важно хорошо продумать варианты решений. Когда выбор сделан, можно переходить к подбору собственно микроконтроллера.

ВЫБОР МИКРОКОНТРОЛЛЕРА

При выборе микроконтроллера надо исходить из его функциональных возможностей и внутренней архитектуры (см. табл. 1.1).

Выбор зависит от наличия или отсутствия системы обеспечения разработки (программной среды разработки и соответствующего оборудования), а также финансовой базы, которой вы будете располагать – это может быть и масштаб промышленного производства, допускающий значительные затраты, и разработка единичных образцов, требующая минимальных средств.

Иногда лучше выбрать микроконтроллер, не содержащий всех необходимых элементов, рискуя добавить к нему затем одну или две внешние микросхемы. Нецелесообразно покупать один «супер-микроконтроллер», если нет средств на приобретение среды

разработки приложения или если капиталовложения, необходимые для создания приложения, не оправдываются его применением.

Когда выбор сделан, можно приступать к написанию и тестированию программного обеспечения. В этой главе речь идет о различных типах средств разработки: какими возможностями они обладают и каких результатов позволяют достичь. Кроме того, рассматривается синтаксис языков программирования. Знакомство с ними необходимо, чтобы свободно читать многочисленные листинги, приведенные в данной книге.

Сначала будет рассказано о наиболее популярных средствах разработки, обеспечивающих эффективную работу с PIC-микроконтроллерами. Описать все средства невозможно, поскольку PIC-микроконтроллеры используются практически повсеместно и множество фирм, иногда очень маленьких и мало знакомых широкой публике, предлагают продукты в этой области. Поэтому обзор ограничивается системами, предложенными фирмами Microchip и Rallex, которые располагают многочисленными, иногда весьма оригинальными продуктами.

АСЕМБЛЕР ИЛИ ЯЗЫКИ ВЫСОКОГО УРОВНЯ

Разработка любой компьютерной программы может быть осуществлена с помощью языка низкого уровня (машинного языка, ассемблера) или языка высокого уровня (C, Basic, Pascal). Для PIC-микроконтроллеров, и особенно входящих в подсемейство PIC 16CXX, ассемблер – самое лучшее решение, даже если он кажется более трудным в применении.

Действительно, размер памяти, доступный программе в микроконтроллерах этого подсемейства, относительно мал (от 512 до 4 К слов), что делает невозможным размещение большой программы. Программа, написанная на ассемблере, требует минимального объема памяти и позволяет реализовать максимальное количество функций. Напротив, если она создана на языке высокого уровня, то в памяти микроконтроллера может храниться код, соответствующий всего лишь нескольким десяткам или сотням операторов языка.

В таких языках, как C, Basic или Pascal, имеется множество типов переменных и обрабатывающих их операторов, что позволяет писать очень компактные исходные тексты программ. Но даже самый простой оператор на языке высокого уровня при компиляции

генерирует несколько десятков команд машинного языка, из-за чего доступная память заполняется очень быстро.

Язык высокого уровня целесообразно использовать для микроконтроллеров с большим объемом памяти, или в тех случаях, когда время, отведенное на разработку приложения, очень ограничено. Конечно, для этого необходимо знание соответствующего языка. Но лучше обратитесь к главам 3 и 4 настоящей книги, где описаны 35 команд PIC-микроконтроллеров и основы ассемблера. Вы убедитесь, что этих знаний вполне достаточно для написания программы.

СИСТЕМА РАЗРАБОТКИ

Некоторые понятия, относящиеся к системам разработки и соответствующему программному обеспечению, воспринимаются достаточно трудно, поэтому имеет смысл напомнить читателям основные термины.

Ассемблер и компилятор языка высокого уровня

Комплекс разработки всегда включает как минимум ассемблер и иногда один или несколько компиляторов языков высокого, которые используются для программирования.

Ассемблер переводит мнемокоды команд машинного языка в двоичные (бинарные) коды, исполняемые PIC-микроконтроллером. Текст с последовательностью мнемокодов называется листингом, или *исходным кодом* программы, в то время как бинарный код называется *объектным*, или *исполняемым*.

Компилятор переводит операции, операторы и другие конструкции языка высокого уровня, образующие исходный текст программы, в исполняемый бинарный код PIC-микроконтроллера.

В хорошо продуманной системе (среде) разработки обе программы, ассемблер и компилятор, могут сосуществовать отдельно или использоваться вместе. Это позволяет строить сложные алгоритмы на языке высокого уровня, а те программные модули, которые требуют высокой производительности или предназначены для управления периферией через порты ввода/вывода (*драйверы*), писать на ассемблере. Очень важно, чтобы компилятор языка высокого уровня допускал включение программных сегментов на ассемблере.

Эти обе программы, ассемблер и/или компилятор, должны обязательно «прогоняться» на так называемом *хост-компьютере*. В таком

качестве может выступать практически любая машина: специализированная система производителя PС-микроконтроллеров (все более и более редкий случай, так как это очень дорого), мощная ЭВМ (VAX, рабочая станция Sun, Apollo или HP и т.п.) или просто, и это становится правилом, IBM совместимый ПК. Последний вариант позволяет уменьшить капиталовложения, поскольку наверняка уже имеется IBM совместимый компьютер, используемый для других целей.

Написав программу на ассемблере или языке высокого уровня и проведя компиляцию, вы получаете исполняемый бинарный код. Однако прежде чем тиражировать созданное приложение, надо тщательно его протестировать. Чтобы обеспечить полный контроль, программа должна тестироваться в условиях, максимально близких к условиям будущего реального использования. Для этого существует несколько возможных решений.

Эмулятор и симулятор

Первое решение, которое является самым эффективным, но также, увы, и наиболее дорогостоящим, – приобрести специальную аппаратуру. На рис. 2.1 показано одно из таких устройств. Аппаратный *эмулятор*, составляющий «сердце» этой системы, фактически представляет собой специальное устройство, иногда очень сложное, которое выполняет все функции PС-микроконтроллера и заменяет его. На рисунке представлена «разрозненная» версия, то есть версия, в которой все внутренние компоненты PС-микроконтроллера реализованы в виде отдельных схем.

Этот эмулятор снабжен специальным соединительным шлейфом, называемым *зондом эмуляции*, на конце которого есть разъем, аналогичный выводам корпуса PС-микроконтроллера. С помощью шлейфа эмулятор подключается к макету приложения вместо настоящего PС-микроконтроллера.

Для подключения эмулятора к компьютеру обычно используется последовательный интерфейс RS232. Через него в эмулятор загружается программа микроконтроллера и осуществляется управление ее выполнением с помощью специальной компьютерной программы. Так как эмулятор – это «разрозненная» версия PС-микроконтроллера, вы имеете доступ к его различным внутренним узлам и шинам, в частности, можете узнать, по каким адресам «проходит» программа, какие коды записаны в управляющие регистры и т.п. В случае

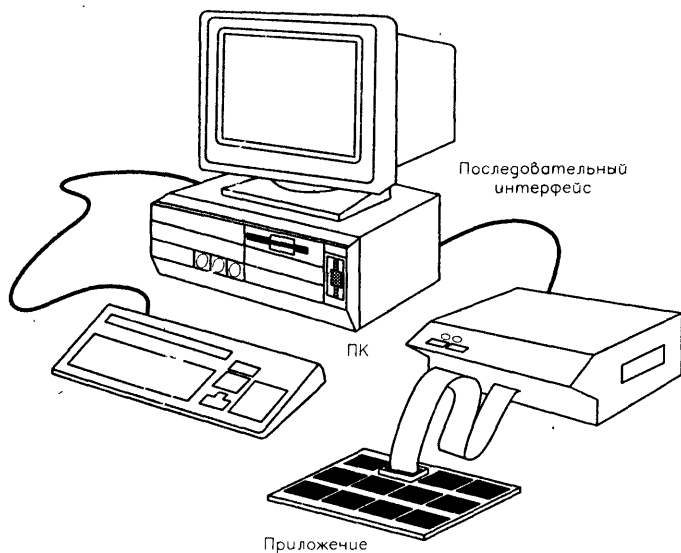


Рис. 2.1

Общий вид эмулятора

необходимости можно остановить исполняемую программу, чтобы считать состояние памяти или некоторых регистров.

Эмулятор обеспечивает работу приложения в реальном времени, поскольку способен функционировать на частоте микроконтроллера. Эмулятор – дорогостоящее промышленное средство, поэтому вряд ли имеет смысл покупать его для создания мелкосерийных или единичных приложений.

Второе возможное решение – использовать *симулятор*, но он не позволяет провести все необходимые тесты. Симулятор – это программа, написанная специально для микроконтроллера, работу которого она будет имитировать. Симулятор обычно функционирует на той же машине, где писалась программа приложения. На ее вход подается объектный (или исполняемый) код, который необходимо протестировать, и программа ведет себя так, как вел бы себя имитируемый микроконтроллер.

Симулятор значительно дешевле, чем эмулятор, ведь это только программа, сложность которой пропорциональна сложности микроконтроллера. Стоит он в десятки раз меньше, чем эмулятор.

Регистры и порты ввода/вывода микроконтроллера представляются переменными, хранящимися в памяти компьютера. Так,

8-разрядный параллельный порт представлен байтом данных. Достаточно считать эти данные, чтобы в любой момент знать о ходе выполнения программы и состоянии выходов. Ввод данных через порт можно имитировать записью соответствующих данных в память. Процедура моделирования операций обмена через порты ввода/вывода «растягивается» во времени, но если она хорошо проведена, то позволяет проверить 80% функций программы микроконтроллера.

Однако положение усложняется, когда необходимо учитывать временные характеристики сигналов, передаваемых через порты ввода/вывода. Симулятор – это программа, воспроизводящая лишь алгоритм работы микроконтроллера. Даже если использован высокопроизводительный компьютер, симулятор работает в десятки, а иногда и в сотни раз медленнее, чем та же программа, непосредственно выполняемая микроконтроллером.

Некоторые функции, связанные со временем, нельзя полностью смоделировать на симуляторе. Тем не менее, хорошее владение симулятором позволяет достаточно быстро и с минимальными капиталовложениями тестировать приложения.

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ПРОМЫШЛЕННОГО ПРИМЕНЕНИЯ

Разработку микроконтроллерных приложений для промышленного использования (о приложениях *ограниченного* применения будет сказано чуть ниже), целесообразно проводить следующим образом.

Сначала необходимо выбрать элементную базу разрабатываемого устройства, поскольку от этого зависит и качество аппаратуры (надежность, стабильность, быстродействие), и объем разрабатываемого программного обеспечения. Выбор элементной базы должен быть тщательно продуман и обоснован до начала разработки программного обеспечения, так как в некоторых случаях дальнейшие замены компонентов могут привести к полной переделке программы.

Как только выбор сделан, нужно определить структуру программного обеспечения. В результате этой работы может оказаться, что в элементную базу требуется внести какие-либо изменения. Допустим, составляя структуру и оценивая время, необходимое для выполнения различных программных функций, вы поняли, что выбранные компоненты (каналы передачи данных, например) не обеспечат необходимой скорости функционирования. Придется отступить на шаг назад, чтобы изменить некоторые компоненты.

Когда структура программного обеспечения составлена, приходит время написания программы. Чтобы облегчить последующее тестирование и отладку, нужно разделить ее на несколько модулей, каждый из которых выполняет определенные функции. Эти модули нужно писать поочередно и тестировать один за другим, что особенно важно для модулей драйверов, управляющих портами ввода/вывода. В идеале, каждым портом должен управлять соответствующий программный модуль.

Когда программное обеспечение написано, его необходимо протестировать. В зависимости от обстоятельств и специфики разрабатываемого устройства применяют симулятор или эмулятор. По завершении тестирования можно выполнить *натурное моделирование* работы устройства. Для этого предназначены перепрограммируемые версии микроконтроллера (EEPROM – с электрическим стиранием или UVPROM – с ультрафиолетовым стиранием), идентичные версии ОТР, которую вы сможете использовать позже, когда программное обеспечение будет доработано.

Именно перепрограммируемые версии следует использовать для проведения последних тестов устройства на макете или опытном образце. Это позволит в случае необходимости внести нужные изменения. Если все окажется правильным, переходите к программированию микросхем ОТР.

Как видите, этапы создания микроконтроллерных приложений и программного обеспечения классической информационной системы практически совпадают, но в первом случае некоторые из них оказываются более критичными из-за тесной взаимосвязи между элементной базой и программным обеспечением.

Общий алгоритм разработки приложения приведен на рис. 2.2.

СРЕДСТВА РАЗРАБОТКИ ФИРМЫ MICROCHIP

Тому, кто разрабатывает приложения на основе микроконтроллеров только от случая к случаю, не имеет смысла инвестировать средства в дорогостоящий эмулятор. Если речь идет о создании небольшого количества приложений с использованием микроконтроллеров PIC 16CXX, можно обойтись и без него благодаря комплектам для разработки, выпускаемым фирмами Microchip и Parallax. Такие комплекты содержат очень хороший ассемблер и достаточно эффективный симулятор.

Конечно, симулятор не имеет смысла полностью заменить эмулятор. Однако он позволяет любому внимательному разработчику

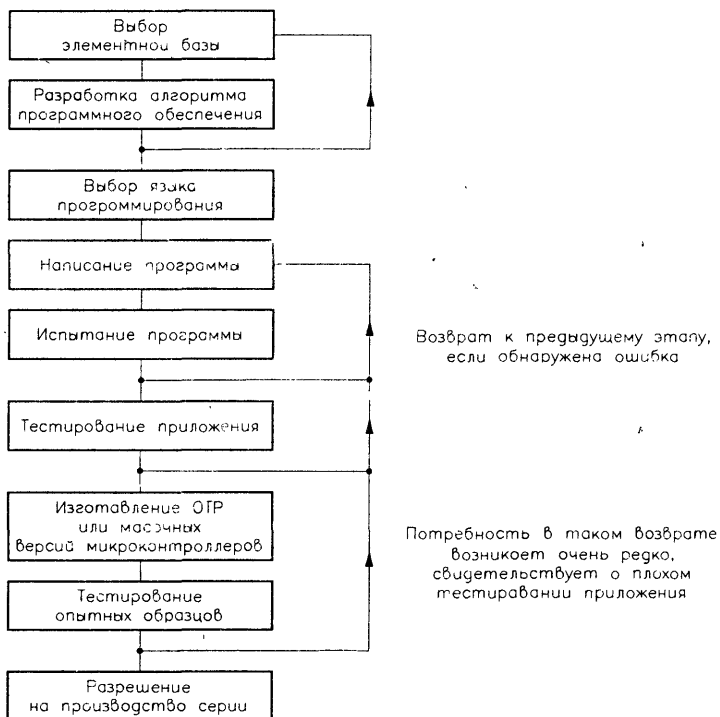


Рис. 2.2

Последовательность разработки приложения

достаточно точно протестировать свое приложение, особенно если окончательную доводку проводить с использованием микросхем версии UVPROM (с ультрафиолетовым стиранием).

Системы Picstart-16b и Picstart-16c

Фирма Microchip предлагает различные средства разработки, среди которых – внутрисхемный эмулятор PICMASTER, способный поддерживать все микросхемы PIC 16CXX. Программное обеспечение эмулятора работает под ОС Windows. Этот продукт можно порекомендовать любому серьезному разработчику, который располагает достаточными средствами для его приобретения.

Есть и более приемлемые по цене продукты, которые позволяют успешно разрабатывать приложения на основе PIC 16CXX. Речь идет о системе разработки микроконтроллерных приложений, известной под названием PICSTART-16.

Когда писалась книга, этот продукт существовал в двух версиях:

- ♦ PICSTART-16B1 – позволяет осуществлять разработку приложений на основе PIC 16C5х, 16C61, 16C71 и 16C84;
- ♦ PICSTART-16C – поддерживает микроконтроллеры 16C64, 16C65, 16C73 и 16C74.

Эти системы отличаются только перечнем поддерживаемых микросхем, общие же их характеристики идентичны.

Несмотря на низкую цену, возможности PICSTART-16 способны удовлетворить самого требовательного пользователя. Обе системы включают:

- ♦ макроассемблер MPALC* для микроконтроллеров подсемейства PIC 16CXX;
- ♦ симулятор для указанных микросхем;
- ♦ плату программатора с соответствующим программным обеспечением;
- ♦ сетевой источник питания и кабель для подключения программатора к последовательному порту компьютера;
- ♦ образцы PIC-контроллеров подсемейства 16CXX (количество и типы меняются согласно комплекту и дате приобретения).

Все программное обеспечение может работать на любом IBM совместимом ПК и не очень требовательно к его ресурсам: достаточно 640 Кб оперативной памяти и операционной системы DOS версии 3.3 или выше.

В очень подробной инструкции по эксплуатации PICSTART-16 содержатся рекомендации по установке данного продукта на компьютере, а также описание программного обеспечения разработки, включающего макроассемблер и симулятор.

Что касается инструкций по применению макроассемблера и симулятора, вам надо будет их распечатать на принтере, так как они находятся в текстовом файле на прилагаемом диске.

Программное обеспечение разработки

Минимально необходимые возможности для разработки приложений предоставляют макроассемблер и симулятор, которые позволяют соответственно писать программы и тестировать их.

* По каталогу фирмы Microchip данный макроассемблер называется MPASM. – *Прим. ред.*

Ассемблер MPALC

Данный ассемблер – продукт очень высокого уровня, и это комплимент не новичка, поскольку автор практикует микропроцессорную технику уже более 15 лет! Помимо классических возможностей любого ассемблера MPALC поддерживает ряд макрокоманд и директив, например *условное ассемблирование*.

В ассемблере заложены модели всех микросхем серии PIC 16CXX, а также традиционные имена регистров, констант, переменных величин и т.д. Цифровые данные могут быть выражены во всех классических представлениях: бинарном, десятичном, восьмеричном, шестнадцатеричном и символьном (ASCII).

Он разрешает запись арифметических и логических выражений над данными и адресами, а также позволяет записывать операции отношения, сдвига вправо и влево.

Файлы с программами могут быть написаны в любом текстовом редакторе, единственное требование – они должны быть представлены в текстовом формате ASCII.

Используемый в MPALC синтаксис – классический для ассемблера, и даже менее консервативный, чем у аналогов. Допускается форматировать исходный текст с помощью табуляции. Идентификаторы способны включать до 32 знаков, причем различаются прописные и строчные буквы, но по желанию эту опцию можно отключить.

На выходе ассемблер создает различные файлы: файл для программирования микроконтроллера, файл листинга, файл сообщений об ошибках и предупреждений и, наконец, файл для симулятора.

Различные форматы файлов для программирования микроконтроллеров позволяют использовать различные программаторы, что соответствует требованиям рынка.

Возможности ассемблера расширяет поддержка макрокоманд. Что такое макрокоманда? Это выбранная вами часть программы, которая выполняет обычно одну часто используемую и хорошо отлаженную функцию, определенная как макрокоманда, названная оригинальным именем и включающая переменные величины. Однажды определенная, эта макрокоманда может быть помещена в любое место программы и интерпретирована ассемблером.

Наконец, условное ассемблирование позволяет, в зависимости от значения некоторых определенных вами параметров, откомпилировать ту или иную часть программы. Это очень интересная функция, хотя и недооцененная многими программистами. Благодаря

ей можно написать программу, общую для нескольких приложений, отличающихся только деталями, например для таймера с жидкокристаллической индикацией и таймера на светодиодах. В ходе компиляции параметры, определяющие условия ассемблирования, показываются в начале листинга, чтобы было ясно, для какого приложения предназначена программа.

Листинги некоторых программ, которые вы найдете в главах 3 и 4, были написаны именно для компоновки с помощью ассемблера MPALC.

Симулятор MPSIM

Симулятор помогает проверить на IBM совместимом ПК «поведение» вашей программы. Он имитирует работу выбранного микроконтроллера подсемейства PIC 16CXX и позволяет моделировать состояние его памяти, управляющих регистров и портов ввода/вывода в процессе выполнения микропрограммы.

Несмотря на то, что MPSIM работает в среде DOS и информация, которую он представляет разработчику в текстовой форме (рис. 2.3), выглядит малопривлекательно по сравнению с документами Windows, эффективность симулятора сомнений не вызывает. В верхней части окна выводится текущее содержимое регистровой памяти микроконтроллера, в том числе управляющих регистров. Эти данные можно редактировать.

Идентификация регистров осуществляется либо по их адресам, либо по именам, которые задаются в тексте программы. Последнее значительно упрощает отладку программы, поскольку определить назначение регистра проще по его имени, чем по обезличенному адресу. Можно отображать не только состояние регистров, но и отдельных битов. В рассматриваемом примере это показано для порта В, представленного бит за битом с RB7 по RB0. Более того, отдельные биты регистров могут идентифицироваться по именам.

Содержимое регистров будет меняться по мере трассировки имитируемой программы.

Нижняя часть экрана – это рабочее окно, где отображается листинг трассируемой (то есть выполняемой в пошаговом режиме) программы. Здесь же пользователь может набирать команды, необходимые для управления процессом отладки, и наблюдать их выполнение. Пример работы в режиме трассировки представлен на рис. 2.4. В правом верхнем углу экрана показывается теоретическое время выполнения программы и номер шага.

```

SAMPLE      RADIX=X  MPSIM 4.14      16c54      TIME=0.00µ 0
mulcnd: 00 mulplr: 00 H_byte: 00 L_byte: 00 count: 00 portb: FF RB7: 1
RB6: 1 RB5: 1 RB4: 1 RB3: 1 RB2: 1 RB1: 1 RB0: 1

% AD mulplr
% AD H_byte
% AD L_byte
% AD count
% AD portb
% AD RB7.B.1
% AD RB6.B.1
% AD RB5.B.1
% AD RB4.B.1
% AD RB3.B.1
% AD RB2.B.1
% AD RB1.B.1
% AD RB0.B.1
% RS
Processor Reset
382304 bytes memory free
%

```

Рис. 2.3

Вид диалогового окна симулятора MPSIM фирмы Microchip

```

SAMPLE      RADIX=X  MPSIM 4.14      16c54      TIME=32.00µ 14
mulcnd: 09 mulplr: FF H_byte: 00 L_byte: 00 count: 08 portb: 05 RB7: 0
RB6: 0 RB5: 0 RB4: 0 RB3: 0 RB2: 1 RB1: 0 RB0: 1

382304 bytes memory free
% to OE
01FF 0A0E      goto      start          4.00 1
000E 0040 start clrf          6.00 2 W:0 F3:1C
000F 0002      option          S.00 3 OPT:C0
0010 0206 main  movf      portb,w        10.00 4 W:FF F3:18
0011 0030      movwf     mulplr          12.00 5 F10:FF F3:18
0012 0206      movf      portb,w        14.00 6 W:9 F3:18
0013 0029      movwf     mulcnd         16.00 7 F9:9 F3:18
0014 0900 call_m   call      mpy_s       20.00 8 [15.0]
0000 0072 mpy_s  clrf      H_byte        22.00 9 F12:0 F3:1C
0001 0073      clrf      L_byte        24.00 10 F13:0 F3:1C
0002 0C08      movlw     8              26.00 11 M:8
0003 0034      movwf     count          28.00 12 F14:8 F3:1C
0004 0209      movf      mulcnd,w       30.00 13 W:9 F3:18
0005 0403      bcf      STATUS,CARRY    32.00 14 STATUS:18
%

```

Рис. 2.4

Вид экрана симулятора MPSIM в режиме трассировки программы

Симулятор MPSIM включает все функции, обычные для таких устройств: запуск симулируемой программы, трассировку программы с заданным шагом, установку точек прерывания, выполнение программ в режиме трассировки с прерываниями по условию, отображение содержимого памяти и управляющих регистров и т.д.

Он обеспечивает возможность редактирования отлаживаемой программы, оперативного изменения содержимого памяти, располагает функцией, позволяющей запоминать только что сделанные изменения таким образом, чтобы их можно было легко переносить потом в исходную программу.

Симулятор имитирует внешние сигналы, подаваемые на входы микроконтроллера. Осуществляется это с помощью специального файла, где хранится информация о том, какие сигналы должны подаваться на различные входы и в какие моменты времени. Данный файл можно сформировать во время работы симулятора. MPSIM поддерживает командный файл, в котором содержатся различные функции (команды) управления процессом симулирования, начальные условия, адреса точек прерываний и т.п. Наличие командного файла избавляет пользователя от необходимости при каждом новом цикле отладки программы повторно набирать все команды и условия. Вместо этого достаточно просто обратиться к командному файлу, и все загрузится автоматически. Напомним, что все файлы, с которыми работает симулятор, – стандартные текстовые файлы формата ASCII, и они могут быть прочитаны и отредактированы с помощью команд DOS или любого текстового редактора.

В заключение добавим, что результаты моделирования можно выводить в файл или на печать, можно вести журнал симулирования, дезассемблировать код, вести поиск данных в памяти и т.д.

Надеемся, что несмотря на краткость обзора возможностей симулятора MPSIM, вы убедились в его эффективности и целесообразности использования при отладке программного обеспечения.

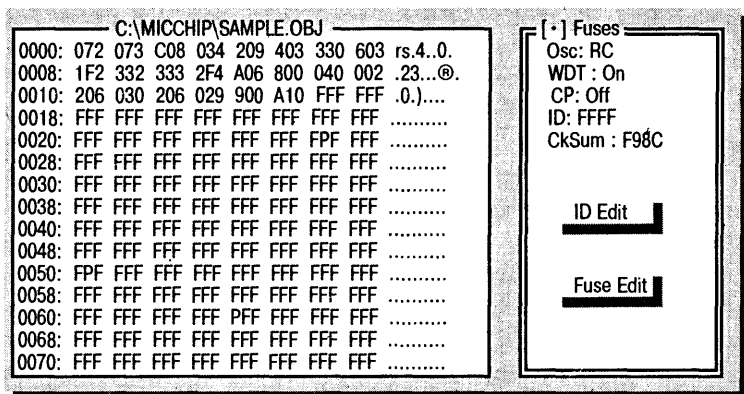
Программатор

Программатор – это последний элемент набора разработки программного обеспечения для микроконтроллеров, наиболее простой из трех представленных. Поддержка его работы осуществляется специальной программой, обеспечивающей развитый диалог в псевдографическом режиме, с разворачивающимися меню и диалоговыми окнами (рис. 2.5).

Он обладает набором типичных для всех микроконтроллерных программаторов функций: выбор типа программируемой микросхемы, тест микросхемы «на чистоту», программирование, считывание и некоторые другие менее важные функции.

Аппаратную часть программатора образует плата, подключаемая к одному из четырех последовательных портов IBM, совместимого ПК. Связь с компьютером через порт тестируется до начала программирования.

PicStart File Windows Options Device Transfer Config P16C54 21:22:37



F1 Help Alt-X Exit F4 Edit F5 Program F6 Verify F7 Blank F8 Read

Рис. 2.5

Экран программатора фирмы Microchip

Работа данного программатора не требует особых комментариев. Отметим лишь, что он не предназначен для массового тиражирования микроконтроллеров.

СИНТАКСИС АССЕМБЛЕРА MPALC

Синтаксис, используемый ассемблером MPALC, достаточно традиционный и не должен вызывать трудностей у пользователей, уже работавших с машинным языком. Но, чтобы вы могли легко читать листинги, приведенные в следующих главах, ниже даются основные сведения об этом ассемблере. Детальное описание языка вы найдете в учебнике по MPALC.

Как и большинство ассемблеров, MPALC нечувствителен к управляющему символу «табуляция», и поэтому листинги можно табулировать как вам нравится. Ассемблер не различает прописные и строчные

буквы в командах и директивах. Они различаются в только в идентификаторах (именах), если не отключена соответствующая опция.

Комментарии допускается размещать в любых строках программы. Им предшествует точка с запятой.

Идентификаторы могут включать от 1 до 32 знаков, но не должны использовать зарезервированные слова, являющиеся директивами и командами.

Цифровые данные могут быть выражены в десятичном, шестнадцатеричном, восьмеричном, двоичном и ASCII кодах. Типы используемых числовых данных указаны в табл. 2.1. В табл. 2.2 приведены математические операторы.

Таблица 2.1

Типы числовых данных, допускаемые MPALC

Формат данных	Способ записи	Пример
Десятичный	Данные	100
	.Данные	.100
Шестнадцатеричный	H'Данные'	H'9f'
	Данные h	900h
	0xДанные	0x9ff
Восьмеричный	O'Данные'	O'777'
	Q'Данные'	Q'777'
	Данные O	777O
	Данные Q	777Q
	\ Данные	\ 777
Двоичный	B'Данные'	B'0011101'
	Данные B	0011101B
ASCII	'Данные'	'C'
	A'Данные'	A'C'

Чаше других употребляются следующие директивы ассемблера:

- ◆ DATA – инициализирует несколько слов, хранящихся в памяти;
- ◆ ZERO – заполняет нулями область памяти, размер которой задается числом, следующим за директивой;
- ◆ SET – определяет значение переменной;
- ◆ RES – резервирует память, объем которой указан следующим за директивой числом;
- ◆ EQU – определяет константу в программе;
- ◆ INCLUDE – позволяет вставить файл внутрь текущей программы;
- ◆ IF – определяет начало кодового раздела условного ассемблирования;
- ◆ ELSE – определяет начало альтернативного кодового раздела условного ассемблирования;

- ◆ **ENDIF** – заканчивает кодовый раздел условного ассемблирования;
- ◆ **ORG** – определяет абсолютный адрес кода;
- ◆ **END** – заканчивает исходный текст;
- ◆ **MACRO** – определяет начало кода макрокоманды;
- ◆ **ENDM** – определяет конец кода макрокоманды;
- ◆ **LOCAL** – определяет локальную переменную в макрокоманде;
- ◆ **EXITM** – выход из макрокоманды.

Есть и другие директивы, относящиеся к форматированию или нумерации страниц. Но поскольку они не влияют на понимание листингов программ, представленных в следующих главах, в этой книге описываться не будут.

Таблица 2.2

Математические операторы

Оператор	Функция
+	Сложение
-	Вычитание
-	Изменение знака
%	Деление по модулю
*	Умножение
/	Деление
<<	Сдвиг влево
>>	Сдвиг вправо
()	Скобки
=	Равно
!=	Не равно
<=	Меньше или равно
>=	Больше или равно
!	Логическое отрицание
1	Поразрядное ИЛИ
&	Поразрядное И
	Логическое ИЛИ
&&	Логическое И
^	Поразрядное исключающее ИЛИ
\$	Текущее значение программного счетчика

АСЕМБЛЕР PASM

Микроконтроллеры семейства MCS51 фирмы Intel очень распространены, и их ассемблер известен многим программистам. В связи с этим американская фирма Parallax создала ассемблер под названием PASM, который поддерживает набор команд PIC-микроконтроллера и псевдоинструкций, аналогичных инструкциям 8051.

Так, вы можете записать в PASM:

```
ADD A, B
```

Что будет переведено им в:

```
MOVF B, 0
```

```
ADDWF A, 1
```

Для тех, кто уже работал с ассемблером 8051, первая строка будет более привычной, чем две эквивалентные команды для PIC-микроконтроллеров.

Этот ассемблер и его псевдоинструкции микроконтроллера 8051 используются в листингах нескольких приложений, представленных в данной книге. Поэтому в табл. 2.3 приведены правила перевода псевдоинструкций ИС 8051 в инструкции PIC-микроконтроллеров. Как можно заметить, эти правила естественно вытекают из функций, соответствующих инструкций.

Синтаксис этого ассемблера подобен синтаксису MPALC, но имеет несколько отличий.

PASM не поддерживает ни макрокоманды, ни условное ассемблирование, и соответствующих директив, естественно, не существует. Отсутствует восьмеричное представление данных. Что касается операторов, то они не столь многочисленны, как видно из табл. 2.4.

Директив ассемблера также меньше:

- ◆ DS – заполняет заданное число ячеек памяти указанным значением;
- ◆ EQU – определяет константу в программе;
- ◆ = – выполняет ту же функцию, что и EQU;
- ◆ INCLUDE – позволяет включить файл в текущую программу;
- ◆ ORG – определяет абсолютный адрес для кода;
- ◆ END – заканчивает текст программы;
- ◆ DEVICE – определяет тип используемого PIC-микроконтроллера;
- ◆ RESET – определяет адрес сброса.

Однако есть у ассемблера PASM и преимущества – он может определять идентификаторы для битов. Так, запись `PORTC.3` будет интерпретирована как «бит 3 регистра PORTC».

Кроме того, в PASM имеется несколько стандартных *логических* имен регистров PIC-микроконтроллера и возможность вызывать файл со стандартными именами регистров посредством директивы `INCLUDE`. Таким образом, `RA` соответствует порту A, `RB` – порту B и т. д. Чтение листингов, представленных в этой книге и написанных на данном ассемблере, не вызовет у вас трудностей.

СРЕДСТВА РАЗРАБОТКИ ФИРМЫ PARALLAX

Фирма Parallax много и успешно занимается выпуском средств разработки микроконтроллеров. В этом разделе будет рассказано о наиболее интересных из предлагаемых продуктов.

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера

Псевдокоманда 8051	Эквивалент PIC
ADD fr, #literal	MOVLWF literal ADDWF fr,1
ADD fra,frb	MOVF frb,0
ADD fr,W	ADDWF fra,1
ADD W,Fr	ADDWF fr,0
ADDB fr,bit	BTFSC bit incf fr,1
AND fr, #literal	MOVLW literal ANDWF fr,1
AND fra,frb	MOVF frb,0 ANDWF fra,1
AND fr,W	ANDWF fr,1
AND W, #literal	ANDLW literal
AND W,fr	ANDWF fr,0
CALL addr	CALL adr8
CJA fr, #literal, addr9	MOVLW literal^0FFh ADDWF fr,0 BTFSC 3,0 GOTO addr9
CJA fr1, fr2, addr9	MOVF fr1,0 SUBWF fr2,0 BTFSC 3,0 GOTO addr9
CJAE fr, #literal, addr9	MOVLW literal SUBWF fr,0 BTFSC 3,0 GOTO addr9

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (продолжение)

Псевдокоманда 8051	Эквивалент PIC
CJAE fr1,fr2,addr9	MOVF fr2,0 SUBWF fr1,0 BTFSC 3,0 GOTO addr9
CJB fr,#literal,addr9	MOVLW literal SUBWF fr,0 BTFSS 3,0 GOTO addr9
CJB fr1,fr2,addr9	MOVF fr2,0 SUBWF fr1,0 BTFSS 3,0 GOTO addr9
CJBE fr,#literal,addr9	MOVLW /literal ADDWF fr,0 BTFSS 3,0 GOTO addr9
CJBE fr1,fr2,addr9	MOVF fr1,0 SUBWF fr2,0 BTFSC 3,0 GOTO addr9
CJE fr,#literal,addr9	MOVLW literal SUBWF fr,0 BTFSC 3,2 GOTO addr9
CJE fr1,fr2,addr9	MOVF fr2,0 SUBWF fr1,0 BTFSS 3,2 GOTO addr9
CJNE fr,#literal,addr9	MOVLW literal SUBWF fr,0 BTFSS 3,2 GOTO addr9

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (продолжение)

Псевдокоманда 8051	Эквивалент PIC
CJNE fr1,fr2,addr9	MOVF fr2,0 SUBWF fr1,0 BTFSS 3,2 GOTO addr9
CLC	BCF 3,0
CLR fr	CLRF fr
CLR W	CLRW
CLR WDT	CLRWDT
CLRB bit	BCF bit
CLZ	BCF 3,2
CSA fr,#literal	MOVLW /literal ADDWF fr,0 BTFSS 3,0
CSA fr1,fr2	MOVF fr1,0 SUBWF fr2,0 BTFSC 3,0
CSAE fr,#literal	MOVLW literal SUBWF fr,0 BTFSS 3,0
CSAE fr1,fr2	MOVF fr2,0 SUBWF fr1,0 BTFSS 3,0
CSB fr,#literal	MOVLW literal SUBWF fr,0 BTFSC 3,0
CSB fr1,fr2	MOVF fr2,0 SUBWF fr1,0 BTFSC 3,0
CSBE fr,#literal	MOVLW /literal ADDWF fr,0 BTFSC 3,0

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (продолжение)

Псевдокоманда 8051	Эквивалент PIC
CSBE fr1,fr2	MOVF fr1,0 SUBWF fr2,0 BTFSS 3,0
CSE fr,#literal	MOVLW literal SUBWF fr,0 BTFSS 3,2
CSE fr1,fr2	MOVF fr2,0 SUBWF fr1,0 BTFSS 3,2
CSNE fr,#literal	MOVLW literal SUBWF fr,0 BTFSC 3,2
CSNE fr1,fr2	MOVF fr2,0 SUBWF fr1,0 BTFSC 3,2
DEC fr	DECF fr,1
DECSZ fr	DECFSZ fr,1
✓ DJNZ fr,addr9	DECFSZ fr,1 GOTO addr9
IJNZ fr,addr9	INCFSZ fr,1 GOTO addr9
INC fr	INCF fr,1
INCSZ fr	INCFSZ fr,1
JB bit,addr9	BTFSC bit GOTO addr9
JC addr9	BTFSC 3,0 GOTO addr9
JMP addr9	GOTO addr9
JMP PC+W	ADDWF 2,1
JMP W	MOVWF 2
JNB bit,addr9	BTFSS bit GOTO addr9

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (продолжение)

Псевдокоманда 8051	Эквивалент PIC
JNC addr9	BTFSS 3,0 GOTO addr9
JNZ addr9	BTFSS 3,2 GOTO addr9
JZ addr9	BTFSC 3,2 GOTO addr9
MOV fr,#literal	MOVLW literal MOVWF fr
MOV fr1,fr2	MOVF fr2,0 MOVWF fr1
MOV fr,W	MOVWF fr
MOV OPTION,#literal	MOVLW literal OPTION
MOV OPTION,fr	MOVF fr,0 OPTION MOV OPTION,W
MOV !port_fr,#literal	MOVLW literal OPTION TRIS port_fr
MOV !port_fr,fr	MOVF fr,0 TRIS port_fr
MOV !port_fr,W	TRIS port_fr
MOW W,#literal	MOVLW literal
MOV W,fr	MOVF fr,0
MOV W,/fr	COMF fr,0
MOV W,fr-W	SUBWF fr,0
MOV W,++fr	INCF fr,0
MOV W,-fr	DECF fr,0
MOV W,<<fr	RLF fr,0
MOV W,>>fr	RRF fr,0
MOV W,<>fr	SWAPF fr,0

Таблица 2.3

Соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (продолжение)

Псевдокоманда 8051	Эквивалент PIC
MOVB bit1,bit2	BTFSS bit2 BCF bit1 BTFSC bit2 BSF bit1
MOVB bit1,/bit2	BTFSC bit2 BCF bit1 BTFSS bit2 BSF bit1
MOVSZ W,++fr	INCFSZ fr,0
MOVSZ W,-fr	DECFSZ fr,0
NEG fr	COMF fr,1 INCF fr,1
NOP	NOP
NOT fr	COMF fr,1
NOT W	XORLW OFFh
OR fr,#literal	MOVLW literal IORWF fr,1
OR fr1,fr2	MOVF fr2,0 IORWF fr1,1
OR fr,W	IORWF fr,1
OR W,#literal	IORLW literal
OR W,fr	IORWF fr,0
RET	RETLW 0
RETW literal1, literal2,...	RETLW literal1 RETLW literal2 ...
RL fr	RLF fr,1
RR fr	RRF fr,1
SB bit	BTFSS bit
SC	BTFSS 3,0
SETB bit	BSF bit

Таблица 2.3

(соответствие псевдоинструкций ассемблера фирмы PARALLAX и классических инструкций PIC-микроконтроллера (окончание))

Псевдокоманда 8051	Эквивалент PIC
SKIP	BTFSS 4,7
SLEEP	SLEEP
SNB bit	BTFSC bit
SNC	BTFSC 3,0
SNZ	BTFSC 3,2
STC	BSF 3,0
STZ	BSF 3,2
SUB fr,#literal	MOVLW literal SUBWF fr,1
SUB fr1,fr2	MOVF fr2,0 SUBWF fr1,1
SUB fr,W	SUBWF fr,1
SUBB fr,bit	BTFSS 3,0 DECF fr,1
SWAP fr	SWAPF fr,1 BTFSS 3,2
TEST fr	MOVF fr,1
TEST W	IORLW 0
XOR fr,#literal	MOVLW literal XORWF fr,1
XOR fr1,fr2	MOVF fr2,0 XORWF fr1,1
XOR fr,W	XORWF fr,1
XOR W,#literal	XORLW literal
XOR w,fr	XORWF fr,0

Таблица 2.4

Операторы ассемблера фирмы Parallax

Оператор	Функция
+	Сложение
-	Вычитание
-	Изменение знака
*	Умножение
/	Деление
<<	Сдвиг влево
>>	Сдвиг вправо
&	Логическое И
	Логическое ИЛИ
^	Исключающее ИЛИ
<	Старший байт
>	Младший байт
.	Адрес бита

Псевдоэмулятор Reflection-5X

В каталоге фирмы Parallax представлен очень интересный симулятор Reflection-5x для микроконтроллеров семейства PIC 16CXX, который способен имитировать работу портов ввода/вывода.

Это устройство, как показано на рис. 2.6, подсоединяется к отлаживаемой схеме как эмулятор, то есть заменяет микроконтроллер в вашем приложении. Псевдоэмулятор управляется через последовательный порт IBM совместимого ПК (COM1 или COM2) и поддерживается специальным программным обеспечением.

Необходимо помнить, что Reflection-5x все же симулятор и не может работать в реальном времени. Тем не менее он позволяет успешно проводить тестирование аппаратуры приложения.

Симулятор Reflection-5x питается от сетевого адаптера, дающего постоянное стабилизированное напряжение от 9 до 12 В при токе в 250 мА. Два плоских кабеля, с 18- и 28-контактными разъемами, позволяют подключать его вместо PIC-микроконтроллера. Благодаря DIP-переключателю устройства можно подключить выводы MCLR и RTCC либо к положительному полюсу питания вашего приложения, либо к аналогичной цепи симулятора Reflection-5x. Связь с ПК осуществляется через выбираемый вами последовательный порт с помощью входящего в комплект устройства кабеля. При этом программное обеспечение Reflection-5x автоматически обнаруживает

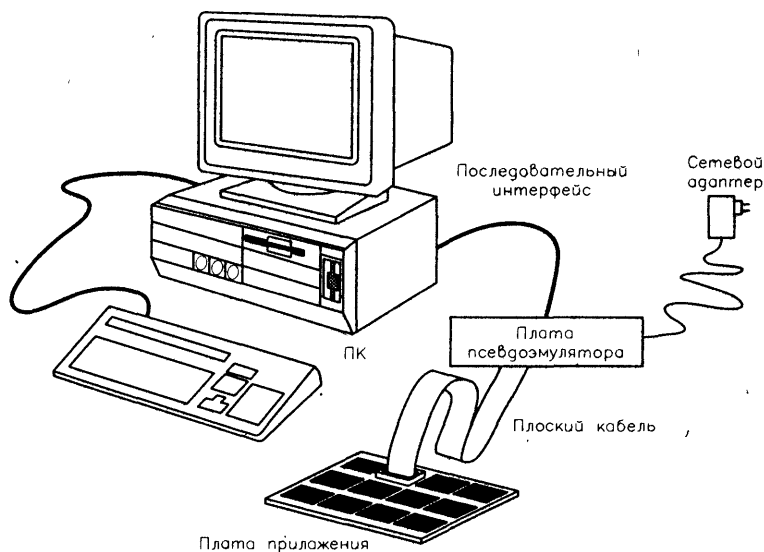


Рис. 2.6

Подключение симулятора Reflection-5x

и настраивает порт. Обмен осуществляется на максимальной скорости передачи порта.

Программное обеспечение симулятора использует для диалога трехоконную конфигурацию экрана. Верхнюю правую часть занимает фрейм, где показано содержимое оперативной памяти микроконтроллера; в центре и справа отображается содержимое управляющих регистров; в нижней части окна вы можете контролировать ход моделирования по исходному листингу.

Симулятор позволяет вводить неограниченное число точек прерывания, допускает пошаговое выполнение программы, прогон от одного прерывания до следующего. Он разрешает оперативно изменять содержимое любого управляющего регистра или ячейки памяти для того, чтобы наблюдать влияние этого на ход программы. Особо стоит рассказать о двух опциях.

Одна из них – выполнение программы до следующей строки программы. Она позволяет при работе в пошаговом режиме, например, за один шаг выполнять всю подпрограмму.

В любой момент можно увидеть точное реальное время выполнения всей программы или любой ее части.

Вы можете задать частоту тактового генератора вашего приложения. Это, конечно, не оказывает никакого влияния на скорость

моделирования, но учитывается программным обеспечением. Необходимо отметить гибкость установки точек прерываний и модификации регистров. Достаточно навести курсор на строку, которую вы хотите определить как точку прерывания, или на регистр, который собираетесь изменить, и впечатать новое значение. Не требуется запоминать адрес прерывания, все видно на экране.

Наконец, довольно редкая, но очень полезная опция: симулятор может переходить назад на величину от одного до ста шагов. Это позволяет, например, увидеть, каким было состояние приложения до остановки программы!

Эмуляторы ClearView 5X и ClearView XX

Эмулятор ClearView 5X предназначен для группы микроконтроллеров PIC 16C5X, в то время как ClearView XX – для остальной части подсемейства PIC 16CXX. Такое деление неудобно, так как вынуждает приобретать два отдельных устройства, чтобы обеспечить работу со всеми микроконтроллерами подсемейства PIC 16CXX.

В том, что касается аппаратуры, данные эмуляторы сходны с симулятором Reflection-5x. Отличие в том, что у эмуляторов ClearView 5X и XX имеется тактовый генератор, способный заменить, в случае необходимости, тактовый генератор приложения. Включается он двумя DIP-переключателями. Сетевой адаптер обоих устройств имеет небольшие размеры и обеспечивает напряжение питания от 7 до 9 В при токе до 1 А. Подключение к приложению осуществляется в соответствии с рис. 2.6, поскольку это настоящие эмуляторы.

Что касается их программного обеспечения, то все сказанное по поводу Reflection-5x действительно и для них. Но поскольку ClearView 5X и XX являются эмуляторами, они работают в реальном времени. Поэтому, если вы, например, тестируете свою программу и в ходе теста осуществляете прогон программы от одного прерывания до другого, он будет происходить с той же скоростью, как если бы на вашем приложении стоял микроконтроллер.

Это остается действительным до частоты тактового генератора 20 МГц для ClearView 5X и до 10 МГц – для ClearView XX.

Естественно, в реальном времени можно имитировать и функционирование портов ввода/вывода, что свойственно только настоящему эмулятору.

Г Л А В А 3

СХЕМНЫЕ РЕШЕНИЯ ИНТЕРФЕЙСОВ МИКРОКОНТРОЛЛЕРОВ

В этой главе:

Параллельные выходы

Параллельные входы

Комбинированное использование портов

Внешняя периферия

Энергонезависимая память с последовательным интерфейсом

Управление аналого-цифровым преобразователем

Большинство устройств на основе микроконтроллеров соединяется с внешними устройствами при помощи портов ввода/вывода. Их назначение может быть различным и определяется конкретным применением микроконтроллера. Так, если ваш микроконтроллер – автомат, управляющий посудомоечной машиной, на его входы будут подаваться сигналы с различных датчиков, а его выходы – управлять двигателями и электромагнитными клапанами. Если же речь идет о программируемом термостате, то порты ввода/вывода микросхемы будут взаимодействовать с клавиатурой, цифровым индикатором, температурным зондом и системой управления нагревательного устройства.

Разнообразие применений портов ввода/вывода не должно вас пугать, поскольку в большинстве случаев в их основе лежит несколько базовых схем, которые и описаны в этой главе.

Ни одно из представленных здесь решений не претендует на звание единственно возможного или самого лучшего. Преимущество микроконтроллеров именно в том и проявляется, что они позволяют получить один и тот же результат разными способами. Отталкиваясь от представленных здесь примеров, вы должны сами определить, какой вариант лучше всего подходит для того или иного приложения.

ПАРАЛЛЕЛЬНЫЕ ВЫХОДЫ

Речь идет о наиболее простом способе использования портов ввода/вывода, как по числу необходимых компонентов, так и по сложности соответствующего программного обеспечения. И именно в устройствах с микроконтроллерами такой вид взаимодействия аппаратурных узлов применяют чаще всего. Поэтому мы с него и начнем.

В качестве выходов параллельные порты могут применяться для управления реле, симисторами, светодиодными или иными цифровыми индикаторами.

Управление светодиодами и оптронами

Управление светодиодами – самое простое, что может встретиться на практике. Схемно оно может немного изменяться в зависимости от используемого микроконтроллера. У некоторых микроконтроллеров выходы рассчитаны на ток большой силы, а поэтому светодиод может быть подключен к ним непосредственно через ограничивающий ток резистор. В качестве примера приведем схему управления четырьмя светодиодами через порт APIC-микроконтроллера 16C5X

(рис. 3.1). Напомним, что допустимая сила тока каждого выхода параллельного порта указанных микросхем составляет 20 мА, что вполне достаточно для зажигания одного светодиода. Но суммарный ток порта не должен превышать 50 мА. В рассматриваемом случае воспользуемся светодиодом на 10 мА, чтобы не превышать этого максимума.

Если требуется, чтобы индикация была хорошо видна, следует использовать светодиоды высокой яркости или применить схему, показанную на рис. 3.2, в которой к выходу микроконтроллера подключен примитивный усилитель на транзисторе.

Ограничительный резистор R_{lim} выбирается в зависимости от требуемой силы тока. Учитывая значение сопротивления в цепи базы транзистора и его коэффициент усиления, через светодиод можно получать ток в 100 мА и более, чего вполне достаточно.

Если один микроконтроллер должен управлять больше чем четырьмя светодиодами, целесообразнее выбрать интегральную микросхему с несколькими усилителями тока. Широко применяются микросхемы ULN2003 и ULN2803 (рис. 3.3), содержащие семь и восемь таких усилителей соответственно. Эти микросхемы изначально выпускались фирмой Sprague, теперь же они предлагаются многими производителями.

Усиление тока в них осуществляют схемы на составных транзисторах (схема Дарлингтона), включающие защитный диод, не используемый в случае, показанном на рис. 3.3, но необходимый при управлении реле, о чем будет рассказано ниже. Каждая схема обеспечивает токи до 500 мА. В цепи базы входного транзистора установлен резистор, ограничивающий входной ток и позволяющий напрямую подключаться к портам микроконтроллера.

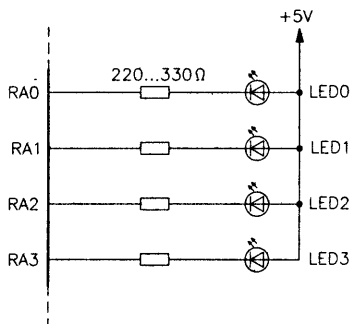


Рис. 3.1

Прямое управление
светодиодами через параллельный порт

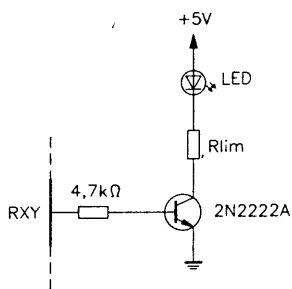


Рис. 3.2

Использование
усилителя на транзисторе

Программа, управляющая данной светодиодной периферией, крайне проста (листинг 3.1). Она составлена для схемы, изображенной на рис. 3.1. Для включения или выключения светодиода достаточно записать в соответствующий бит регистра порта 1 или 0.

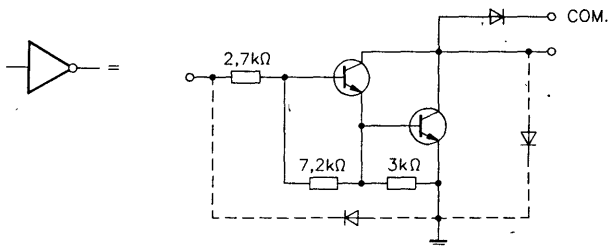
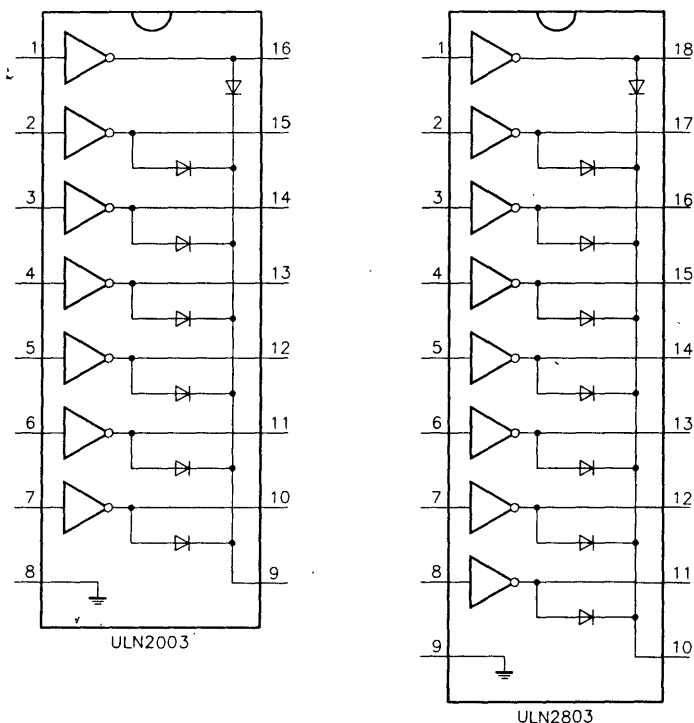


Рис. 3.3

Микросхемы ULM2003 и ULM2803

Листинг 3.1

Управление светодиодами, подключенными к параллельному порту. Эта программа показывает, как управлять светодиодами, подключенными к параллельному порту PIC 16CXX.

```
LED3    =    ra.3
LED2    =    ra.2
LED1    =    ra.1
LED0    =    ra.0
org      0
;
; Определение используемого PIC-микроконтроллера.
;           device pic16c54,xt_osc,wdt_off,protect_off
;           reset      start
;
; Инициализация порта.
iniport  mov    ra,#FFh    ; Записываем 1 во все разряды порта A,
                           ; при этом все светодиоды гаснут.
                           mov    !ra,#F0h    ; Определяем 4 младших разряда порта как выходы.
;
; Управление светодиодами.
comled   clrb   LED0       ; Зажигаем светодиод LED0.
          setb   LED0       ; Выключаем светодиод LED0.
          mov    ra,#11110110b    ; Зажигаем светодиоды LED0 и LED3.
          mov    ra,#11111111b    ; Выключаем все светодиоды.
          mov    ra,#F0h         ; Зажигаем все светодиоды.
          ...и так далее...
end
```

Процедура инициализации порта используется во многих программах. В самом ее начале, прежде чем необходимые разряды будут определены как выходы, во все разряды регистра данных порта записывается 1. Такая запись позволяет избежать замыкания с линиями, имеющими противоположные логические уровни. Конечно, при управлении светодиодами это неважно, но в некоторых устройствах замыкание недопустимо.

Управление оптронной парой осуществляется аналогично управлению светодиодами. Поэтому все, что было сказано выше о светодиодах, применимо и для оптронов. Значения сопротивления ограничительных резисторов для них должны выбираться с учетом максимально допустимой силы тока светодиодов, которые используются оптронами.

Уточним, что микроконтроллером могут управляться оптроны самых разных типов, в том числе и содержащие фотосимисторы. Принцип управления от этого не изменяется. На рис. 3.4 показана схема управления через симисторные оптроны (МОС3040 или МОС3041 фирмы Motorola) током нагрузки до 8 А.

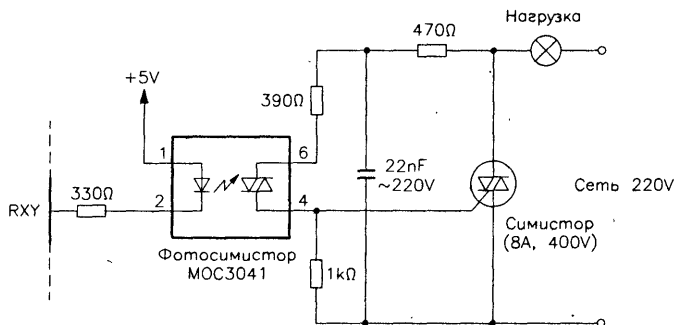


Рис. 3.4

Управление нагрузкой с помощью фотосимистора

Управление реле

Некоторые специалисты по электронике не используют реле, считая их устаревшими компонентами, но во многих устройствах реле незаменимы. Это почти идеальные переключатели, которыми легко управлять и которые обеспечивают превосходную гальваническую развязку между схемой и нагрузкой. Кроме того, реле постоянно совершенствуются: повышается их надежность, уменьшаются размеры. В использовании реле вместе с микроконтроллерами нет, таким образом, ничего анахроничного.

Принцип управления реле очень близок к принципу управления светодиодами. Но учитывая, что даже самые маленькие реле потребляют ток значительной силы, для управления ими требуется внешний транзисторный усилитель. Поэтому, как и в случае со светодиодами, при подключении не более четырех реле лучше использовать отдельные транзисторы, а при большем количестве – микросхемы ULN2003 или ULN2803, выходные токи которых (500 мА) позволяют управлять реле любого типа.

Поскольку реле – компоненты индуктивные, не надо забывать о защитном диоде, включенном в обратном направлении параллельно

обмотке, как это показано на рис. 3.5. Напомним, что в используемых микросхемах уже стоят защитные диоды.

Хотя есть реле с напряжением питания 5 В, часто этого значения бывает недостаточно. Управление реле с большим напряжением успешно осуществляется схемами с транзисторами или с микросхемами ULN2003 или ULN2803. Схемы, изображенные на рис. 3.5 и 3.6, предназначены именно для такого случая и могут управлять реле со стабилизированным или нестабилизированным напряжением, которое для первой схемы (с транзистором) не превышает 30 В, а для второй (с ULN2003 или ULN2803) – 50 В.

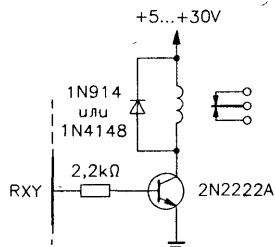


Рис. 3.5

Использование транзисторного усилителя для управления реле

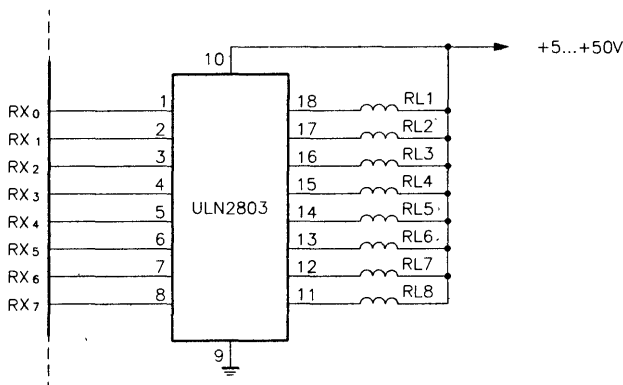


Рис. 3.6

Управление несколькими реле

Прямое управление нагрузкой, питающейся от источника постоянного напряжения

При коммутации «больших» токов в цепи питания микроконтроллера могут оказаться значительные помехи, влияющие на его работу. Несмотря на это здесь приводится схема управления мощной нагрузкой, питаемой постоянным током.

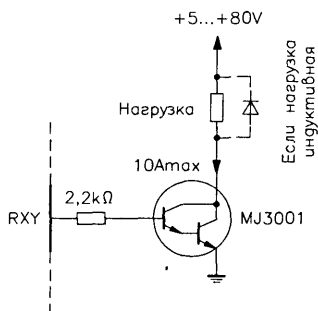


Рис. 3.7

Управление нагрузкой, питающейся от источника постоянного напряжения

Чтобы такая схема успешно функционировала, достаточно включить в нее мощный составной транзистор или интегральную схему, обеспечивающую большое усиление по току. Схема, приведенная на рис. 3.7, использует ИС типа MJ3001 фирмы Motorola (но подойдет и любая аналогичная). Управлять ей можно через любой порт любого микроконтроллера PIC 16CXX, при этом схема обеспечивает коммутацию тока нагрузки величины до 10 А.

Если нагрузка индуктивная (например, реле или электродвигатели), необходимо подключить защитный диод. При управлении двигателями целесообразно использовать типовую схему фильтра, включающую дроссели и конденсаторы, поскольку во время работы двигателя неизбежны сильные импульсные помехи. Конкретная схема фильтра и номиналы ее элементов должны выбираться в зависимости от характеристик двигателя.

Управление светодиодным цифровым индикатором

Цифровой светодиодный индикатор представляет собой несколько объединенных в одном корпусе одиночных светодиодов. Поэтому принципы управления цифровым светодиодным индикатором и простым светодиодом аналогичны. В зависимости от количества индикаторов могут применяться различные варианты схем управления.

Если надо управлять только одним индикатором и имеется достаточное количество портов микроконтроллера, можно применить схему, изображенную на рис. 3.8. Сегменты индикатора управляются каждым портом напрямую. Если индикатор требует значительных токов (из-за большого размера или необходимости обеспечить яркое свечение), лучше включить в схему усилитель.

Если вам нужна индикация с несколькими цифровыми разрядами, придется использовать динамическое управление, предполагающее быструю коммутацию индицируемых разрядов (временное уплотнение). Динамическое управление экономит порты микроконтроллера. Общий принцип работы такой схемы представлен на рис. 3.9.

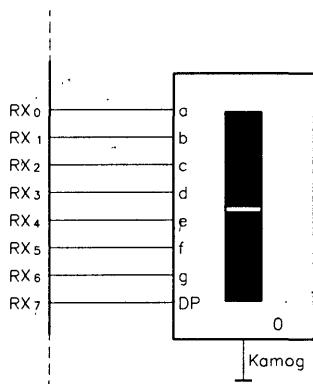


Рис. 3.8

Управление светодиодным семисегментным индикатором

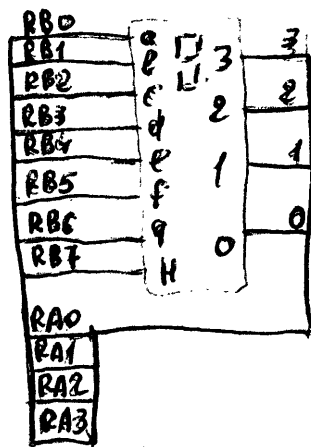


Рис. 3.9

Динамическое управление многоразрядными светодиодными индикаторами

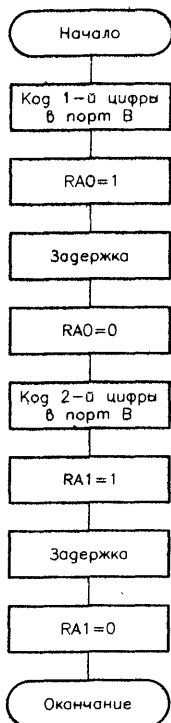


Рис. 3.10

Временная диаграмма работы динамического индикатора

Сегменты индикаторов всегда управляются восьмью линиями параллельного порта (напрямую или через соответствующие усилители), в то время как объединенные катоды (общие катоды) по очереди коммутируются транзисторами через другой порт. Могут быть использованы и индикаторы с общим анодом, при условии замены в схеме транзисторов $n-p-n$ на транзисторы $p-n-p$ и соединения их эмиттеров не с нулевым потенциалом, а с плюсом питания.

Временная диаграмма управления динамической индикацией приведена на рис. 3.10. Порт В в этом случае используется для управления сегментами, управление цифрами ведется через порт А. Индикация цифр осуществляется поочередно, за счет переключения соответствующих разрядов порта А. Для нормального функционирования индикатора микроконтроллер должен обеспечить достаточную частоту переключения цифр, которая не была бы заметна для глаз (не менее 40 Гц, то есть время цикла должно быть не более 25 мс).

Пример программы динамической индикации чисел от 0000 до 9999 приведен в листинге 3.2. Для того чтобы вы смогли наблюдать смену индикации, счет осуществляется с частотой 1 Гц.

Листинг 3.2

```

; *****
; Эта программа предназначена для динамического управления четырьмя семисегментными
; светодиодными индикаторами.
; Она выполняет счет чисел от 0000 до 9999 с интервалом в одну секунду.
; Обновление индикации осуществляется с циклом в 20 мс,
; при этом каждая цифра светится в течение 5 мс.
; Частота переключений задается таймером (RTCC),
; работающим в режиме прерывания с периодом 5 мс.
;
; Программа соответствует инструкции по применению AN557 фирмы Microchip.
;

```

```

;*****
;
;      LIST P=16C71 , F=INHX8M
;
;      Include "picreg.equ"
;
TempC      equ    6x0c      ; Временные регистры общего использования.
TempD      equ    0x0d
TempE      equ    0x0e
Count      equ    0x0f      ; Счетчик.
MsdTime     equ    0x10      ; Байт старших цифр счетчика времени.
LsdTime     equ    0x11      ; Байт младших цифр счетчика времени.
OptionReg   equ    1
PCL         equ    2
BcdMsd      equ    26
Bcd         equ    27
;
;      org    0
;      goto   Start          ; Подпрограмма прерывания.
;
;      org    4
;      goto   ServiceInterrupts
;
Start
;      call   InitPorts
;      call   InitTimers
loop
;      goto   loop
;
InitPorts
;      bsf    STATUS,RP0      ; Банк (страница) 1.
;      movlw  3                ; RA0 - RA3 - цифровых входов/выходов.
;      movwf  ADCON1           ; /
;      clrf   TRISA            ; RA0 - RA4 - выходы.
;      clrf   TRISB            ; RB0 - RB7 - выходы.
;      bcf    STATUS,RP0      ; Банк (страница) 0.
;      clrf   PORT_A           ; Все выходы на низком уровне.
;      clrf   PORT_B           ; /
;      bsf    PORT_A,3
;      return
;
; Частота тактового генератора PIC-микроконтроллера составляет 4,096 МГц.
; частота внутреннего тактового генератора 1,024 МГц.
; За счет делителя на 32 инкрементирования RTCC происходит каждые 31,25 мкс.
; Модуль счета RTCC равен 96, что соответствует периоду прерывания в 5 мс.
;

```

InitTimers

```

    clrf    MsdTime      ; Сброс счетчика времени.
    clrf    LsdTime      ; /
    bsf     STATUS, RP0  ; Выбираем банк 1.
    movlw   B'10000100'  ; Назначаем прескаллеру таймер RTCC.
    movwf   OptionReg    ; Назначаем коэффициент деления 32.
    bcf     STATUS, RP0  ; Банк 0.
    movlw   B'00100000'  ; Разрешаем прерывание таймера.
    movwf   INTCON       ;
    movlw   .96          ; Установка модуля счета таймера.
    movwf   RTCC         ; Начало счета.
    retfie

```

ServiceInterrupts

```

    btfsc   INTCON, RTIF ; Проверка флага прерывания таймера.
    goto    ServiceRTCC ; Да, переход к подпрограмме обработки.
    movlw   B'00100000'  ; Нет, сброс.
    movwf   INTCON
    retfie

```

ServiceRTCC

```

    movlw   .96          ; Инициализация RTCC..
    movwf   RTCC
    bcf     INTCON, RTIF ; Стирание флага прерывания таймера.
    call    IncTimer     ; Приращение счетчика времени.
    call    UpdateDisplay ; Индикация.
    retfie

```

```

; Значение счетчика увеличивается один раз в секунду.
;

```

IncTimer

```

    incf    Count, w     ; Инкрементирование счетчика.
    xorlw   .200         ; = 200?
    btfsc   STATUS, Z    ; Нет, тогда переход.
    goto    DoIncTime    ; Да, приращение счетчика.
    incf    Count        ; Инкрементируем счетчик.
    return

```

DoIncTime

```

    clrf    Count        ; Сброс счетчика.
    incf    LsdTimer, w  ; Пробное инкрементирование байта
                        ; младших цифр.
    andlw   0x0F         ; Маскирование старшего полубайта.
    xorlw   0x0a         ; = 10?
    btfsc   STATUS, Z    ; Нет, тогда переход.
    goto    IncSecondLsd ; Приращение второй цифры.
    incf    LsdTime      ; Приращение первой цифры.
    return

```

IncSecondLsd

```

swapf   LsdTime,w      ; Меняем местами полубайты.
andlw   0x0f            ; Маскирование старшего полубайта.
addlw   1                ; Инкрементируем значение.
movwf   LsdTime
swapf   LsdTime         ; Снова меняем местами полубайты.
xorlw   0x0a            ; = 10?
btfsc   STATUS,Z        ; Нет, тогда переход.
goto    IncThirdLsd     ; Приращение третьей цифры.
return

```

IncThirdLsd

```

clrf    LsdTime
incf    MsdTime,w       ; Пробное приращение байта старших цифр.
andlw   0x0f            ; Маскирование старшего полубайта.
xorlw   0x0a            ; = 10?
btfsc   STATUS,Z        ; Нет, тогда переход.
goto    IncMsd           ; Да, приращение четвертой цифры.
incf    MsdTime         ; Приращение третьей цифры.
return

```

IncMsd

```

swapf   MsdTime,w
andlw   0x0f            ; Маскирование старшего полубайта.
addlw   1                ; Инкрементируем значение.
movwf   MsdTime
swapf   MsdTime
xorlw   0x0a            ; = 10?
btfsc   STATUS,Z        ; Нет, тогда выход.
clrf    MsdTime         ; Сброс байта старших цифр.
return

```

UpdateDisplay

```

movf    PORT_A,w
clrf    PORT_A          ; Выключение всех цифр.
andlw   0x0f
movwf   TempC           ; Сохранение номера цифры в TempC.
bsf     TempC,4          ; Подготовка индикации.
rrf     TempC            ; Определяем следующую цифру.
btfss   STATUS,CARRY     ; c = 1?
bcf     TempC,3          ; Нет, тогда сброс Lsd.
btfsc   TempC,0
goto    UpdateMsd
btfsc   TempC,1
goto    Update3rdLsd
btfsc   TempC,2
goto    Update2ndLsd

```

UpdateLsd

```

movf    LsdTime,w       ; Пересылка Lsd в регистр W.
andlw   0x0f            ; Маскирование старшего байта.
goto    DisplayOut      ; Переход к индикации.

```


Update2ndLsd

```

call    Chk2LsdZero    ; Старшая (Msd) и вторая цифры равны 0?
btfss   STATUS,Z        ; Да, тогда переход.
swapf   LsdTime,w       ; Меняем местами полубайты в W.
andlw   0x0f            ; Маскируем старший полубайт.
goto    DisplayOut      ; Переход к индикации.

```

Update3rdLsd

```

call    ChkMsdZero      ; Msd = 0?
btfss   STATUS,Z        ; Да, тогда переход.
movf    MsdTime,w       ; Пересылка 3-й цифры в W.
andlw   0x0f            ; Маскируем старший полубайт.
goto    DisplayOut      ; Переход к индикации.

```

UpdateMsd

```

swapf   MsdTime,w       ; Пересылка Msd в W.
andlw   0x0f            ; Маскируем младший полубайт.
btfsc   STATUS,Z        ; Msd! = 0, тогда переход
movlw   0x0a            ;

```

DisplayOut

```

call    LedTable         ; Чтение цифр.
movwf   PORT_B           ; Управление LED.
movf    TempC,w          ;
movwf   PORT_A           ;
return

```

Ledtable

```

addwf   5555             ; Модифицируем программный счетчик.
retlw   B'00111111'      ; Цифра 0.
retlw   B'00000110'      ; Цифра 1.
retlw   B'01011011'      ; Цифра 2.
retlw   B'01001111'      ; Цифра 3.
retlw   B'01100110'      ; Цифра 4.
retlw   B'01101101'      ; Цифра 5.
retlw   B'01111101'      ; Цифра 6.
retlw   B'00000111'      ; Цифра 7.
retlw   B'01111111'      ; Цифра 8.
retlw   B'01100111'      ; Цифра 9.
retlw   B'00000000'      ; Цифра выключена (пробел).

```

Chk2LsdZero

```

call    ChkMsdZero      ; Msd = 0?
btfss   STATUS,Z        ; Да, тогда продолжаем.
return  ; В противном случае возврат.
swapf   LsdTime,w       ; Пересылка 2-й цифры Lsd в W.
andlw   0x0f            ; Маскируем младший полубайт.
btfss   STATUS,Z        ; = 0? Тогда продолжаем.
return

```

```

retlw .10 ; В противном случае возвращаем 10.
;
ChkMsdZero
movf MsdTime,w ; Пересылка Msd в W.
btfss STATUS,Z ; = 0? Тогда продолжаем.
return ; В противном случае возврат.
retlw .10 ; Возвращаем 10.
;
end

```

Если нужно управлять многоразрядными индикаторами при помощи небольшого числа выходных портов, то существует два решения. Первое – использовать внешний декодер, а второе – микросхему специализированного контроллера индикации.

В первом случае применяют схему, представленную на рис. 3.11.

Микроконтроллеру не нужно формировать и передавать семи-сегментный код на индикатор, что требовало бы семи или восьми выводов, достаточно передать только двоично-десятичный код (BCD) цифры, для чего хватает всего четырех выводов. Поскольку

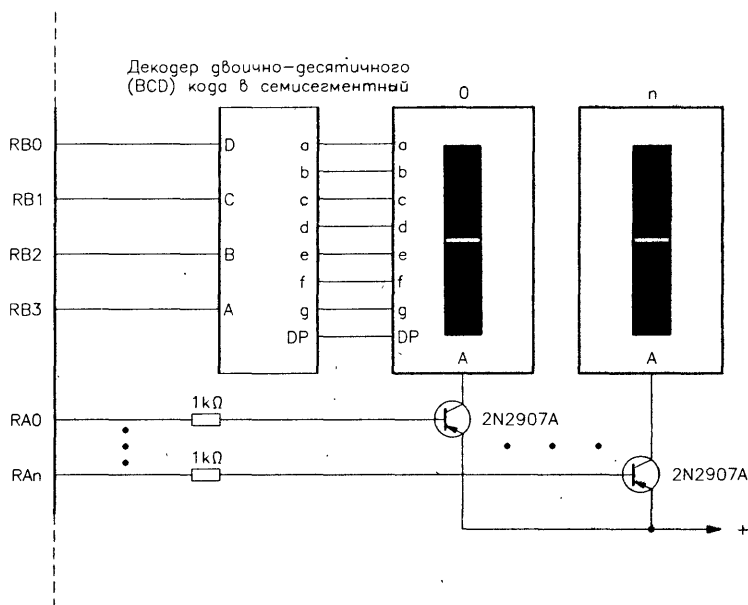


Рис. 3.11

Использование внешнего декодера

микроконтроллер управляет светодиодами не напрямую, то обеспечить необходимый ток нетрудно. Остальная часть схемы аналогична той, что приведена на рис. 3.9, единственное отличие – здесь применен индикатор с общим анодом.

Алгоритм работы микроконтроллера сходен с алгоритмом работы предыдущей схемы. Изменения сведены к замене семисегментного кода кодом BCD и увеличению числа шагов программы, что связано с управлением большим числом цифровых разрядов через соответствующие транзисторы. Если их число слишком велико, то из-за увеличения цикла работы динамической индикации переключения могут стать заметными для глаз. Тогда лучше обратиться к решению, которое будет описано ниже.

Это решение использует микросхему специализированного контроллера индикации, который осуществляет не только декодирование двоично-десятичного кода в семисегментный, но и реализует циклограммы необходимых при динамической индикации переключений. На рынке представлено множество микросхем контроллеров для светодиодных индикаторов. Мы выбрали в качестве примера контроллер MC14499 фирмы Motorola, достаточно распространенный и недорогой.

Его внутренняя структура показана на рис. 3.12. Как видите, эта микросхема может управлять четырьмя семисегментными индикаторами через последовательный интерфейс, включающий три линии: DATA, CLOCK и EN. Обратите внимание, что передача информации по интерфейсу происходит в синхронном режиме и скорость обмена, определяющаяся частотой тактовых импульсов на линии CLOCK, может быть произвольной и даже переменной. Это значительно упрощает программирование.

Микросхема реализует принцип динамического управления индикаторами, что предполагает матричный доступ к каждому сегменту. Это сокращает общее число необходимых линий и соответственно выводов микросхемы, используемых для управления индикатором. Упрощается и программное обеспечение микроконтроллера: его задача, в части поддержки индикации, сводится к посылке на микросхему последовательности цифр, которые надо вывести на индикатор.

Если приложение требует более четырех индикаторов, можно использовать несколько ИС MC14499, как это показано на рис. 3.13.

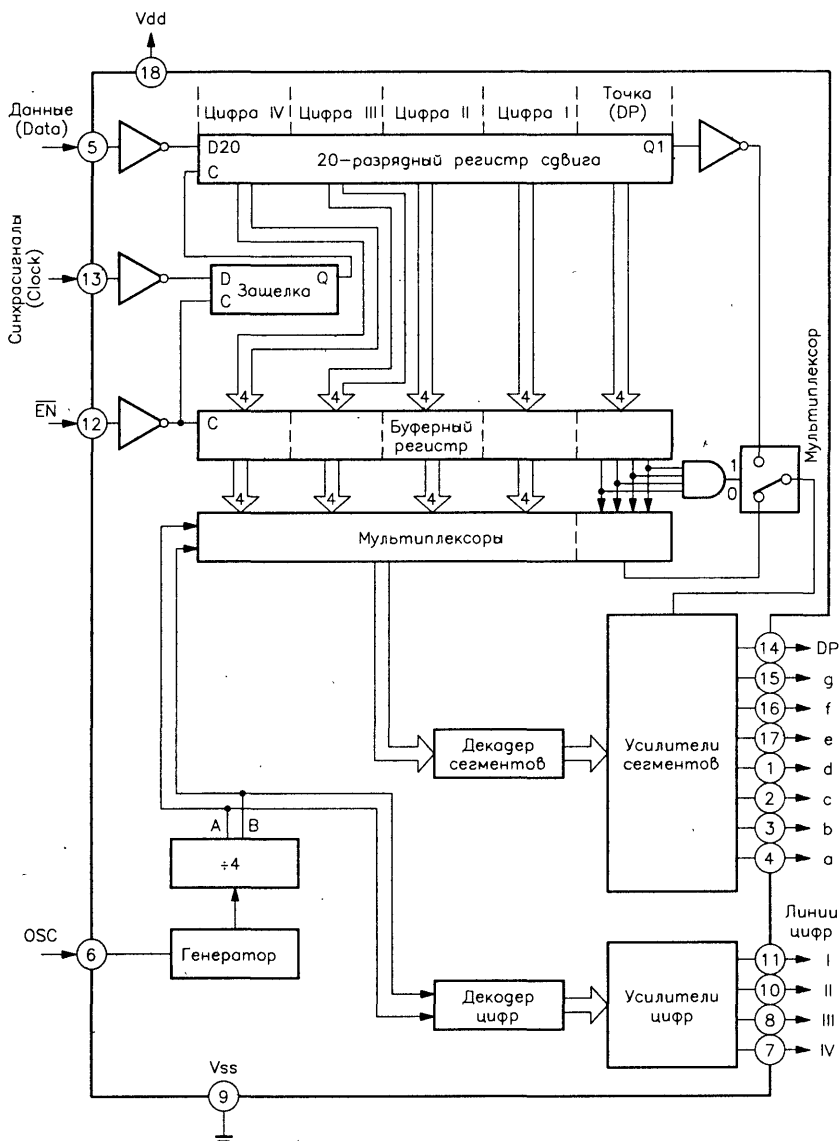


Рис. 3.12

Структурная схема контроллера динамической индикации MC14499

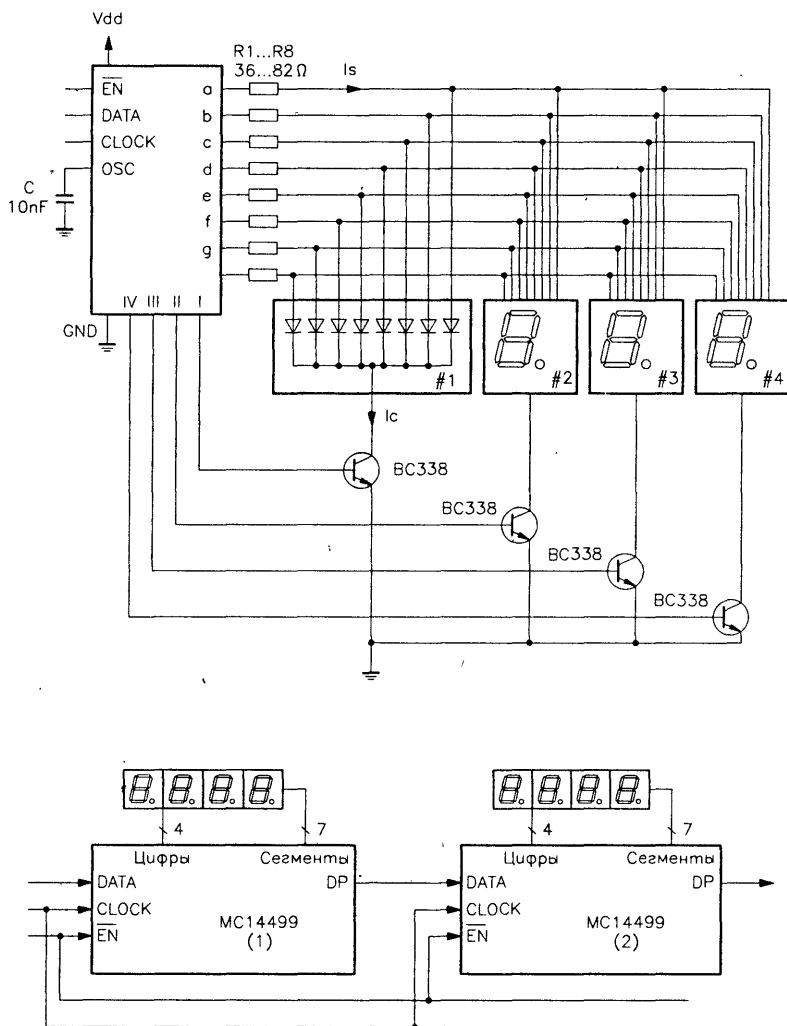


Рис. 3.13

Использование для индикации одной или нескольких ИС MC14499

Временные диаграммы пересылки данных к ИС MC14499 представлены на рис. 3.14. Данные, передаваемые по линии DATA, записываются в микросхему по заднему фронту сигнала синхронизации на линии CLOCK.

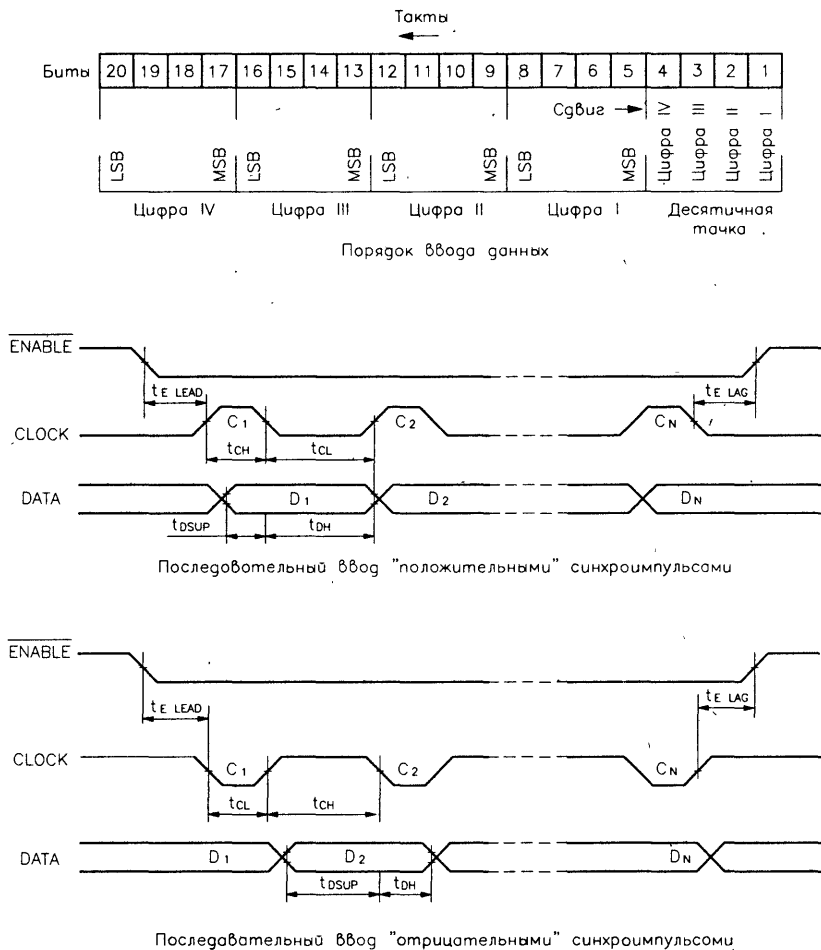


Рис. 3.14

Временные диаграммы передачи данных на ИСМС14499

Пример управления ИС МС14499 через порт микроконтроллера подсемейства PIC 16CXX показан на рис. 3.15.

Простая программа (см. листинг 3.3) осуществляет индикацию четырех цифр, кодированных в форме BCD и объединенных попарно в переменные DON1 и DON2. Обратите внимание на способ формирования

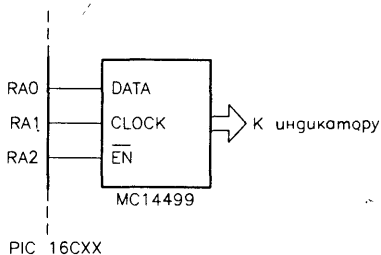


Рис. 3.15

Управление ИС MC14499 с помощью микроконтроллера PIC 16CXX

нем, что MC14499 не накладывает никаких временных ограничений на процесс передачи.

Листинг 3.3

; Управление ИС контроллера динамической светодиодной индикации MC14499.
; В качестве примера демонстрируется вывод четырех цифр, содержащихся в DON1 и DON2.

```
D      =      ra.0      ; Линия DATA ИС MC14499.
CLK    =      ra.1      ; Линия тактовых сигналов ИС MC14499.
EN      =      ra.2      ; Линия выбора кристалла ИС MC14499.
```

```
; Определение переменных величин.
org      8
```

```
temp ds      1      ; Временный регистр DON1.
DON1 ds      1      ; Первая группа из двух цифр.
DON2 ds      1      ; Вторая группа из двух цифр.
clocks ds    1      ; Номер такта для подпрограммы Shout.
```

```
; Определение используемого PIC.
```

```
device pic16c54,xt_osc,wat_off,protect_off;
reset      start      ;
```

```
; Происхождение программы в ПЗУ в 0.
```

```
org      0
start mov    ra,#FFh      ; Установка всех разрядов порта A в 1.
mov       !ra,#0          ; Определяем RA как порт вывода.
boucle mov  temp,#0        ; Обнуление.
mov       clocks,#4        ; Готовимся к выводу полубайта.
call      sortie
mov       temp,DON1        ; Загружаем первую переменную.
```

последовательных сигналов на линии DATA и CLOCK. Эту задачу выполняет подпрограмма Shout, которая, сканируя содержимое временного регистра temp, осуществляет вывод из него байта бит за битом. Генерация сигналов синхронизации передачи CLOCK сводится всего лишь к выполнению инструкций, поочередно устанавливающих уровни логических единиц и нулей на используемую для этой цели линию порта. Подчерк-

```

mov    clocks,#8      ; Готовимся к выводу байта.
call   sortie
mov    temp,DON2      ; Загружаем вторую переменную.
mov    clocks,#8      ; Готовимся к выводу байта.
call   sortie
goto   boucle         ; Повторяем цикл.

sortie  clrb    EN      ; Выбор ИС MC14499.
        call   Shout    ; Обращение к подпрограмме вывода.
        setb   EN      ; Отключение MC14499.
        ret

; Эта подпрограмма осуществляет вывод данных в последовательной форме на ИС MC14499.
; Перед выводом информация должна быть помещена в регистр temp, число выводимых битов
; задано в clocks, а на выводе EN должен быть установлен низкий уровень.
; В конце операции на EN надо установить высокий уровень.

Shout   rl      temp      ; Сдвиг temp влево.
        movb   D,c        ; Устанавливаем бит переноса на линию DATA.
        setb   CLK        ; Генерируем синхриимпульс.
        nop     ; Удерживаем его высокий уровень еще 500 нс.
        clrb   CLK        ; Сбрасываем линию CLOCK в 0.
        djnz   clocks,Shout ; Декрементируем переменную clocks.
        ret
end

```

Управление индикаторами на жидких кристаллах

Когда необходимо обеспечить автономную работу аппаратуры и уменьшить ее мощность потребления, *индикаторы на жидких кристаллах* – идеальное решение. Однако жидкокристаллическими индикаторами (*ЖКИ*) управлять немного труднее, чем индикаторами на светодиодах. Управление сегментами светодиодных индикаторов осуществляется за счет подачи сигналов высоких или низких логических уровней, в то время как управление одним или несколькими общими электродами (back plane) жидкокристаллических индикаторов гораздо деликатнее, особенно в моделях, включающих множество цифр или знаков, где приходится прибегать к двойному или даже тройному мультиплексированию.

Наиболее простое решение – использовать внешний контроллер, приспособленный для управления именно данным индикатором. Но они довольно дорогие. Поэтому лучше управлять индикаторами ЖКИ с помощью программного обеспечения прямо с PIC-микроконтроллера. Полный листинг такой программы будет дан ниже.

Сначала рассмотрим, как в качестве внешнего контроллера для управления ЖКИ, имеющим до 48 сегментов, используется ИС типа MC145000 фирмы Motorola (рис. 3.16).

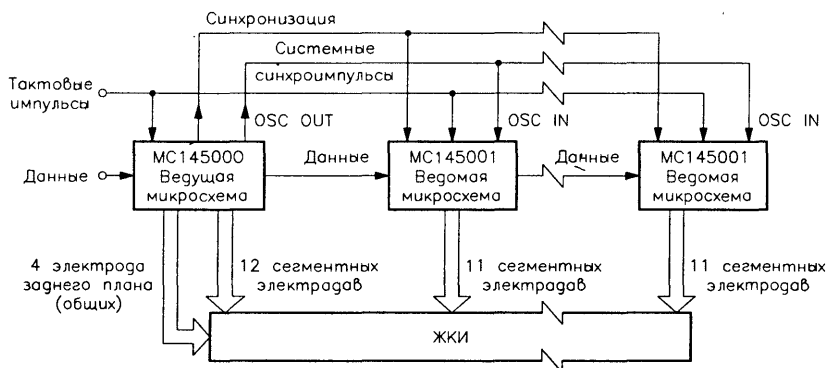


Рис. 3.16

Схема применения контроллера MC145000

MC145000 предполагает матричное управление индикатором и имеет четыре выхода для общих электродов (back plane) и двенадцать для сегментных (фронтальных). Если этого недостаточно, можно к схеме MC145000 подсоединить подчиненную микросхему MC145001.

При управлении светодиодными индикаторами посредством ИС MC14499 для передачи информации достаточно двух линий: линии данных и линии синхронизации. Задача программного обеспечения сводится к пересылке по этому интерфейсу данных для отображения на индикаторе. Микросхема MC145000 предоставляет дополнительные возможности. Она позволяет программировать таблицу *знакогенератора*, используемую для индикации данных. Используя сегментное представление необходимых знаков, можно отобразить самые разнообразные знаки. Эта таблица перекодировки (знакогенератор) должна быть загружена в контроллер MC145000 прежде, чем ему станут посылаться данные. Для загрузки таблицы и отправки данных используются одни и те же линии передачи.

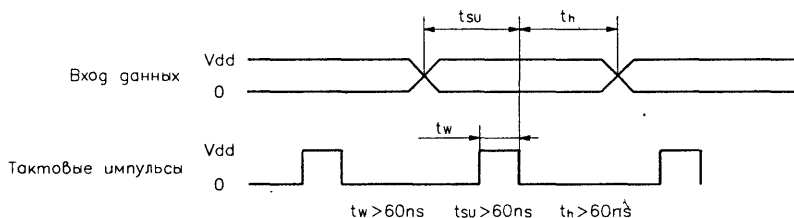


Рис. 3.17

Временные диаграммы обмена с MC145000

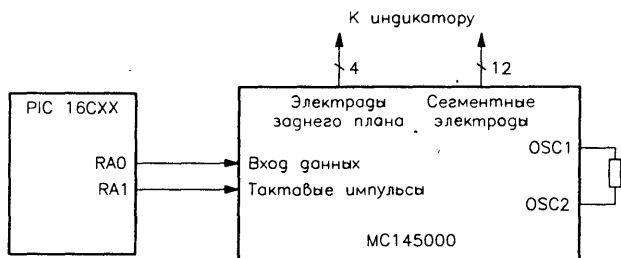
Временные диаграммы передачи данных по последовательному интерфейсу ИС MC145000 представлены на рис. 3.17. Принцип передачи данных очень прост. Данные, поступившие по линии DATE, принимаются бит за битом по заднему фронту импульса синхронизации. После завершения передачи данные отображаются MC145000 в автоматическом режиме.

Схема, таблица перекодировки данных, управляющие коды представлены на рис. 3.18. Программа, управляющая обменом, для этой схемы не приведена, так как она аналогична программе для микросхемы MC14499. Отличия имеются лишь на уровне передачи команд и на уровне кодирования данных, посылаемых микросхеме.

Остановимся на формах сигналов, которые подаются на индикатор.

Индикатор на жидких кристаллах имеет задний (общий) электрод и электроды сегментов в количестве, соответствующем числу отдельных элементов индикации в данном устройстве.

Чтобы жидкий кристалл индикатора не поляризовался, на электроды не должно поступать постоянное напряжение, а только импульсные сигналы, как показано на рис. 3.19. В этих условиях только разница потенциалов между сигналами, приложенными к общему электроду (COM), и потенциалами сегментов (SEG), определяет, станет ли непрозрачным (темным) тот или иной сегмент. Переключение сегмента из одного состояния в другое происходит, если разность потенциалов оказывается выше порогового значения V_s , определяемого типом индикатора.



Отображаемые знаки	Семисегментные коды знаков
0	D7
1	06
2	E3
3	A7
4	36
5	B5
6	F5
7	07
8	F7
9	B7
A	77
b	F4
C	D1

Отображаемые знаки	Семисегментные коды знаков
d	E6
E	F1
F	71
P	73
Y	B6
H	76
U	D6
L	D0
Пробел	00
- (тире)	20
= (равно)	A0
n	64
r	60
° (степень)	33

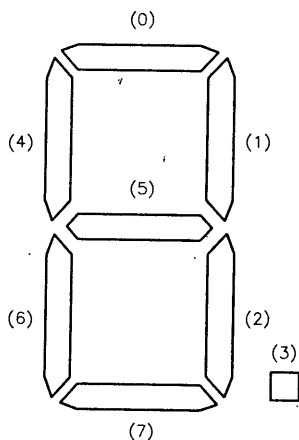


Рис. 3.18

Использование PIC-микроконтроллера
16CXX для управления MC145000

Мультиплексированное управление индикатором на жидких кристаллах осуществляется матрицей, показанной на рис. 3.20, где каждый сегмент находится на пересечении общих и сегментных электродов.

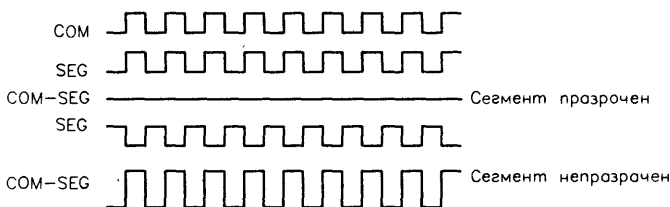


Рис. 3.19

Временные диаграммы работы жидкокристаллического индикатора

На общие электроды подаются сигналы фиксированной формы (что видно в верхней части рис. 3.21), на сегменты же, чтобы сделать их прозрачными или непрозрачными, надо подать сигналы изменяемой формы.

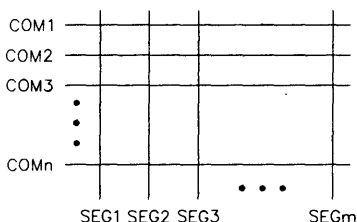


Рис. 3.20

Принцип матричного управления жидкокристаллическим индикатором

Как показано в нижней части рис. 3.21, к сегментам надо прикладывать напряжение равное $V_s + V_d$, если они должны быть непрозрачными, и $V_s - V_d$, если требуется прозрачный сегмент.

Генерация трехуровневых сигналов PIC-микроконтроллерами, как, впрочем, и любыми другими логическими схемами, невозможна, поскольку они питаются одним напряжением и функционируют в двухуровневой логике. Таким образом, для управления ЖКИ необходимо искусственно создавать среднюю точку.

Пример управления ЖКИ изображен на рис. 3.22. Используемый в ней индикатор включает четыре цифры. Он имеет два общих электрода и две группы сегментных электродов, на две цифры каждая. Искусственная средняя точка создается резисторами $R1 - R4$ и, в соответствии с сигналами, формируемыми на линиях портов $RA0$ и $RA1$ (аналогично $RA2$ и $RA3$), позволяет получать напряжения V_{DD} , V_{SS} и $(V_{DD} - V_{SS})/2$.

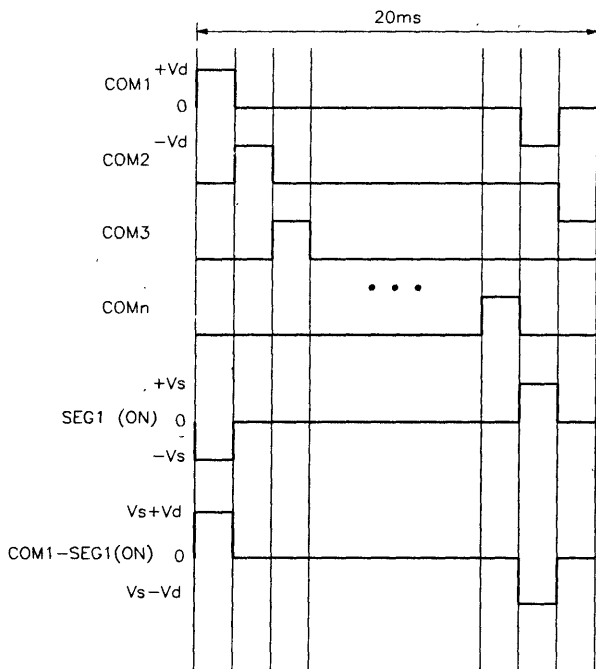


Рис. 3.21

Временные диаграммы управления ЖКИ с мультиплексированием

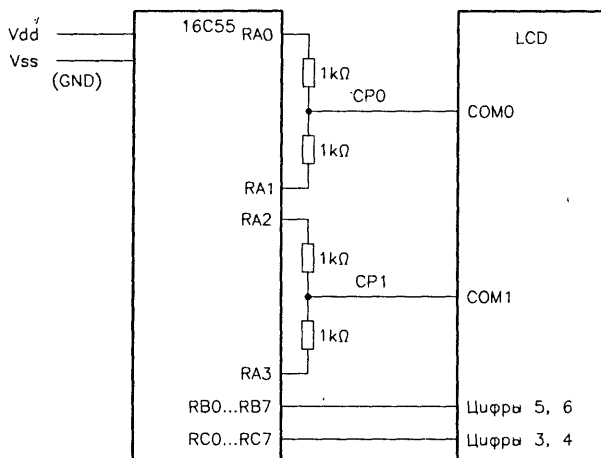


Рис. 3.22

Использование PIC-микроконтроллера для управления ЖКИ

На рис. 3.23 показано, как генерируются сигналы, подаваемые на электроды COM0 и COM1, и четыре соответствующих состояния.

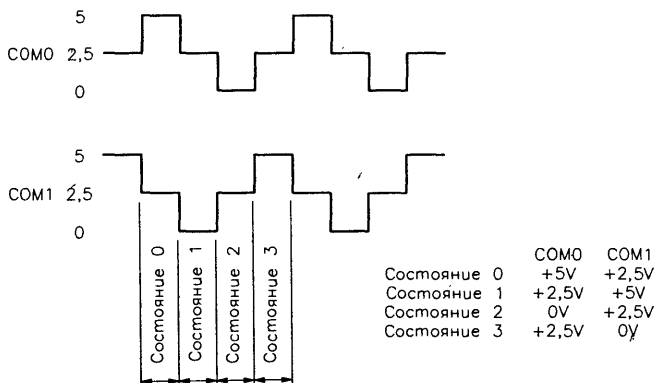


Рис. 3.23

Временные диаграммы работы схемы

В табл. 3.1 представлен способ кодирования различных знаков.

Таблица 3.1

Причины кодирования цифр для ЖКИ

Цифра	COM0 SEG0				COM1 SEG1				COM0 SEG2				COM1 SEG3			
	F	E	D	P	A	G	C	B	F	E	D	P	A	G	C	B
0	0	0	0	1	0	0	1	0	1	1	1	0	1	1	0	1
1	1	1	1	1	1	0	1	0	0	0	0	0	0	1	0	1
2	1	0	0	1	0	0	0	1	0	1	1	0	1	1	1	0
3	1	1	0	1	0	0	0	0	0	0	1	0	1	1	1	1
4	0	1	1	1	1	0	0	0	1	0	0	0	0	1	1	1
5	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1	1
6	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1
7	1	1	1	1	0	0	1	0	0	0	0	0	1	1	0	1
8	0	0	0	1	0	0	0	0	1	1	1	0	1	1	1	1
9	0	1	0	1	0	0	0	0	1	0	1	0	1	1	1	1
a	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1	1
b	0	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1
c	1	0	0	1	1	1	0	1	0	1	1	0	0	0	1	0
d	1	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1
e	0	0	0	1	0	1	0	1	1	1	1	0	1	0	1	0
f	0	0	1	1	0	1	0	1	1	1	0	0	1	0	1	0

Программа, представленная листингом 3.4, позволяет генерировать необходимые сигналы и управлять ЖКИ.

Листинг 3.4

```
List                               C=132,n=0,p=16c55,r=dec
;*****
; Управление ЖКИ с помощью PIC 16CXX.
;
; Программа соответствует инструкции по применению AN563 фирмы Microchip.
;
;*****
; Определение констант.
pic54      equ    0x1ff
pic55      equ    0x1ff
pic56      equ    0x3ff
pic57      equ    0x7ff

rtcc       equ    1           ; f1.
pc         equ    2           ; f2.
status     equ    3           ; f3.
fsr        equ    4           ; f4.

porta      equ    5           ; f5.
portb      equ    6           ; f6.
portc      equ    7           ; f8.
; Регистры "реального времени".

currentState equ    8           ;
msTimer     equ    currentState+1 ; Интервал в миллисекундах.
sTimerLow   equ    msTimer+1    ; Младший байт второго таймера.
sTimerHigh  equ    sTimerLow+1  ; Старший байт второго таймера.
digit56     equ    sTimerHigh+1 ;
digit34     equ    digit56+1    ;

; Различные определения.

FIVEMSEC    equ    96           ; Если частота кварцевого генератора 4,096 MHz.
w           equ    0           ;
f           equ    1           ;

z           equ    2

; Определение битов регистров.
;*****
; Назначение портов.
; porta -   bit0:    общий электрод 0      *
;           bit1:    общий электрод 0      *
;           bit2:    общий электрод 1      *
;           bit3:    общий электрод 1      *
```



```

swapf    sTimerLow, w    ;
andlw     0xf             ; Маскирование цифры 5.
call      Table           ;
movwf     digit56         ;
swapf     digit56, f      ;
movf      sTimerLow, w    ;
andlw     0xf             ; Маскирование цифры 6.
call      Table           ;
iorwf     digit56, f      ;
swapf     sTimerHigh, w   ;
andlw     0xf             ; Маскирование цифры 3.
call      Table           ;
movwf     digit34         ;
swapf     digit34, f      ;
movf      sTimerHigh, w   ;
andlw     0xf             ; Маскирование цифры 4.
call      Table           ;
iorwf     digit34, f      ;
movf      digit34, w      ; Индикация цифр 3 и 4.
movwf     portc           ;
movf      digit56, w      ;
movwf     portb           ; Индикация цифр 5 и 6.
endm
org        0

```


; Инициализация портов А, В и С и RTCC. Записываем данные
; и определяем назначение отдельных портов (входы/выходы).

Initialize

```
movlw    00000001b ; Запись данных в порт А.
movwf    porta      ;
movlw    00001000b ; Определение входов/выходов порта А.
tris     porta
movlw    00000000b
movwf    portb      ; Обнуляем порт В.
movlw    00000000b
tris     portb      ; Все линии порта В - выходы.
movlw    00000000b ;
movwf    porte      ; Обнуляем порт Е.
movlw    00000000b ;
tris     porte      ; Все линии порта Е - выходы.
movlw    0x04       ;
movlw    FIVEMSEC   ; RTCC = 5 мс.
movwf    rtcc       ;
movlw    4
movwf    currentState
movlw    0xd
movwf    msTimer    ; Установка таймера.
clrf     sTimerLow  ; Сброс второго таймера.
clrf     sTimerHigh
retlw    0
```

; Проверяем таймер на 0.

; Если не 0, ожидаем.

Timer_Check

```
movf     rtcc, w
btfss    status, z
goto     Timer_Check
movlw    FIVEMSEC
movwf    rtcc
decfsz   msTimer, f
goto     Update_Backplane
incfsz   sTimerLow ; Индикация второго таймера.
goto     Update_Backplane
incf     sTimerHigh, f
```

; RA0 и RA1 контролируют напряжение на COM 0.

; RA2 и RA3 контролируют напряжение на COM 1.

; Возможны четыре различных состояния.

;

; Состояние 0 - cp0 = +5 В ra0=1, ra1=x

; cp1 = +2.5 В ra2=1, ra3=0

; Состояние 1 - cp0 = +2.5 В ra0=1, ra1=0

; cp1 = +5 В ra2=1, ra3=x

```
; Состояние 2 - cp0 = 0 B      ra0=0, ra1=x
;                               cp1 = +2.5 B  ra2=1, ra3=0
; Состояние 3 - cp0 = +2.5 B   ra0=1, ra1=0
;                               cp1 = 0 B     ra2=0, ra3=x
```

Update_Backplane

```
    clrwdt          ; Сброс сторожевого таймера.
    decf    currentState, w ; Индикация W.
    andlw    0x03      ; Выделяем биты 0 и 1.
    movwf    currentState
    addwf    pc, f
    goto     State3
    goto     State2
    goto     State1
    goto     State0
```

```
;
; Состояние 0.
State0
```

```
    UpdateState      State0, S0_Table
    movlw    00000101b
    movwf    porta
    movlw    00000110b
    tris     porta
    retlw    8
```

S0_Table

```
; Выбираем данные из таблицы.
```

```
    addwf    pc, f ; Добавляем смещение к программному счетчику
    retlw    0100b ; 0
    retlw    1100b ; 1
    retlw    0010b ; 2
    retlw    0000b ; 3
    retlw    1000b ; 4
    retlw    0001b ; 5
    retlw    1111b ; 6
    retlw    0100b ; 7
    retlw    0000b ; 8
    retlw    0000b ; 9
    retlw    0000b ; a
    retlw    1001b ; b
    retlw    1011b ; c
    retlw    1000b ; d
    retlw    0011b ; e
    retlw    0011b ; f
```

```
;
; Состояние 1.
State1
```

```
    UpdateState      State1, S1_Table
    movlw    00000101b
```

```

movwf    porta
movlw    00001000b
tris     porta
retlw    0

```

; Для состояния 1 выбираем данные из таблицы.

S1_Table

```

addwf    pc, f      ;
retlw    0001b      ; 0
retlw    1111b      ; 1
retlw    1001b      ; 2
retlw    1101b      ; 3
retlw    0111b      ; 4
retlw    0101b      ; 5
retlw    1111b      ; 6
retlw    1111b      ; 7
retlw    0001b      ; 8
refclw   0101b      ; 9
retlw    0011b      ; a
retlw    0001b      ; b
retlw    1001b      ; c
retlw    1001b      ; d
retlw    0001b      ; e
retlw    0011b      ; f

```

; Состояние 2.

State2

```

UpdateState    State2, S2_Table
movlw    00000100b
movwf    porta
movlw    00000010b
tris     porta
retlw    0

```

; Для состояния 2 выбираем данные из таблицы.

S2_Table

```

addwf    pc, f      ;
retlw    1011b      ; 0
retlw    0011b      ; 1
retlw    1101b      ; 2
retlw    1111b      ; 3
retlw    0111b      ; 4
retlw    1110b      ; 5
retlw    1110b      ; 6
retlw    1011b      ; 7
retlw    1111b      ; 8
retlw    1111b      ; 9
retlw    1111b      ; a
retlw    0110b      ; b

```

```
retlw    0100b    ; c
retlw    0111b    ; d
retlw    1100b    ; e
retlw    1100b    ; f
```

; Состояние 3.

State3

```
UpdateState    State3, S3_Table
movlw          00000001b
movwf          porta
movlw          00001000b
tris           porta
retlw          0
```

; Для состояния 3 выбираем данные из таблицы.

S3_Table

```
addwf          pc, f    ; .
retlw          1011b    ; 0
retlw          0000b    ; 1
retlw          0110b    ; 2
retlw          0010b    ; 3
retlw          1000b    ; 4
retlw          1010b    ; 5
retlw          1110b    ; 6
retlw          0000b    ; 7
retlw          1110b    ; 8
retlw          1010b    ; 9
retlw          1100b    ; a
retlw          1110b    ; b
retlw          0110b    ; c
retlw          0110b    ; d
retlw          1110b    ; e
retlw          1100b    ; f
```

; Основная программа.

Start

```
Repeat    call    Initialize
          call    Timer_Check
          goto    Repeat
          org      pic55
```

System_Reset

```
goto      Start
END
```

Еще одной причиной применения индикаторов на жидких кристаллах является их большая информативность. Она проявляется, когда потребителю недостаточно индикации нескольких цифр или простых символов. Сегодня на рынке можно найти совсем недорогие

модели ЖКИ, обеспечивающие отображение одного или двух рядов от 16 до 40 знаков, причем это могут быть не только цифры, но и алфавитно-цифровые символы, имеющиеся, например, на клавиатуре компьютера.

Подобные индикаторы всегда поставляются в виде маленькой печатной платы, включающей собственно ЖКИ и схему управления. Электронная схема облегчает управление индикатором со стороны микроконтроллера, которому уже не приходится формировать сложные сигналы, как в предыдущем примере.

Схема, изображенная на рис. 3.24, использует индикатор фирмы Hitachi, хотя может быть применен любой другой индикатор этого типа, так как сигналы интерфейса у различных марок почти идентичны.

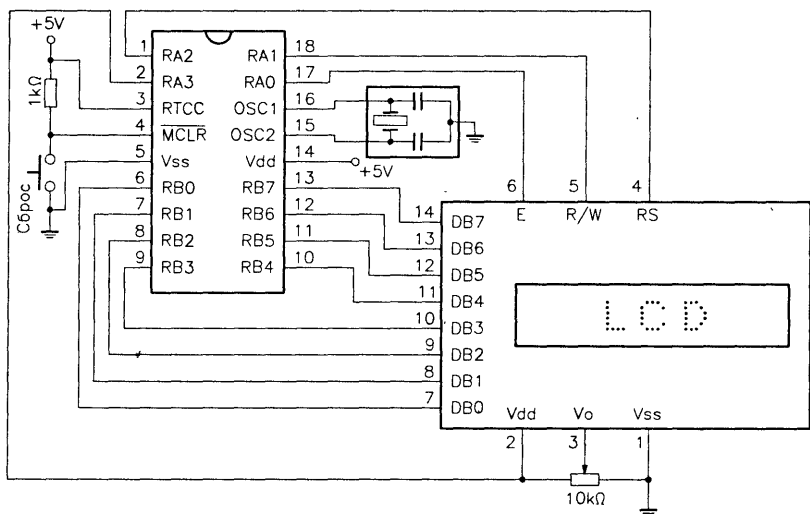


Рис. 3.24

Управление «интеллектуальным» ЖКИ с помощью PIC-микроконтроллера 16CXX

Данные на индикатор подаются по восьми линиям порта В (DB0 – DB7) PIC-микроконтроллера 16C54, а сигналы трех линий управления формируются через порт А:

- ♦ линия E (Enable) – высокий уровень сигнала на данной линии разрешает выполнение операции обмена, при этом индикатор

может получать команды или данные. Нулевой уровень запрещает доступ к индикатору;

- ♦ линия R/W (Read/Write) – указывает тип операции при обращении к индикатору (запись или чтение данных). Индикатор имеет внутренний регистр состояния, информация из которого может быть считана;
- ♦ линия RS (Register Select) – определяет тип передаваемой информации: команды (RS = 0) или данные (RS = 1).

В режиме передачи данных индикатор принимает и отображает знаки кода ASCII, полученные от микроконтроллера. Позиция курсора при этом изменяется автоматически.

Индикатор может выполнять некоторое количество команд, обеспечивающих стирание отдельных символов, полное стирание всей информации, указание позиции курсора и т.д. Эти команды ускоряют управление индикатором. Некоторые модели индикаторов располагают даже памятью, где хранится конфигурация отображаемых знаков (знакогенератор). С ее помощью вы можете изменять форму знаков.

Диалог с таким индикатором предельно прост. Запись информации в индикатор происходит, например, в следующей последовательности:

- ♦ установить в ноль сигнал линии R/W;
- ♦ указать состояние линии RS, которое должно определять тип передаваемой информации (данные или команда);
- ♦ установить код данных или команды на шину DB0 – DB7;
- ♦ установить уровень логической единицы на линию E, разрешая индикатору принять информацию;
- ♦ обнулить сигнал линии E, заканчивая обмен.

Этот процесс может повторяться многократно, но, учитывая «медлительность» ЖКИ, между двумя последовательными передачами команд или данных должна быть некоторая пауза. Ее типовая длительность: от 100 мкс для операции простой передачи индицируемых данных до 5 мс для более сложных операций.

Команда считывания выполняется за одну микросекунду. Она использована в листинге 3.5 для того, чтобы управляющий PIC-микроконтроллер не тратил время на бесполезное ожидание, если несколько знаков и/или команд должны быть посланы последовательно друг за другом.

Программа, представленная листингом 3.5, предназначена для схемы, изображенной на рис. 3.25⁴. Она обнуляет индикатор, загружает в его память четыре специальных символа и выводит на экран слово *Parallax* в прямом и зеркальном представлении благодаря зеркально отраженным буквам, которые будут заранее помещены в его память.

Листинг 3.5

```
; Управление "интеллектуальным" ЖКИ.
; Программа соответствует руководству по применению фирмы Parallax.
; Предназначена для ЖКИ фирмы Hitachi или эквивалентного.
; Выводит на индикатор четыре знака и осуществляет индикацию слова
; Parallax в прямом и обратном порядке в зеркальном отражении
; благодаря специальным знакам. Программа включает подпрограмму blip_E,
; которая управляет обменом и отправляет данные или команды на индикатор.
```

```
LCD_pwr = ra.3 ; Включение питания индикатора (5 В).
RS      = ra.2 ; 0 = команда, 1 = данные.
RW      = ra.1 ; 0 = запись, 1 = считывание.
E       = ra.0 ; 0 = запрет приема/передачи.
          ; 1 = разрешение приема/передачи.
```

```
Data    = rb ; Данные ЖКИ.
Count   = 16 ; Счетчик знаков.
Char_cnt = 32 ; Счетчик байтов.
; Определение констант, используемых в командах ЖКИ.
; Изменится, если будет использован индикатор другого типа.
```

```
Clear    = , 1 ; Сброс индикации.
Home     = 2 ; " home "
shift_l  = 24 ; Сдвиг влево.
shift_r  = 28 ; Сдвиг вправо.
crsr_l   = 16 ; Курсор влево.
crsr_r   = 20 ; Курсор вправо.
Blink_c  = 11 ; Мигание символа, что указывает
          ; положение курсора.
No_crsr  = 8 ; Гашение курсора.
```

```
; Определение переменных величин.
```

```
org 8
temp    ds 1 ; Временный счетчик.
temp2   ds 1 ; Передача данных в blip_E.
Counter ds 1 ; Индекс.
org 0
```

```
; Определение используемого PIC.
```

```
device pic16c54,xt_osc,wdt_off,protect_off
reset start
```

```

start    mov    ra, #0                ; Инициализация порта А.
          mov    rb, #0                ; Инициализация порта В.
          mov    !ra, #0h              ; Все линии порта А – выходы.
          mov    !rb, #0h              ; Все линии порта В – выходы.
          setb   LCD_pwr               ; Включение питания ЖКИ.
          call   wait                 ; Ожидание.
          mov    temp2, #00110000b     ; Инициализация LCD.
          call   blip_E
          mov    temp2, #00001110b
          call   blip_E
          mov    temp2, #00000110b
          call   blip_E
          mov    temp2, #01000000b     ; Запись знаков начиная с адреса 0.
          call   blip_E
          setb   RS                   ; 1 на RS для передачи данных.
          mov    counter, #0
:stuff    mov    w, counter
          call   my_chars              ; Следующий байт.
          mov    temp2, w              ; Запись байта.
          call   blip_e
          inc    counter
          cjb    counter, #char_cnt, :stuff
          clrb   RS                   ; 0 на RS для передачи команд.
          mov    temp2, #10000000b
          call   blip_E
          setb   RS

send_msg  mov    counter, #0           ; Передача последовательности знаков.
:loop     mov    w, counter
          call   msg
          mov    temp2, w
          call   blip_E
          inc    counter
          cjb    counter, #count, :loop
:loop2    jmp    loop2

```

; Бесконечный цикл, сброс для выхода.

; Запись данных содержится в переменной temp2 индикатора.

```

blip_E    movb   pa2, RS              ; Запись состояния RS.
          clrb   RS                   ; 0 на RS в 0 для передачи команд.
          setb   RW                   ; 1 на RW для считывания.
          mov    !rb, #255            ; Порт В – входной.
          clrb   RS                   ; 0 на RS в 0 для передачи команд.
          mov    temp, data           ; Принятие из ЖКИ данные в temp.
          clrb   E                    ; Запрет обмена.
          snb    temp.7               ; ЖКИ занят?

```



```

        jmp      :loop      ; Если да, повторяем.
        movb    RS, pa2     ; В противном случае передаем данные
                               ; или команды.

        clrb    RW
        mov     !rb, #0
        mov     data, temp2
        setb    E
        nop
        clrb    E
        ret

wait     mov     temp2, #200 ; Сброс ЖКИ.
:loop    djnz    temp; :loop ;
        djnz    temp2; :loop ;
        ret

msg      jmp     pc+w        ; Таблица символов для индикации.
        retw

'P', 'a', 'r', 'a', 'l', 'l', 'a', 'x', 2, 0, 0, 2, 1, 2, 3
My_chars jmp     pc+w        ; Таблица специальных символов.
        retw     6, 4, 4, 4, 4, 14, 0 ; l "зеркальный вид".
        retw     0, 0, 13, 19, 1, 1, 1, 0 ; r "зеркальный вид".
        retw     0, 0, 14, 16, 30, 17, 30, 0 ; a "зеркальный вид".
        retw     15, 17, 17, 15, 1, 1, 1, 0 ; P "зеркальный вид".

```

В этом листинге собраны все классические подпрограммы управления ЖКИ. Единственное, что надо предусмотреть в случае использования другой марки индикатора, – изменение кодов посылаемых команд.

Эта программа была взята из технического описания устройства фирмы Parallax, она написана с использованием псевдоинструкций микроконтроллера 8051 ассемблера PASM. При желании вы можете преобразовать данный исходный текст в программу на классическом ассемблере PIC-микроконтроллеров, воспользовавшись табл. 2.3 и пояснениями к ней.

ПАРАЛЛЕЛЬНЫЕ ВХОДЫ

Параллельные входы обычно используются для контроля состояния различных коммутационных элементов: выключателей, кнопок и т.п., или чтобы проверять наличие напряжения на некоторых устройствах, например оптронных контактных датчиках. Часто входы микроконтроллеров применяются для опроса клавиатуры пульта управления.

Кнопки и переключатели

Считывать состояние кнопок, выключателей или кодирующих дисков (которые представляют собой особые виды выключателей)

довольно просто. Достаточно подсоединить их между входом и нулевой шиной, как показано на рис. 3.25. Вход «притянут» к высокому логическому уровню ограничительным резистором, величина сопротивления которого может достигать до 100 кОм, чем обеспечивается малое потребление тока.

В том случае, если ваше устройство работает вблизи мощного источника помех (например, двигателя), желательно использовать резистор с небольшим сопротивлением (обычно 4,7 или 10 кОм). Наводки на высокоомный вход будут значительно большими, чем на низкоомный. Однако включать в схему резисторы с еще меньшими значениями сопротивления целесообразно только в особых случаях.

Когда контакты выключателя разомкнуты, на входе будет высокий логический уровень, при замыкании контактов – низкий.

Все механические выключатели имеют одно негативное свойство, известное как «дребезг контактов», которое обусловлено колебаниями упругих контактов при их замыкании и размыкании. Длительность колебаний составляет всего несколько миллисекунд. При этом вместо «чистого» прямоугольного импульса (рис. 3.26а) получается искаженный импульс или пачка импульсов (рис. 3.26б).

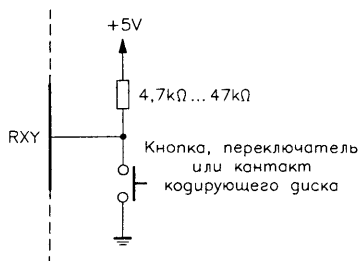


Рис. 3.25

Подключение кнопок и выключателей к микроконтроллеру

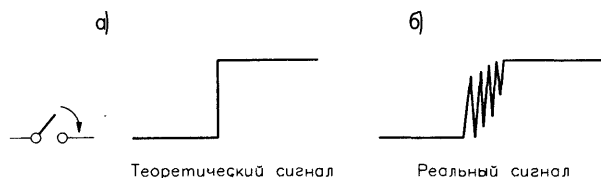


Рис. 3.26

Явление «дребезга контактов»

Обычно такой недостаток устраняют с помощью RS-триггеров, одновибраторов или интегрирующих R-C цепочек, устанавливаемых перед триггерами Шмитта. В устройствах на базе микроконтроллеров борьбу с «дребезгом контактов» возлагают на программу, которая осуществляет многократное считывание состояния входа, подключенного к переключателю, определяя момент устойчивого изменения его состояния. Опрос входа может производиться периодически либо

нерегулярно, по мере того как микроконтроллер освобождается от выполнения текущих задач. В любом случае необходима временная задержка между двумя последовательными считываниями состояния входа.

Гальваническая развязка входов

Когда микроконтроллер должен получать информацию от устройств, находящихся под высоким напряжением или связанных с электрической сетью, самое лучшее решение состоит в том, чтобы обеспечить *гальваническую развязку* входа, например, посредством оптрона. Данный принцип иллюстрируется схемой, представленной на рис. 3.27.

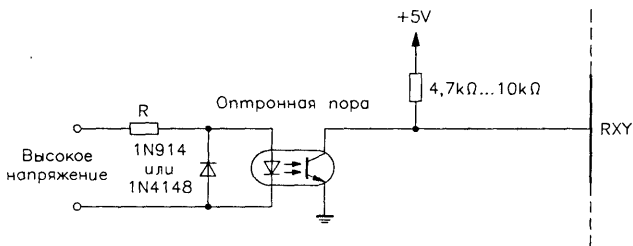


Рис. 3.27

Использование
оптрона
для гальванической
развязки входов

Когда на внешнюю часть схемы подается напряжение, через светодиод оптрона проходит ток и фототранзистор оптрона открывается, переводя вход микроконтроллера на низкий логический уровень. При отключении внешней схемы фототранзистор закрывается, и на входе микроконтроллера будет высокий логический уровень.

Для нормальной работы схемы необходимо, чтобы протекающий через светодиод ток не превышал предельно допустимый, но был достаточным для перевода фототранзистора в режим насыщения, который гарантирует получение на входе микроконтроллера низкого логического уровня. Обеспечить выполнение последнего требования можно, выбрав соответствующий оптрон или использовав усилитель.

Процесс считывания сигнала такой же, как в описанной ранее схеме с переключателем. Борьба с «дребезгом контактов» в данном случае не актуальна.

Клавиатуры

Во многих устройствах для ввода информации используется клавиатура, даже если у нее всего несколько клавиш. Так, телефонный номеронабиратель нуждается по крайней мере в двенадцати клавишах, для домашнего программируемого таймера требуется больше клавиш, чтобы обеспечить удобный ввод времени и даты. Можно привести и другие примеры.

Если клавиатура состоит из нескольких клавиш, они могут быть подключены к микроконтроллеру как отдельные кнопки, то есть каждая через свой порт. Если клавиатура большая, необходимо искать другое решение, поскольку портов у микроконтроллера не слишком много. Здесь возможны два варианта.

Первый состоит в том, чтобы применить внешний клавиатурный кодер, который подключается к N клавишам, а на выходе формирует M -разрядный код, причем $N = 2^M$. Таким образом, шестнадцать клавиш могут быть закодированы четырьмя битами.

На рис. 3.28 показан пример использования подобной микросхемы, а именно ИС типа 74C922 фирмы National Semiconductor. К ее входу подключается клавиатура из шестнадцати клавиш. Данная ИС осуществляет постоянное сканирование клавиш, устраняет влияние «дребезга контактов» и кодирует состояние клавиатуры. Выходной код снимается с выходов D, C, B и A. Нажатие одной из клавиш и наличие кодированных данных на выходе индицируется высоким уровнем сигнала на выходе DA.

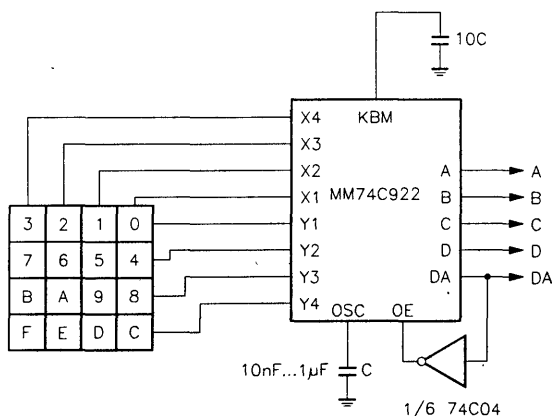


Рис. 3.28

Использование внешнего
клавиатурного кодера

Принцип взаимодействия кодера с микроконтроллером очень прост. Функция микроконтроллера сводится в данном случае к проверке состояния выхода DA и, при обнаружении уровня логической единицы на нем, считыванию кода данных с выходов D, C, B и A.

Алгоритм опроса линии DA может быть произвольным, в том числе может быть увязан с алгоритмом выполнения микроконтроллером системных функций. В этом случае управляющая программа, проходя большой цикл, периодически проводит тестирование линии DA.

Второй вариант – работа в режиме прерывания. В этом случае выход DA соединяется с линией порта B микроконтроллера PIC 16C64 или 16C74. При нажатии на клавишу состояние линии DA изменяется, что вызывает прерывание текущей программы и переход к соответствующей подпрограмме обработки клавиатурного прерывания. Данное решение гарантирует, что любое нажатие на клавишу не будет пропущено, как это иногда происходит при программном опросе линии DA.

Микроконтроллер может с успехом выполнять и функции клавиатурного кодера. Использование микроконтроллеров в матричных клавиатурах – самое универсальное решение. На рис. 3.29 представлен вариант подключения к микроконтроллеру матричной клавиатуры с шестнадцатью клавишами, причем их число может быть без труда увеличено.

Клавиши находятся на пересечении строк и столбцов матрицы. При нажатии на клавишу происходит замыкание соответствующей строки и столбца. Программа по номерам строки и столбца может определить, какая клавиша была нажата. Программа работает следующим образом. Линии столбцов соединены с портами RB0 – RB3 микроконтроллера, являющимися выходами, линии строк, напротив, подключены к входным портам RB4 – RB7. Программа (листинг 3.6) осуществляет сканирование клавиатуры по строкам и столбцам, определяя момент появления логической единицы на одном из входных портов. В процессе сканирования непрерывно инкрементируется переменная key. При обнаружении логической единицы значение переменной key определяет номер нажатой клавиши.

Сканирование столбцов осуществляется путем изменения позиции единицы в четырехразрядном позиционном коде, то есть путем последовательной выдачи на линии столбцов кодов 0001, 0010, 0100 и 1000. При каждом коде производится сканирование строк. Если

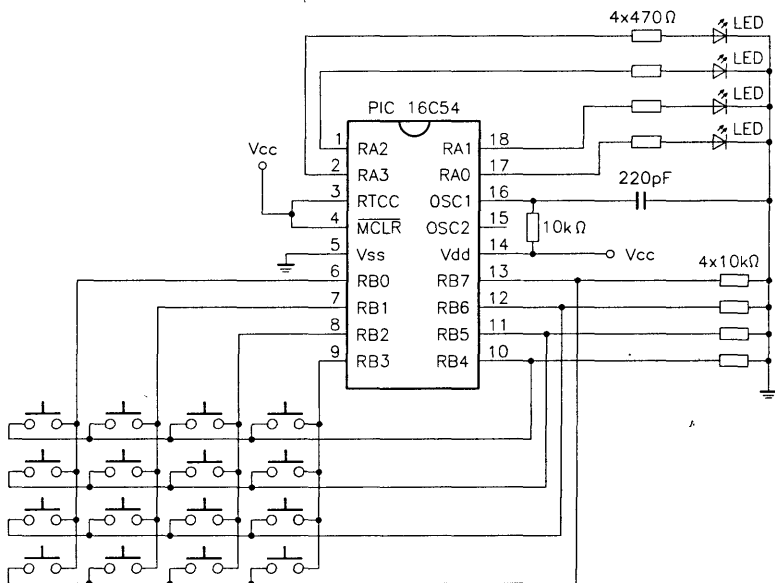


Рис. 3.29

Подключение матричной клавиатуры к микроконтроллеру

какая-либо клавиша была нажата, то в одной из строк будет обнаружена единица. Сканирование в этот момент завершается, а значение переменной `key` идентифицирует нажатую клавишу.

До начала сканирования значение переменной `key` устанавливается равным нулю. Если ни одна клавиша не была нажата, программа возвращает в качестве значения переменной `key` число 16 (10h). Программа осуществляет индикацию номера клавиши в двоичном коде с помощью четырех светодиодов, подсоединенных к порту A.

Листинг 3.6

- ; Интерфейс матричной клавиатуры 4x4.
- ; Программа соответствует инструкции по применению фирмы Parallax.
- ; Эта программа - пример управления матричной клавиатурой 4x4.
- ; Подпрограмма определяет номер нажатой
- ; клавиши в шестнадцатеричном коде от 0 до F (переменная `key`).
- ; Если никакая клавиша не нажата, значение `key` равно 10h.

```
keypad  =  rb
row1    =  rb.4
row2    =  rb.5
```

```
row3      =      rb.6
row4      =      rb.7
```

; Определение переменных величин.

```

      org      8
cols      ds      1
key       ds      1
index     ds      1

```

; Определение используемого PIC-микроконтроллера.

```
device pic16c54,rc_osc,wdt_off, protect_off
reset      start
```

; Исходная программа.

```

start    org 0
mov      !rb, #11110000b ; RB7 - RB4 - входы, RB3 - RB0 - выходы.
mov      !ra, #0          ; Все линии порта A - выходы.
:key     call scankeys
cje      key, #16, :delay
mov      _ra, key
:delay   nop
         nop
         djnz index, :delay
         goto :keys

```

; Подпрограмма сканирования клавиатуры.

```
scankeys clr    key
            clr  keypad
            mov  cols,#4           ; Клавиатура 4x4.
            setb c                  ; Установка бита переноса.
:scan        rl   keypad
            clrb c                  ; Обнуляем бит переноса.
            jb   row1, press
            inc  key
            jb   row2,press
            inc  key
            jb   row3,press
            inc  key
            jb   row4, press
            inc  key
            djnz cols,:scan        ; Проверка окончания цикла.
press        ret                   ; Возвращение с номером в key.
```

Вывод из sleep-режима с помощью клавиатуры

В некоторых устройствах микроконтроллер активизируется только в ответ на воздействие пользователя на клавиатуру, а все остальное время остается в «спящем» режиме (sleep) для экономии энергии. Такой возможностью обладают все PIC-микроконтроллеры подсемейства 16CXX. Выход из sleep-режима может осуществляться по любому внешнему прерыванию, в том числе по прерыванию, вызванному изменением состояния линий порта В.

Чтобы это опробовать, достаточно реализовать схему, приведенную на рис. 3.30. Соответствующая программа должна до перевода микроконтроллера в sleep-режим разрешить прерывания по изменению состояния линий порта В. Нажатие любой клавиши вызовет прерывание, которое выведет микроконтроллер из sleep-режима.

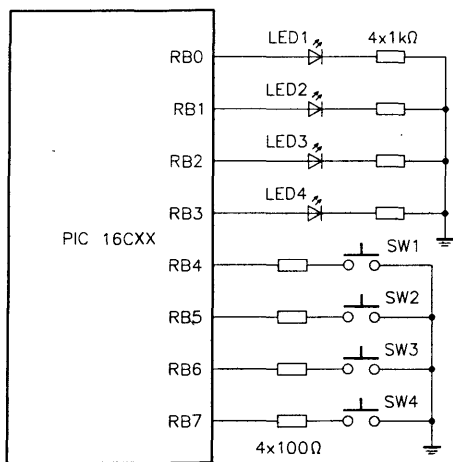


Рис. 3.30

Пример «пробуждения» PIC 16CXX
в результате нажатия на клавишу

Все это реализует программа, приведенная в листинге 3.7. Индикация активизированной клавиши осуществляется светодиодами.

Листинг 3.7

- ; Эта программа иллюстрирует принцип "пробуждения" PIC-микроконтроллера
- ; в результате воздействия на подключенную к нему клавиатуру.
- ; Выход из sleep-режима осуществляется за счет прерывания,
- ; возникающего при изменении состояния линий параллельного порта.


```

; В этом примере был использован микроконтроллер PIC 16C71.
;
; Программа соответствует инструкции по применению AN552 фирмы Microchip.
;
List P=16C71,F=INHX8M
;
Z      equ    2
RBPU   equ    7
temp   equ    10h
OptionReg equ  1h
include "picreg.equ"
;
org    0
;
goto   start
;
org    4
goto   ServiceInterrupt
;
start
call    InitPortB      ; Инициализация порта В.
loop
sleep   ; Вход в sleep-режим.
goto loop
;
ServiceInterrupt

btfsc   INTCON,RBIF    ; Проверка флага RBIF, который индицирует
                        ; изменения состояния порта В.
goto    ServiceWakup   ; Да, тогда обработка.
bcf     INTCON, RTIE    ; Запрет прерываний RTCC.
bcf     INTCON, RTIF    ; Сброс флага RTCC.
return
;
; Эта подпрограмма определяет, какая клавиша нажата,
; и зажигает соответствующий светодиод.
; Затем ожидает, когда все клавиши будут отпущены.
;
ServiceWakup
bcf     INTCON,RBIE     ; Запрет прерываний от порта В.
comf    PORT_B,w        ; Считывание порта В с инверсией.
bcf     INTCON,RBIF     ; Сброс флага прерываний от порта В.
call    delay16         ; Задержка на 16 мс.
comf    PORT_B,w        ; Чтение порта В с инверсией.
andlw   B'11110000'     ; Маскировка выходов.
movwf   temp
swapf   temp,w          ; Перемена мест полубайтов.

```

```

movwf PORT_B      ; Включение индикации.
call  KeyRelease   ; Проверка отпускания клавиши.
retfie

```

```

; Эта подпрограмма ожидает, чтобы все клавиши были отпущены.
;

```

```

KeyRelease

```

```

call  delay16      ; Задержка 16 мс.
comf  PORT_B,w     ; Считывание порта В с инверсией.
bcf   INTCON,RBIF   ; Сброс флага RBIF.
bsf   INTCON,RBIE   ; Разрешение прерываний от порта В.
andlw B'11110000'  ; Маскирование выходов.
btfsc STATUS,z     ; Клавиша остается нажатой?
return ; Нет, тогда возврат.
sleep ; Если да, то вход в sleep-режим.
bcf   INTCON,RBIE   ; Запрет прерываний от порта В.
comf  PORT_B,w     ; Считывание порта В с инверсией.
bcf   INTCON,RBIF   ; Сброс флага RBIF.
goto  KeyRelease    ; Повтор проверки.

```

```

; Эта подпрограмма инициализирует порт В.
;

```

```

InitPortB

```

```

bsf   STATUS,RPO    ; Банк 1.
movlw B'00000011'   ; Порт А определяем как
                    ; цифровой порт ввода/вывода.
movwf ADCON1        ; /
movlw 0
movwf PORT_A        ; Обнуление порта А.
movlw B'11110000'   ; RB0 - RB3 - выходы.
movwf PORT_B        ; RB4 - RB7 - входы.
bcf   OptionReg,RBPU ; Разрешить подключения "подтягивающих"
                    ; резисторов к линиям порта В.
bcf   STATUS,RPO    ; Банк 0.
clrf  PORT_B        ; Обнуление порта В.
clrf  PORT_A        ; Обнуление порта А.
bsf   PORT_A,0      ; Установка первого бита порта А в 1.
bcf   INTCON,RBIE   ; Запрет прерываний от порта В.
movf  PORT_B,w      ; Считывание порта В.
bcf   INTCON,RBIF   ; Сброс флага RBIF.
bsf   INTCON,RBIE   ; Разрешить прерывание от порта В.

retfie ; Возврат.

```

```

; Подпрограмма задержки на 16 мс, рассчитанная на работу
; микроконтроллера с кварцевым резонатором частотой 4,096 МГц.

```

```

;
delay16

```

```

    bsf    STATUS, RPO      ; Банк 1.
    movlw  B'00000111'     ; fosc/256 -> RTCC.
    movwf  OptionReg       ; /
    bcf    STATUS, RPO      ; Банк 0.
    clrf   RTCC
    bcf    INTCON, RTIF     ; Сброс флага прерываний таймера.
    bsf    INTCON, RTIE     ; Разрешить прерывания таймера.

```

```

CheckAgain

```

```

    btfss  INTCON, RTIF     ; Переполнение таймера?
    goto   CheckAgain      ; Нет, ожидание.
    bcf    INTCON, RTIE     ; Запрет прерываний таймера.
    bcf    INTCON, RTIF     ; Сброс флага прерываний таймера.
    return                          ; Возврат.

```

```

;
end

```

Еще один пример использования матричной клавиатуры показан на рис. 3.31. В данном случае сокращение количества необходимых внешних компонентов получено за счет использования внутренних «подтягивающих» резисторов порта В на линиях RB4 – RB7. Последовательно включенные резисторы номиналом 100 Ом обеспечивают защиту микроконтроллера PIC 16CXX от электростатических разрядов.

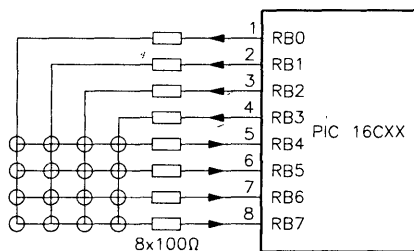


Рис. 3.31

Усовершенствованный вариант схемы
матричной клавиатуры

Данная схема непригодна для PIC-микроконтроллеров подсемейства 16C5X, так как они не имеют входов внешних прерываний. Однако свое решение есть и для них (рис. 3.32).

Использованный в схеме микроконтроллер (16C54) установлен в sleep-режим и потребляет, таким образом, минимум энергии. При нажатии одной из двух клавиш он «пробуждается» и зажигает соответствующий светодиод, затем снова возвращается в sleep-режим.

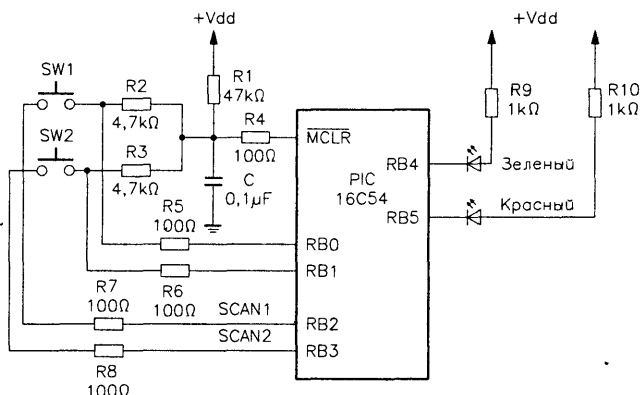


Рис. 3.32

Схема вывода PIC 16C5X из sleep-режима в результате нажатия клавиши

Этот принцип может быть реализован в любом приложении для клавиатуры с любым числом клавиш.

Когда микроконтроллер находится в sleep-режиме, на выходах RB2 и RB3 низкий логический уровень. Конденсатор С заряжен, поэтому на входе mclr будет высокий логический уровень. При нажатии на одну из двух клавиш конденсатор разряжается через R2 или R3, что приводит к сбросу микроконтроллера и переводу линий параллельных портов в высокоимпедансное состояние. Разряд конденсатора С прекращается и микроконтроллер может выйти из состояния сброса и начать опрос клавиш.

Кратковременный перевод линий RB2 и RB3 в низкий уровень не оказывает воздействия на вывод MCLR, так как за это время (10 мкс) конденсатор С не успеет разрядиться через R2 и R3. Как только нажатая клавиша дешифрована и соответствующая подпрограмма завершена (включен светодиод), команда sleep переводит микросхему в режим низкого потребления.

Резисторы R5 – R8 предназначены для защиты микроконтроллера от электростатических разрядов, возникающих при манипулировании клавиатурой.

Единственное условие, которое надо соблюсти, чтобы эта схема нормально работала, состоит в том, что время заряда и разряда конденсатора С должно быть меньше продолжительности цикла сброса 16C54, обычно равной 18 мс.

Данный алгоритм реализует программа, представленная листингом 3.8. Пример расширения этой схемы до классической матричной клавиатуры на шестнадцать клавиш представлен на рис. 3.33.

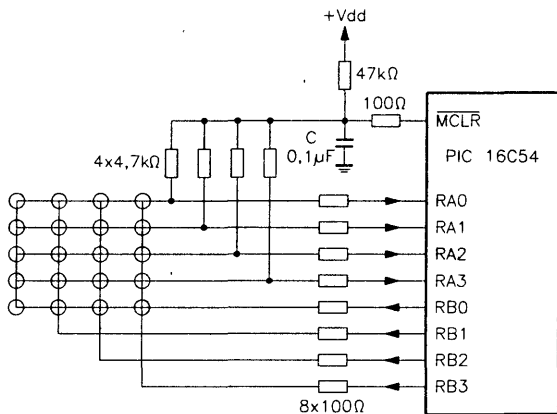


Рис. 3.33

Усовершенствованная схема вывода PIC 16C5X из sleep-режима

Листинг 3.8

Вывод микроконтроллера из sleep-режима с помощью клавиатуры.

LIST P = 16C54

```
*****
;
; Эта программа иллюстрирует принцип вывода PIC 16C5X из sleep-режима
; путем воздействия на клавиши клавиатуры.
; Программа написана для двух клавиш, но может быть расширена.
; Когда нажата клавиша SW1, загорается зеленый светодиод.
; Когда нажата клавиша SW2, загорается красный светодиод.
; Программа соответствует инструкции по применению AN528 фирмы Microchip.
;
*****
;
; Определение констант.
```

```
PC          EQU      2
PORT_B      EQU      6
SCAN1       EQU      2
SCAN2       EQU      3
SW1         EQU      0
SW2         EQU      1
GRN_LED     EQU      4
RED_LED     EQU      5
MSEC_20     EQU      D'20'
```

DB1	EQU	8
GP	EQU	8
DB2	EQU	9

; Назначение линий порта В:

```

;
; 0 -> SW1      ВХОД
; 1 -> SW2      ВХОД
; 2 -> SCAN1    ВЫХОД
; 3 -> SCAN2    ВЫХОД
; 4 -> GRN_LED  ВЫХОД
; 5 -> RED_LED  ВЫХОД
; 6 и 7 -> Неиспользованные ВЫХОДЫ;
;

```

```

ORG      0
;

```

START

```

CALL      INIT_PORT_B      ; Инициализация порта В.
CALL      DELAY             ; Задержка 20 мс.
CALL      SCAN_KEYS        ; Считывание клавиш.
MOVWF     GP                ; Запись в переменную GP.
BTFSK     GP, SW1           ; Продолжение, если не нажата
; клавиша SW1,
CALL      TURN_GREEN_ON    ; или включение зеленого светодиода.
BTFSK     GP, SW2           ; Продолжение, если не нажата
; клавиша SW2,
CALL      TURN_RED_ON      ; или включение красного светодиода.
CHK_FOR_KEY
CALL      DELAY             ; Задержка 20 мс.
CALL      SCAN_KEYS        ; Считывание клавиш
XORLW     0                 ;
BNZ       CHK_FOR_KEY      ; Если клавиша еще нажата,
; повторяем цикл.
NO_KEY_PRESSED
BCF       PORT_B, SCAN1     ; Сканируем линии порта.
BCF       PORT_B, SCAN2     ; /
SLEEP
; Устанавливаем sleep-режим.
;
INIT_PORT_B
MOVLW     B'00000011'      ; RB0 и RB1 - входы.
TRIS      PORT_B           ; RB2 - RB7 - выходы.
MOVLW     OFFh             ;
MOVWF     PORT_B           ;
RETLN      0               ;
;

```

```

; Эта подпрограмма сканирует две клавиши и возвращает:
;
;   • 0, если никакая клавиша не нажата;
;   • 1, если нажата клавиша SW1;
;   • 2, если нажата клавиша SW2;
;   • 3, если нажаты клавиши SW1 и SW2;
;
SCAN_KEYS
    BCF     PORT_B,SCAN1    ; Обнуляем линию SCAN1.
    BCF     PORT_B,SCAN2    ; Обнуляем линию SCAN2.
    MOVLW   B'00000011'     ; Загружаем маску
    ANDWF   PORT_B,0        ; и опрашиваем порт B.
    BSF     PORT_B,SCAN1     ; Устанавливаем 1 на линиях
    BSF     PORT_B,SCAN2     ; SCAN1 и SCAN2.
    ADDWF   PC,1            ; Считываем таблицу.
    RETLW   3               ; Нажаты клавиши SW1 и SW2.
    RETLW   2               ; Нажата клавиша SW2.
    RETLW   1               ; Нажата клавиша SW1.
    RETLW   0               ; Клавиши не нажаты.
;
; Формируем интервал 20 мс (Fosc = 2 МГц).
;
DELAY
    MOVLW   MSEC_20
    MOVWF   DB1
DLY1
    CLRF    DB2
    DECFSZ  DB1
    GOTO    DLY2
    RETLW   0
DLY2
    DECFSZ  DB2
    GOTO    DLY2
    GOTO    DLY1
;
; Включение зеленого светодиода.
TURN_GREEN_ON
    BCF     PORT_B,GRN_LED
    RETLW   0
;
; Включение красного светодиода.
TURN_RED_ON
    BCF     PORT_B,RED_LED
    RETLW   0
;
END

```

КОМБИНИРОВАННОЕ ИСПОЛЬЗОВАНИЕ ПОРТОВ

Даже при использовании высокотехнологичных компонентов, таких как матричная клавиатура и микросхемы, управляющие цифровыми индикаторами (как рассмотренная ранее ИС МС14499), случается, что для управления устройством микроконтроллеру может не хватить линий портов.

Однако за то время, пока нажимается клавиша, микроконтроллер успевает выполнить множество операций, в том числе связанных с портами ввода/вывода. Благодаря этому можно использовать одни и те же порты для разных операций, что позволит вам отказаться от применения дорогостоящих микроконтроллеров. Особенно наглядна такая возможность при взаимодействии с индикаторами и клавиатурами. Обычно одни и те же порты используются как выходы для управления индикаторами, и как входы при опросе клавиатуры. Порт в этом случае – двунаправленный.

Пример схемы, где реализуется описанный принцип, представлен на рис. 3.34. Она обеспечивает управление четырьмя семисегментными индикаторами и взаимодействие с матричной клавиатурой, имеющей шестнадцать клавиш. Для этого задействуются только двенадцать линий портов.

Резисторы с R1 по R8 предназначены для ограничения тока индикаторов. Время цикла работы индикатора составляет 20 мс, что гарантирует визуальную стабильность изображения. Каждая цифра высвечивается в течение 5 мс. Семисегментный код отображаемой цифры выдается на линии RB0 – RB7. Управление общими электродами индикаторов осуществляется через четыре линии порта A. Номиналы резисторов R9 – R12 значительно выше, чем значения сопротивлений R1 – R8. Это необходимо, чтобы нажатие на клавишу не влияло на свечение индикаторов.

При опросе клавиатуры микроконтроллер работает следующим образом. Порты RB0 – RB3 являются выходами, а порты RB4 – RB7 назначены как входы. Через внутренние резисторы входные линии RB4 – RB7 подключены к потенциалу питания, что эквивалентно подаче логических единиц. Через порты RB0 – RB3 осуществляется сканирование (перебор) столбцов клавиатуры низкими логическими уровнями. Каждый раз после переключения столбца считываются линии RB4 – RB7. Низкий уровень на этих линиях может появиться, только если нажата клавиша. Зная номер текущего активного столбца и определив номер строки, в которой обнаружен нулевой уровень, можно

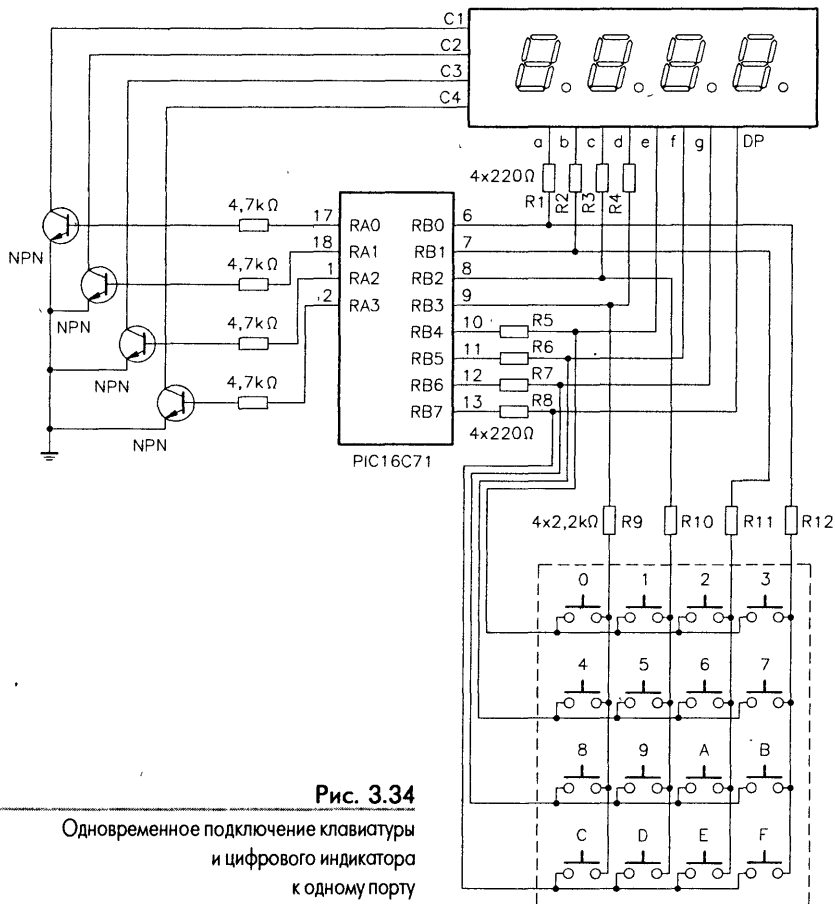


Рис. 3.34

Одновременное подключение клавиатуры
и цифрового индикатора
к одному порту

определить номер нажатой клавиши. Описанный алгоритм реализует программа, приведенная в листинге 3.9. При включении устройства на его индикаторе отображаются четыре нуля. Затем в правой позиции индикатора будет отображаться номер последней нажатой клавиши в шестнадцатеричном представлении. При каждом новом нажатии клавиши индикация будет сдвигаться влево на один разряд.

Листинг 3.9

```

;*****
; Эта программа показывает, как посредством одних и тех же портов
; можно управлять 4-разрядным семисегментным светодиодным
; индикатором и матричной клавиатурой на 16 клавиш.

```

; Главная программа сначала отображает 0000, а затем номер очередной нажатой клавиши
 ; в шестнадцатеричном коде в правой позиции индикатора, с автоматическим смещением
 ; номеров предыдущих клавиш в левую сторону индикатора.
 ; Индикация обновляется каждые 20 мс, клавиатура сканируется в том же темпе.
 ; Таймер использован для генерации прерываний через каждые 5 мс.

; Программа соответствует инструкции по применению AN557 фирмы Microchip.

LIST P=16C71, F=INHX8M

include "picreg-equ"

TempC	EQU	0x0c	; Временные регистры общего применения.
TempD	EQU	0x0d	
TempE	EQU	0x0e	
PABuf	EQU	0x20	
PBBuf	EQU	0x21	
Count	EQU	0x0f	; Счетчик.
MsdTime	EQU	0x10	; Старший байт.
LsdTime	EQU	0x11	; Младший байт.
KeyFlag	EQU	0x12	; Флаг клавиатуры.
keyhit	EQU	0	; Бит 0 - значит, клавиша нажата.
DebnceOn	EQU	1	
noentry	EQU	2	; Нет клавиши = 0.
ServKey	EQU	3	; Бит 3 - значит, обработка клавиши.
Debnce	EQU	0x13	
NewKey	EQU	0x14	
WBuffer	EQU	0x2f	
StatBuffer	EQU	0x2e	
OptionReg	EQU	1	
PCL	EQU	2	

; Макрооперация сохранения байта состояния и содержимого рабочего регистра в буфере.

```
push    macro
        movwf  WBuffer
        swapf  WBuffer
        swapf  STATUS, w
        movwf  StatBuffer
    endm
```

; Макрооперация считывания байта состояния и содержимого рабочего регистра из буфера.

```
pop     macro
        swapf  StatBuffer, w
        movwf  STATUS
        swapf  WBuffer, w
    endm
```

```

org      0
goto     Start      ; Старт.
org      4

;
; Сохраняем рабочий регистр и регистр состояния в момент прерывания.
;
Push
    call   ServiceInterrupts
    pop
    retfie

;
Start
    call   InitPorts
    call   InitTimers

loop
    btfsc  KeyFlag, ServKey ; Проверка флага нажатия.
    call   ServiceKey      ; Да, тогда обработка.
    goto   loop

;
; Подпрограмма обработки нажатия клавиши. Msd - старшая цифра, Lsd - младшая цифра.
;
ServiceKey
    movf   NewKey, w      ; Считывание регистра NewKey.
    movwf  TempE          ; Сохранение в TempE.
    swapf  MsdTime, w     ; Считывание Msd и перестановка полубайтов.
    andlw  B'11110000'    ; Обнуление младшего полубайта.
    movwf  MsdTime        ; Сохранение.
    swapf  LsdTime, w     ; Считывание Lsd и перестановка полубайтов.
    andlw  B'00001111'    ; Обнуление старшего полубайта.
    iorwf  MsdTime        ; Логическое сложение с Msd.
    swapf  LsdTime, w     ; Считывание Lsd и перестановка полубайтов.
    andlw  B'11110000'    ; Обнуление младшего полубайта.
    iorwf  TempE, w       ; Логическое сложение с TempE.
    movwf  LsdTime        ; Сохранение.
    bcf    KeyFlag, ServKey ; Сброс флага нажатия.
    return                ; Возврат.

;
InitPorts
    bsf    STATUS, RPO    ; Банк 1.
    movlw  3              ; RA0 - RA3 - цифровые порты.
    movwf  ADCON1         ;
    clrf   TRISA          ; RA0 - RA4 - выходы.
    clrf   TRISB          ; RB0 - RB7 - выходы.
    bcf    STATUS, RPO    ; Банк 0.
    clrf   PORT_A         ; Обнуление порта A.

```

```

    clrf    PORT_B          ; Обнуление порта В.
    bsf     PORT_A, 3       ; Установка бита 3 порта А.
    return          ; Возврат.

```

```

; Частота тактового генератора - 4,096 МГц, частота командных циклов - 1,024 МГц
; что с предварительным делителем на 32 осуществляет инкрементацию RTCC
; каждые 31,25 мсек.
; Модуль счета таймера - 96, поэтому прерывание будет происходить каждые 5 мс.

```

InitTimers

```

    clrf    MsdTime        ; Сброс MsdTime
    clrf    LsdTime        ; и LsdTime.
    clrf    KeyFlag        ; Сброс флагов.
    bsf     STATUS, RP0    ; Банк 1.
    movlw   B'10000100'    ; Предварительное деление на 32.
    movwf   OptionReg      ;
    bcf     STATUS, RP0    ; Банк 0.
    movlw   B'00100000'    ; Разрешение прерывания таймера.
    movwf   INTCON         ;
    movlw   .96            ; Предзагрузка таймера.
    movwf   RTCC           ; Начало счета.
    retfie                ; Выход из прерывания.

```

ServiceInterrupts

```

    btfsc   INTCON, RTIF   ; Прерывание таймера?
    goto    ServiceRTCC   ; Да, обработка.
    clrf    INTCON        ; Нет, сброс INTCON.
    bsf     INTCON, RTIE   ; Разрешение прерывания от таймера.
    return

```

ServiceRTCC

```

    movlw   .96            ; Инициализация таймера.
    movwf   RTCC          ;
    bcf     INTCON, RTIF   ; Сброс флага прерываний таймера.
    btfsc   PORT_A, 0     ; Если бит 0 порта А равен единице,
    call    ScanKeys      ; быстрое сканирование клавиатуры.
    call    UpdateDisplay  ; Обновление индикации.
    return

```

```

; Сканируем клавиатуру 4x4 и выдаем номер клавиши в NewKey,
; если клавиша была нажата. Если нет, обнуляем указатель keyhit.
; Подпрограмма устраняет "дребезг контактов".
; Клавиатура сканируется каждые 20 мс.

```

ScanKeys

```

btfss KeyFlag, Debnce0n ; Задержка окончена?
goto Scan1 ; Нет, тогда сканирование
; клавиатуры.
decfsz Debnce ; Да, тогда уменьшаем счетчик
; задержки.
return ; Возвращаемся, если не 0.
bcf KeyFlag, Debnce0n ; Сброс флага
return ; и возврат.

```

Scan1

```

call SavePorts ; Сохранение портов.
movlw B'11101111' ; Загрузка TempD.
movwf TempD ;

```

ScanNext

```

movf PORT_B, w ;
bcf INTCON, RBIF ; Сброс флага прерывания от
; порта В.
rrf TempD ; Правый сдвиг TempD.
btfss STATUS, C ; Перенос = 1?
goto NoKey ; Нет, тогда окончание
movf TempD, w ; ИЛИ TempD и w
movwf PORT_B ; и выдача в PORT_B.
pop ;
btfss INTCON, RBIF ; Значение флага прерывания от порта В равно 1?
goto ScanNext ; Нет, тогда продолжаем.
btfsc KeyFlag, keyhit ; Последняя клавиша отпущена?
goto SKreturn ; Нет, тогда выход.
bsf KeyFlag, keyhit ; Устанавливаем флаг нажатия
; новой клавиши.
swapf PORT_B, w ; Считывание порта В.
movwf TempE ; Сохранение в TempE.
call GetKeyValue ; Считывание значения клавиши
; от 0 до F.
movwf NewKey ; Сохранение в NewKey.
bsf KeyFlag, ServKey ; Установка флага обработки клавиш.
bsf KeyFlag, Debnce0n ;
movlw 4 ;
movwf Debnce ; Изменение времени задержки.

```

SKreturn

```

call RestorePorts ; Возмещение портов.
;
return ; NoKey
bcf KeyFlag, keyhit ; Сброс флага.
goto SKreturn

```

Соответствие номеров клавиш, строк, столбцов и портов.

	Col1 (RB3)	Col2 (RB2)	Col3 (RB1)	Col4 (RB0)
; Ran1(RB4)	0	1	2	3
; Ran2(RB5)	4	5	6	7
; Ran3(RB6)	8	9	A	B
; Ran4(RB7)	C	D	E	F


```

GetKeyValue
    clrf    TempC          ;
    btfss   TempD,3        ; Первый столбец.
    goto    RowValEnd      ;
    incf    TempC          ;
    btfss   TempD,2        ; Второй столбец.
    goto    RowValEnd      ;
    incf    TempC          ;
    btfss   TempD,1        ; Третий столбец.
    goto    RowValEnd      ;
    incf    TempC          ; Последний столбец.
RowValEnd
    btfss   TempE,0        ; Первая строка?
    goto    GetValCom      ; Да, считывание клавишей 0,1,2,3.
    btfss   TempE,1        ; Вторая строка?
    goto    Get4567        ; Да, считывание 4,5,6,7.
    btfss   TempE,2        ; Третья строка?
    goto    Get89ab        ; Да, считывание 8,9,a,b.
Getcdef
    bsf     TempC,2        ;
Get89ab
    bsf     TempC,3        ;
    goto    GetValCom      ;
Get4567
    bsf     TempC,2        ;
GetValCom
    movf    TempC, w       ; Таблица номеров клавиш.
    addwf   PCL            ;
    retlw   0              ;
    retlw   1              ;
    retlw   2              ;
    retlw   3              ;
    retlw   4              ;
    retlw   5              ;
    retlw   6              ;
    retlw   7              ;

```

```

retlw 8      ;
retlw 9      ;
retlw 0a     ;
retlw 0b     ;
retlw 0c     ;
retlw 0d     ;
retlw 0e     ;
retlw 0f     ;

```

; Сохраняем состояния портов А и В во время сканирования клавиатуры.

SavePorts

```

movf  PORT_A,w
movwf PABuf      ; Сохраняем порт А.
clrf  PORT_A     ; Выключение индикации.
movf  PORT_B,w
movwf PBBuf      ; Сохраняем порт В.
movlw 0xff       ; На всех линиях порта В высокий уровень.
movwf PORT_B
bsf    STATUS,RPO ; Банк 1.
bcf    OptionReg,7 ; Разрешаем подключение
        ; "подтягивающих" резисторов.
movlw B'11110000' ; Старшие четыре разряда порта В - входы.
movwf TRISB       ; Младшие - выходы.
bcf    STATUS,RPO ; Банк 0.
return

```

; Восстанавливаем порты А и В после сканирования клавиатуры.

RestorePorts

```

movf  PBBuf,w      ; Восстановление содержимого
movw  fPORT_B      ; порта В.
movf  PABuf,w      ; Восстановление содержимого
movwf PORT_A       ; порта А.
bsf    STATUS,RPO  ; Банк 1.
bsf    OptionReg,7 ; Запрет "подтягивающих" резисторов.
clrf  TRISA        ; Все линии порта А - выходы.
clrf  TRISB        ; Все линии порта В - выходы.
bcf    STATUS,RPO  ; Банк 0.
return

```

; Обновление индикации.

UpdateDisplay

```

movf  PORT_A,w      ;
clrf  PORT_A        ;

```

```

andlw 0x0f      ;
movwf TempC     ;
bsf    TempC, 4  ;
rrf    TempC     ;
btfss  STATUS, CARRY ;
bcf    TempC, 3  ;
btfsc  TempC, 0  ;
goto   UpdateMsd ;
btfsc  TempC, 1  ;
goto   Update3rdLsd ;
btfsc  TempC, 2  ;
goto   Update2ndLsd ;

```

; Отображение первой цифры.

UpdateLsd

```

movf   LsdTime, w      ; Считывание Lsd в регистр W.
andlw  0x0f            ; Маскирование старшего полубайта.
goto   DisplayOut      ; Переход к индикации.

```

; Отображение второй цифры.

Update2ndLsd

```

swapf  LsdTime, w      ;
andlw  0x0f            ;
goto   DisplayOut      ;

```

; Отображение третьей цифры.

Update3rdLsd

```

movf   MsdTime, w      ;
andlw  0x0f            ;
goto   DisplayOut      ;

```

; Отображение четвертой цифры.

UpdateMsd

```

swapf  MsdTime, w      ; Считывание Msd в регистр W.
andlw  0x0f            ; Индикация.

```

DisplayOut

```

call   LedTable        ; Считывание семисегментного кода цифры.
movwf  PORT_B           ; Управление светодиодами.
movf   TempC, w         ;
movwf  PORT_A           ;
return

```

; Таблица индикации.

LedTable

```

addwf  PCL              ; Модификация программного счетчика (PC).
retlw  B'00111111'      ; Индикация цифры 0.
retlw  B'00000110'      ; Индикация цифры 1.
retlw  B'01011011'      ; Индикация цифры 2.

```



```
retlw B'01001111' ; Индикация цифры 3.  
retlw B'01100110' ; Индикация цифры 4.  
retlw B'01101101' ; Индикация цифры 5.  
retlw B'01111101' ; Индикация цифры 6.  
retlw B'00000111' ; Индикация цифры 7.  
retlw B'01111111' ; Индикация цифры 8.  
retlw B'01100111' ; Индикация цифры 9.  
retlw B'01110111' ; Индикация цифры A.  
retlw B'01111100' ; Индикация цифры B.  
retlw B'00111001' ; Индикация цифры C.  
retlw B'01011110' ; Индикация цифры D.  
retlw B'01111001' ; Индикация цифры E.  
retlw B'01110001' ; Индикация цифры F.
```

end

ВНЕШНЯЯ ПЕРИФЕРИЯ

Хотя микроконтроллеры постоянно совершенствуются и их технические возможности расширяются, они не всегда имеют аппаратные узлы, реализующие необходимую функцию. Причины разные: либо функция еще не реализована в данном семействе микроконтроллеров, либо микросхема конкретного типа, на который вы ориентируетесь, ею не обладает. Это особенно актуально для PIC-микроконтроллеров 16C5X, не обладающих развитыми аппаратными ресурсами для реализации интерфейсных функций.

Существует два способа реализации необходимых функций. Первый путь – программный. Но это не всегда возможно либо в силу особенностей функции, либо ввиду чрезмерной нагрузки на ЦПУ.

Второе решение состоит в том, чтобы использовать периферийную микросхему, выполняющую функции, которых нет у микроконтроллера. Если микросхема рассчитана на работу со стандартным интерфейсом, в частности с шинами адресов и данных, его необходимо реализовать программно, через порты ввода/вывода. Суть программной реализации заключается в том, чтобы генерировать временные диаграммы сигналов интерфейса и алгоритмы обмена сигналами.

Программная эмуляция интерфейса не вызовет затруднений, если интерфейс асинхронный, то есть обмен не тактируется и скорость обмена невелика. Если же обмен по интерфейсу синхронный и время его выполнения жестко ограничено, можно столкнуться с трудностями из-за загруженности микроконтроллера.

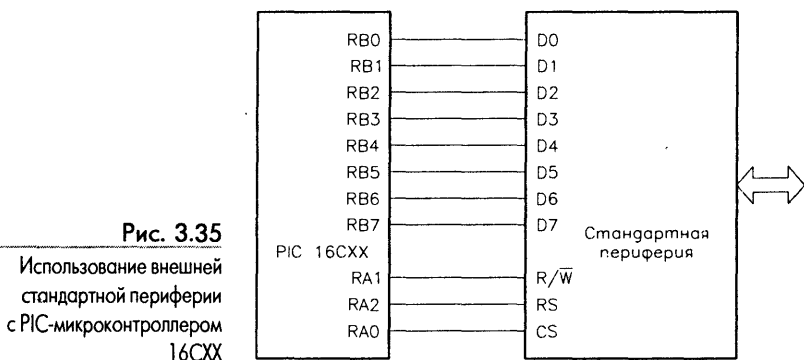
Сначала рассмотрим основные особенности интерфейса параллельного типа.

Стандартная периферия

Под стандартной периферией понимается большинство периферийных интерфейсных схем, предназначенных обычно для восьмибитных микропроцессоров: PIO – фирмы Intel, PIA – фирмы Motorola и т. д.

За редкими исключениями эти микросхемы имеют следующие выводы (рис. 3.35, правая сторона):

- ◆ восемь линий данных D0 – D7;
- ◆ линия считывания / записи R/W;
- ◆ одна или несколько линий выбора (разрешения);
- ◆ одна или несколько линий адресации внутренних регистров (в зависимости от сложности микросхемы).



На рис. 3.35 показана микросхема с линией выбора микросхемы CS и линией выбора регистра RS. Очевидно, что при использовании нескольких микросхем необходимо организовывать несколько линий RS. Линии же CS могут быть объединены для экономии линий параллельных портов.

Как вы могли заметить, схемы на рис. 3.35 и 3.24 похожи. И это не удивительно, поскольку использованный ЖКИ имеет контроллер со стандартным параллельным интерфейсом.

Поэтому программа, представленная листингом 3.10, не вызовет у вас затруднений: она функционирует точно так же, как программа управления ЖКИ, и включает следующие этапы:

- ♦ обнуление линии R/W;
- ♦ установка соответствующего уровня линии RS в зависимости от того, к какому регистру должен обратиться микроконтроллер;
- ♦ выдача данных на линии D0 – D7;
- ♦ установка логической единицы на линию CS, чтобы разрешить прием сообщения;
- ♦ установка логического нуля на линию CS, чтобы защелкнуть данные, принятые по линиям D0 – D7.

Единственное, что нужно принимать в расчет при создании схем с таким интерфейсом, это ограничения по скорости. Действительно, при использовании обычного восьмибитного PIC-микроконтроллера, тактируемого частотой 4 МГц, невозможно обеспечить скорость передачи выше одного мегабайта в секунду. При тактовой частоте PIC-микроконтроллера в 20 МГц превысить этот предел довольно легко. И наоборот, в некоторых случаях для доступа к периферийной микросхеме может потребоваться временная задержка, чтобы замедлить обмен.

Листинг 3.10

```
; Управление периферией по стандартному параллельному интерфейсу.
; Программа предназначена для восьмибитного микроконтроллера
; и обеспечивает считывание и запись.
```

```
RS    =    ra.2        ; Линия выбора регистра.
RW    =    ra.1        ; 0 = запись, 1 = считывание.
CS    =    ra.0        ; 1 = обмен разрешен, 0 = запрещен.
data  =    rb          ; Данные.
```

```
; Определение переменных величин.
```

```
org    8
temp   ds    1          ; Служебный регистр.
org    0
```

```
; Определение используемого PIC.
```

```
device pic16c54,xt_osc,wdt_off,protect_off
reset          start
```

```
start mov    ra,#0      ; Инициализация порта A.
      mov    rb,#0      ; Инициализация порта B.
      mov    !ra,#0h     ; Все линии порта A – выходы.
      mov    !rb,#0h     ; Все линии порта B – выходы.
```

; Запись данных, содержащихся в переменной temp, в периферийную микросхему.

```

;
есrit clrb RS ; Выбор регистра.
      clrb RW ; Если RW = 0, производится операция записи.
      mov data,temp ; Выдача данных.
      setb CS ; Выбор микросхемы (исполнение операции).
      nop ; Задержка в одну операцию.
      clrb CS ; Завершение записи.

```

; Считывание данных из периферийной микросхемы в переменной temp.

```

;
lit mov !rb,#255 ; Порт В - входной.
     setb RS ; RS = 1 (напримеp).
     setb RW ; Если RW = 1, выполняется считывание.
     setb CS ; Выбор микросхемы.
     nop ; Задержка в одну операцию.
     mov temp,data ; Считывание данных.
     clrb CS ; Завершение операции.
end

```

Взаимодействие с периферией по последовательному интерфейсу

Часто для организации параллельного интерфейса у микроконтроллера не хватает линий портов. В таком случае лучше использовать последовательный интерфейс, который встречается, например, в микросхемах памяти, аналого-цифровых преобразователях и т.д.

Некоторые из этих микросхем имеют специфичный интерфейс: используемый алгоритм обмена сигналами и их временные диаграммы свойственны только данной микросхеме. Конечно, во многих случаях микросхемы различных типов и различных фирм оказываются похожими, но их интерфейсы все же не стандартизованы.

Другие микросхемы, напротив, используют последовательные стандартные шины, такие как, например, шина I²C. Изначально эта шина была предложена фирмой Philips, а сейчас микросхемы, поддерживающие ее, выпускаются многими производителями. Она применяется в устройствах, выполняющих самые разные функции: в контроллерах индикации, микросхемах энергонезависимой памяти, часов реального времени, телевизионных микросхемах и т.д. Главным преимуществом данной шины является то, что для организации двухстороннего обмена в ней используются лишь две линии связи. Таким образом, она занимает только две линии портов микроконтроллера (в некоторых случаях три).

Многие модели микроконтроллеров подсемейства PIC 16CXX (например, 16C62, 16C63, 16C64, 16C65, 16C73 и 16C74) помимо I²C поддерживают еще одну стандартную и популярную шину SPI, которая позволяет осуществлять быструю передачу данных. В этой книге данная шина рассматриваться не будет.

Отметим только, что поддерживающая интерфейс I²C программа крайне проста: она должна задать формат (конфигурацию) обмена и определить направление передачи (запись или чтение). Остальное реализуется аппаратно. Во всех других PIC-микроконтроллерах подсемейства 16CXX такой интерфейс может быть реализован только программно. Это будет показано на примере взаимодействия PIC 16CXX с микросхемой энергонезависимой памяти, имеющей последовательный интерфейс (SERIAL EEPROM), и микросхемой аналого-цифрового преобразователя. Кроме того, вы можете обратиться к главе 5, где среди примеров применений PIC 16C54 есть схема, использующая шину I²C.

ЭНЕРГОНЕЗАВИСИМАЯ ПАМЯТЬ С ПОСЛЕДОВАТЕЛЬНЫМ ИНТЕРФЕЙСОМ

Поскольку в книге рассказывается преимущественно о PIC-микроконтроллерах фирмы Microchip, то использование энергонезависимой памяти с последовательным интерфейсом мы рассмотрим на примере ИС SERIAL EEPROM типа 93C56, выпускаемой также фирмой Microchip. Речь идет об электрически программируемой и стираемой памяти. Вы сможете использовать данный пример для всех приложений, где потребуется, чтобы данные сохранялись при выключении питания.

Память этого типа имеет четырехпроводный интерфейс, включающий следующие линии:

- ◆ \overline{CS} – выбор микросхемы;
- ◆ CLK – тактирование обмена;
- ◆ DI – вход данных;
- ◆ DO – выход данных.

Можно свести количество линий к трем (рис. 3.36), соединяя DI и DO через резистор, который не допускает короткого замыкания линий DO и RA0 во время считывания памяти.

Считывание памяти осуществляется со скоростью до 2 Мбит/с. Запись и стирание, напротив, требуют значительно больше времени,

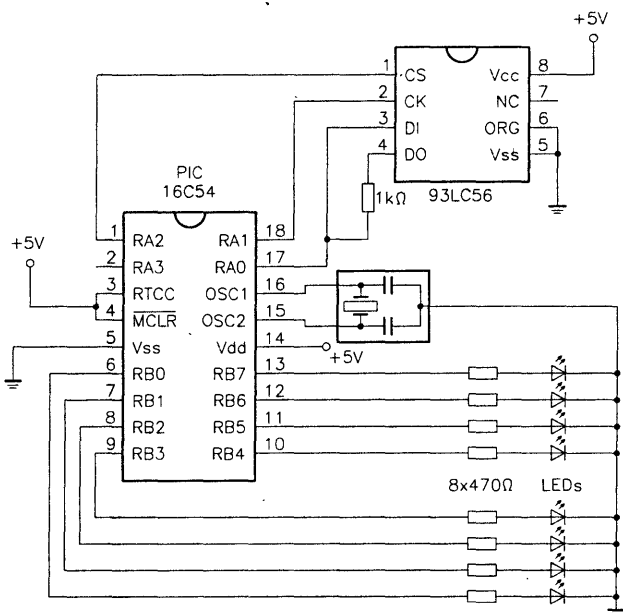


Рис. 3.36

Взаимодействие PIC-микроконтроллера с IIC SERIAL EEPROM

поскольку выполняются внутренним автоматом, реализующим достаточно сложный алгоритм и в том числе формирующим внутри микросхемы достаточно высокие напряжения. Для стирания или записи одного байта требуется 2 мс. В течение этого времени микросхема не может выполнять никаких других операций. Процесс записи и стирания индицируется нулевым уровнем напряжения на линии DO. В листинге 3.11 представлены все базовые подпрограммы, необходимые для диалога с данным устройством (считывание, запись, стирание), и две подпрограммы, реализующие специальные операции: разрешение записи/стирания и запрет записи/стирания.

Чтобы обеспечить безопасность данных, в момент подачи напряжения питания включен режим запрета записи/стирания. Если вы хотите осуществить запись или стирание, то сначала систему нужно вывести из этого режима, что выполняет подпрограмма EEnable. Когда операция записи будет завершена, систему необходимо вернуть в первоначальное состояние, для чего предназначена подпрограмма EDisbl.

При реализации всех операций основной является подпрограмма Shout, которая позволяет выводить в последовательном виде данные, содержащиеся в регистре temp. Длина слов передаваемых данных может составлять 8 или 16 бит и зависит от способа организации памяти. Помимо данных передаются коды команд (3 бита) и адреса (8 бит). Длина текущей посылки определяется переменной clocks, которая указывает необходимое количество тактов передачи.

Эта программа априори подходит для большинства микросхем памяти с аналогичным последовательным интерфейсом, но разные типы микросхем могут иметь различную продолжительность циклов записи и стирания, различную разрядность адресов, различные форматы данных. Поэтому некоторые программные константы придется изменить.

Листинг 3.11

```
; Управление EEPROM с последовательным интерфейсом.
; Соответствует инструкции по применению фирмы Parallax.
; Эта программа содержит подпрограммы, необходимые для управления
; SERIAL EEPROM типа 93C56 или аналогичными.
; Она заполняет все ячейки памяти одинаковыми данными,
; которые индицируются светодиодами, подключенными к порту В.

D   =  ra.0   ; Линии DI и DO EEPROM.
CLK =  ra.1   ; Линия CLK EEPROM.
CS  =  ra.2   ; Линия выбора микросхемы CS
                ; (активный логический уровень - высокий).

; Определение кодов команд памяти. При необходимости
; эту часть надо изменить в соответствии с типом используемой памяти.

ROP = 192    ; Считывание.
WROP = 160   ; Запись.
EWEN = 152   ; Разрешение записи и стирания.
EWDS = 128   ; Запрет записи и стирания.
ERAS = 224   ; Стирание одного байта.
ERAL = 144   ; Стирание всей памяти.
WRAL = 136   ; Заполнение всех ячеек памяти одним и тем же байтом.

; Определение переменных величин.
```

```
org 8
temp ds 1      ; Временная переменная величина.
EEdata ds 1    ; Регистр данных.
EEaddr ds 1    ; Регистр адреса.
clocks ds 1    ; Количество тактов для подпрограммы Shout.
tick1 ds 1     ; Счетчик задержки 1.
tick2 ds 1     ; Счетчик задержки 2.
```

; Определение используемого PIC.

```
devicepic16c54,xt_osc,wdt_off,protect_off
reset start
```

; Код программы, запись в память.

```
org 0
start mov ra,#0 ; Обнуление регистров портов A
mov rb,#0 ; и B.
mov !ra,#0 ; RA - выходной порт.
mov !rb,#0 ; RB - выходной порт.
call EEnable ; Разрешение записи/стирания.
mov EEdata,#1 ; Загрузка регистра данных.
mov EEaddr,#0 ; Загрузка регистра адреса (стартового).
:loop call EEwrite ; Запись данных.
call Busy ; Ожидание исполнения.
rr EEdata ; Сдвиг данных.
ijnz EEaddr,:loop ; Увеличение адреса и повтор.
call EEdisbl ; Запрет записи.

; Чтение из памяти.
mov EEaddr,#0 ; Загрузка регистра адреса (стартовым адресом).
:loop2 call ERead ; Команда считывания.
mov rb, EEdata ;
inc EEaddr ; Увеличение адреса.
call delay ; Задержка.
goto :loop2 ; Повтор.
```

; Эта подпрограмма используется всеми остальными. Она

; осуществляет выдачу сообщений в последовательной форме на вход D EEPROM.

; Перед выводом информация должна быть помещена в temp,

; число сдвигов - в clocks, CS = 1, CS = 0 в конце операции.

```
Shout rl temp ; Сдвиг bit7 temp в бит переноса.
movb D,c ; Посылка бита переноса на линию данных.
setb CLK ; Генерация тактирующего импульса EEPROM.
nop ; Пауза.
clrb CLK ; Сброс линии CLK.
djnz clocks,Shout ; Уменьшение счетчика битов и повтор посылки.
ret ; Возврат в вызывающую программу.
```

; Считывание байта по адресу EEaddr и помещение его в EEdata.

```
ERead mov temp,#ROP ; Код операции считывания помещаем в temp.
mov clocks,#4 ; Число тактов для вывода кода.
setb CS ; Выбор микросхемы.
call Shout ; Вывод кода команды.
```



```

mov    clocks,#8      ; Для вывода адреса задаем 8 тактов.
mov    temp,Eeaddr    ; Загрузка адреса в temp.
call   Shout          ; Пересылка адреса.
mov    !ra,#1         ; RAO = 1.
mov    clocks,#8      ; Для считывания байта задаем 8 тактов.
:read  setb  CLK       ; CLK = 1.
movb   C,D            ; Устанавливаем бит C равным биту данных линии.
rl     temp           ; Бит переноса сдвигаем в temp.0.
clrb   CLK            ; CLK = 0.
djnz   clocks,:read   ; Уменьшаем счетчик тактов, повторяем прием битов.
mov    EEdata,temp    ; Принятый байт перепишем в EEdata.
mov    !ra,#0         ; Порта A - выходной.
clrb   CS             ; Заканчиваем обмен.
ret                               ; Возвращаемся.

```

; Подпрограмма разрешения записи.

```

EEEnable setb  CS      ; Разрешаем доступ к EEPROM.
mov    clocks,#12     ; Формат послыки - 12 бит.
mov    temp,#EWEN     ; Код операции EWEN помещаем в temp.
call   Shout          ; Пересылка.
clrb   CS             ; Запрет доступа.
ret                               ; Возврат.

```

; Подпрограмма запрета записи в памяти.

```

EEdisbl setb  CS      ; Разрешаем доступ к EEPROM.
mov    clocks,#12     ; Формат послыки - 12 бит.
mov    temp,#EWDs     ; Код операции EWDS помещаем в temp.
call   Shout          ; Пересылка.
clrb   CS             ; Запрет доступа.
ret                               ; Возврат.

```

; Запись байта из EEdata по адресу, содержащемуся в EEaddr.

```

EEwrite mov    temp,#WROP ; Код операции записи в temp.
mov    clocks,#4         ; Формат послыки - 4 бита.
setb   CS               ; Разрешаем доступ к EEPROM.
call   Shout            ; Пересылка.
mov    clocks,#8         ; Формат послыки - 8 бит.
mov    temp,EEaddr       ; Установка адреса.
call   Shout            ; Пересылка.
mov    clocks,#8         ; Формат послыки - 8 бит.
mov    temp,EEdata       ; Подготовка данных.
call   Shout            ; Пересылка.

```

```

clrb CS      ; Запрет доступа.
ret          ; Возврат.

```

; Стирание ячейки по адресу EEaddr. После стирания в ячейке окажется байт FFh.

```

EErase mov  temp,#ERAS      ; Код операции записи в temp.
mov  clocks,#4              ; Формат посылки - 4 бита.
setb CS                     ; Разрешаем доступ к EEPROM.
call Shout                  ; Пересылка.
mov  clocks,#8              ; Формат посылки - 8 бит.
mov  temp,Eeaddr            ; Установка адреса.
call Shout                  ; Пересылка.
ret                          ; Возврат.

```

; Стирание 256 байт EEPROM (всей памяти).

```

EEwipe setb CS              ; Разрешаем доступ к EEPROM.
mov  temp,#ERAL            ; Код операции записи в temp.
mov  clocks,#12             ; Формат посылки - 12 бит.
call Shout                  ; Пересылка.
clrb CS                     ; Запрет доступа.
ret                          ; Возврат.

```

; Запись байта, содержащегося в EEdata, во все ячейки памяти.

```

EEwrrall setb CS            ; Разрешаем доступ к EEPROM.
mov  temp,#WRAL            ; Код операции записи в temp.
mov  clocks,#12             ; Формат посылки - 12 бит.
call Shout                  ; Пересылка.
mov  clocks,#8              ; Формат посылки - 12 бит.
mov  temp,EEdata            ; Данные помещаем в temp.
call Shout                  ; Пересылка.
clrb CS                     ; Запрет доступа.
ret                          ; Возврат.

```

; Ожидание завершения выполнения текущей команды.

```

Busy  nop                  ; Пауза.
      mov  !ra,#1           ; Линия RAO - вход.
      setb CS                ; Разрешаем доступ к EEPROM.
:wait jnb  D,:wait          ; Опрос линии D, ожидание.
      clrb CS                ; Запрет доступа.
      mov  !ra,#0           ; Все линии порта A - выходы.
      ret                    ; Возврат.
Delay djnz tick1,Delay      ; Задержка до тех пор,
      djnz tick2,Delay      ; пока tick1 и tick2 не равны 0.
      ret                    ; Выход.

```

УПРАВЛЕНИЕ АНАЛОГО-ЦИФРОВЫМ ПРЕОБРАЗОВАТЕЛЕМ

Только PIC-микроконтроллеры 16C71, 16C73 и 16C74 уже имеют внутренние аналого-цифровые преобразователи. Что касается микроконтроллеров других типов, то их взаимодействие с внешними АЦП может производиться по последовательному интерфейсу.

В качестве примера рассмотрим популярную микросхему ADC0831 фирмы National Semiconductor (рис. 3.37). Этот восьмиразрядный АЦП выдает нулевой байт, когда напряжение прямого входа $V_{IN}(+)$ равно напряжению инвертирующего входа $V_{IN}(-)$. Когда напряжение между прямым и инвертирующим входами равно V_{REF} , на выходе будет байт, равный 255 (или FF в шестнадцатеричном представлении).

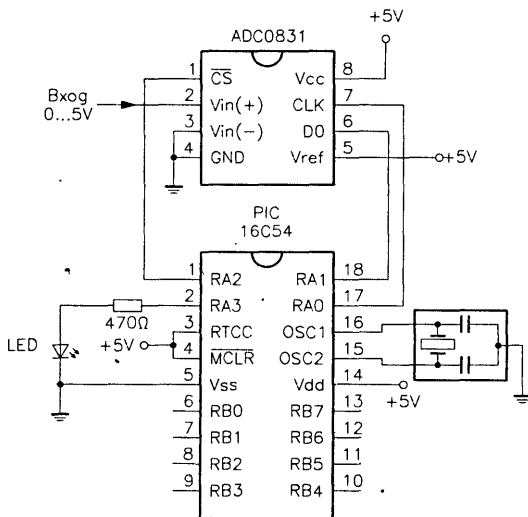


Рис. 3.37

Подключение PIC-микроконтроллера
к аналого-цифровому преобразователю через последовательный интерфейс

В этом примере $V_{IN}(-)$ подсоединен к нулевому потенциалу схемы, а вход V_{REF} – к потенциалу +5 В, но вы можете выбрать свои потенциалы входов, поскольку это не влияет на программное обеспечение, реализующее взаимодействие между PIC-микроконтроллером и преобразователем.

Для взаимодействия используется трехпроводный интерфейс, включающий следующие линии:

- ♦ \overline{CS} – линия выбора микросхемы (разрешения обмена), активный уровень – низкий;
- ♦ CLK – тактирование обмена данными;
- ♦ DO – выход данных преобразователя.

Программа, управляющая работой этой схемы, приведена в листинге 3.12. Индикатором уровня измеренного напряжения является светодиод. Он будет мигать тем медленнее, чем выше напряжение.

Основную часть диалога с преобразователем выполняет подпрограмма convert. Алгоритм взаимодействия следующий. Сначала на линию \overline{CS} выдается низкий логический уровень, разрешающий обмен. Затем микроконтроллер генерирует первый тактовый импульс на линию CLK, чтобы инициировать преобразование. Для считывания результата микроконтроллер генерирует восемь тактовых импульсов на линию CLK и принимает бит за битом байт результата. Прием байта осуществляется в цикле loop, побитная запись во внутренний регистр выполняется через бит переноса посредством команды сдвига влево (rl).

Листинг 3.12

```
; Управление аналого-цифровым преобразователем
; по последовательному интерфейсу Microwire.
; Соответствует инструкции по применению фирмы Parallax.
; Эта программа управляет взаимодействием с внешним АЦП
; с последовательным интерфейсом.
; Индикацию уровня напряжения осуществляет светодиод, который мигает тем медленнее,
; чем больше величина измеренного напряжения.
clk = ra.0 ; Линия синхронизации.
data = ra.1 ; Линия данных.
CS = ra.2 ; Линия выбора микросхемы (разрешения обмена).
LED = ra.3 ; Линия управления светодиодом.

; Определение переменных величин.
org 8
counter1 ds 1 ;
counter2 ds 1 ;
ADresult ds 1 ; Регистр результата.

; Определение используемого PIC.

devicepic16c54,xt_osc,wdt_off,protect_off
reset start
```

```

; Программа.
org      0
start    mov     ra,#00000100b    ; CS = 1, выбор ИС АЦП.
        mov     !ra,#00000010b    ; RA.1 - вход.
        mov     !rb,#00000000b    ; Все линии RB - выходы.
blink    xor     ra,#8             ; Инверсия сигнала линии
                                     ; управления светодиодом RA3.
        call    wait              ; Ожидание окончания преобразования.
        goto    blink             ; Повтор.
wait     call    convert           ; Считывание результата ADC в counter2.
        mov     counter2,Adresult
        add     counter2,#1        ; Добавление единицы.

; Задержка.
:loop    djnz    counter1,:loop
        djnz    counter2,:loop
        ret

convert  clrb    CS                ; Подтверждение ADC.
        mov     counter2,#8        ; Инициализация приема восьми битов данных.
        setb    clk               ; Генерация тактового импульса,
        nop     ; если таймер PIC > 4 МГц.
        clrb    clk               ; Сброс тактового сигнала.
        clr     Adresult           ; Сброс байта для приема новых данных.
:loop    setb    clk               ; Генерация тактового импульса,
        nop     ; если таймер PIC > 4 МГц.
        clrb    clk
        movb    c,data            ; Бит данных в C.
        rl      Adresult          ; Сдвиг влево Adresult с вводом переноса.
        djnz    Counter2,:loop    ; восемь битов получено?
        setb    CS                ; Конец преобразования.
        ret                       ; Выход.

```

ЗАКЛЮЧЕНИЕ

Конечно, в одной главе невозможно рассмотреть все ситуации, с которыми вы можете столкнуться при создании приложений на основе микроконтроллеров. Но, проанализировав предложенные примеры, вы будете обладать прочной базой для самостоятельной работы. Мы также предлагаем посмотреть программы, представленные в главе 5. Они содержат подпрограммы, которые здесь не приведены (например, совместное управление клавиатурой/индикатором/аналого-цифровым преобразователем и др.).

Г Л А В А 4

БИБЛИОТЕКА ПРОГРАММ

В этой главе:

Арифметические подпрограммы

Программная реализация прерываний
микроконтроллеров 16C5X

Принцип многозадачности

Расширение стековой памяти
микроконтроллеров 16C5X

Передача асинхронной последовательности
при отсутствии последовательного порта

В этой главе вашему вниманию предлагается библиотека программ, которая включает функции, необходимые для разработки собственных приложений. Некоторые программы, ориентированные на аппаратуру, т.е. на управление периферийными устройствами, были описаны в главе 3, здесь же рассматриваются программы обработки данных.

Мы не стали приводить элементарные программы, такие как инициализация блока памяти или пересылка данных одного блока памяти в другой, поскольку для их реализации нужно всего несколько команд, а решили сконцентрироваться на более сложных проблемах.

В этой главе вы найдете целую коллекцию арифметических подпрограмм, которые часто затрудняются написать даже те, кто хорошо знает машинный язык, несколько подпрограмм, связанных с обработкой прерываний, и подпрограмм, обеспечивающих реализацию асинхронных последовательных интерфейсов на микроконтроллерах, не имеющих для этого аппаратной поддержки. Если вам будет что-то непонятно в листингах, обращайтесь к главе 2, где описан синтаксис двух используемых ассемблеров.

АРИФМЕТИЧЕСКИЕ ПОДПРОГРАММЫ

Если разработка арифметических подпрограмм на языке высокого уровня не вызывает трудностей, то их написание на машинном языке требует знаний и опыта, каким бы простым ни был используемый микроконтроллер. Поэтому вам предлагается ряд готовых решений. В этом разделе содержатся арифметические подпрограммы, реализующие следующие функции:

- ◆ целочисленное умножение двух беззнаковых 8-разрядных операндов;
- ◆ умножение двух 16-разрядных операндов с учетом знаков;
- ◆ деление 16-разрядных операндов со знаками и без знаков;
- ◆ сложение и вычитание 16-разрядных операндов;
- ◆ умножение чисел с плавающей запятой;
- ◆ сложение и вычитание чисел с плавающей запятой;
- ◆ преобразование двоично-десятичных кодов (BCD) в двоичные и наоборот;
- ◆ сложение и вычитание чисел в двоично-десятичном представлении;
- ◆ извлечение квадратного корня.

Для наиболее сложных операций приводятся схемы алгоритмов.

Некоторые из этих программ допускают две формы реализации: одна обеспечивает максимальную скорость выполнения, так как включает минимум командных циклов, другая же благодаря компактности кода позволяет сократить объем занимаемой памяти, но выполняется достаточно долго.

Беззнаковое умножение 8-разрядных чисел

Эта подпрограмма вычисляет произведение двух чисел, представленных 8-разрядными двоичными кодами без знаков и формирует 16-разрядный результат. Схема ее алгоритма приведена на рис. 4.1.

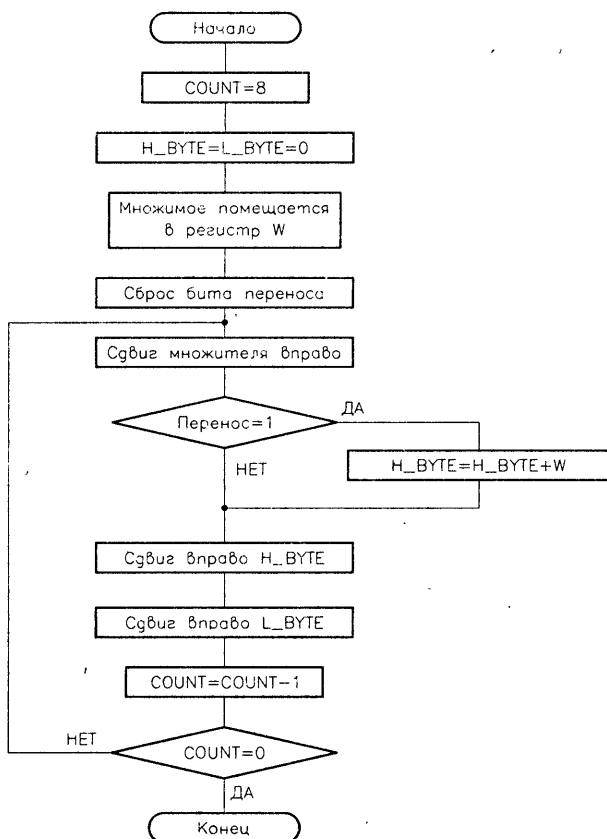


Рис. 4.1

Алгоритм умножения восьмизначных операндов

В листингах 4.1 и 4.2 вам предлагаются две версии программы:

- ♦ *быстрая версия* требует 35 ячеек памяти и исполняется за 37 командных циклов;
- ♦ *компактная версия* требует 16 ячеек памяти и исполняется за 71 командный цикл.

Перемножаемые числа задаются переменными `mulplr` и `mulcnd`. Результат будет предоставлен в двух байтах `H_byte` и `L_byte`, где `H_byte` – старший байт произведения.

Листинг 4.1

```

;*****
; Умножение 8x8 битов
; (компактная версия).
;*****
; Множитель должен быть загружен в переменную mulplr.
; Множимое должно быть загружено в переменную mulcnd.
; Результат помещается в байты H_byte и L_byte.
;
; Показатели:
; объем программной памяти: 15;
; количество циклов: 71;
; объем используемого ОЗУ: 0.
; программа соответствует инструкции по применению AN526 фирмы Microchip.
;*****
mulcnd EQU 09 ; Множимое восьми битов.
mulplr EQU 10 ; Множитель восьми битов.
H_byte EQU 12 ; Старший байт результата.
L_byte EQU 13 ; Младший байт результата.
count EQU 14 ; Счетчик циклов.
;
;
include "mpreg.h"
;***** Начало умножения *****
mpy_S cllf H_byte ;
      cllf L_byte ;
      movlw 8 ;
      movwf count ;
      movf mulcnd,w ;
      bcf STATUS,CARRY; Сброс переноса.
loop rrf mulplr ;
     btfsc STATUS,CARRY;
     addwf H_byte,Same ;

```

```

    rrf      H_byte, Same ;
    rrf      L_byte, Same ;
;
    decfsz   count       ;
    goto     loop        ;
    retlw    0           ;
;
;*****
; Тестовая программа.
;*****
main    movlw    OFF      ;
        movwf    mulplr   ; Множитель равен OFF.
        movlw    OFF      ; Множимое равно OFF.
        movwf    mulcnd   ;
;
        call     mpy_S    ; Результат OFFOFF * OFF = FE01
                           ; помещается в H_byte и L_byte.
;
self    goto     self
;
        org      01FF
        goto     main
;
        END

```

Листинг 4.2

```

;*****
; Умножение 8x8 битов
; (быстрая версия).
;*****
; Множитель должен быть загружен в переменную mulplr.
; Множимое должно быть загружено в переменную mulcnd.
; Результат помещается в байты H_byte и L_byte.
;
; Показатели:
; объем памяти: 35;
; количество циклов: 37;
; объем используемого ОЗУ: 0.
;
; Программа соответствует инструкции по применению AN526 фирмы Microchip.
;*****
mulcnd   EQU     09      ; Множимое.
mulplr   EQU     10      ; Множитель.

```

```
H_byte EQU 12 ; Старший байт результата.
L_byte EQU 13 ; Младший байт результата.
```

```
;
;
include "mpreg.h"
```

```
;***** Определение макроса сложения и сдвига вправо.*****
```

```
;
mult MACRO bit ; Начало макроса.
    btfsc mulplr, bit ;
    addwf H_byte, Same;
    rrf H_byte, Same;
    rrf L_byte, Same;
ENDM ; Конец макроса.
```

```
;***** Начало умножения *****
```

```
mpy_F clrf H_byte ;
      clrf L_byte ;
      movf mulcnd,w ; Множимое помещается в регистр W.
      bcf STATUS,CARRY; Сброс переноса.
      mult 0
      mult 1
      mult 2
      mult 3
      mult 4
      mult 5
      mult 6
      mult 7
;
      retlw 0
```

```
;*****
; Тестовая программа.
```

```
;*****
main movlw OFF
     movwf mulplr ; Множитель равен OFF.
     movlw OFF
     movwf mulcnd ; Множимое равно OFF.
;
     call mpy_F ; Результат OFF * OFF = FE01
               ; помещается в H_byte и L_byte.
self goto self
;
     org 01FF
     goto main
;
END
```

Знаковое и беззнаковое умножение 16-разрядных чисел

Эта подпрограмма вычисляет произведение двух 16-разрядных чисел и формирует 32-разрядный результат. Подпрограмма использует возможности условного ассемблирования, имеющиеся в ассемблере MPALC фирмы Microchip. Для получения беззнакового варианта умножения достаточно ассемблировать подпрограмму с командой:

```
Signed equ FALSE
```

А чтобы получать знаковое умножение, надо всего лишь заменить эту строку:

```
Signed equ TRUE
```

Две версии подпрограммы даны в листингах 4.3 и 4.4:

- ♦ *быстрая версия*: требует 240 ячеек программной памяти и выполняется за 233 командных цикла;
- ♦ *компактная версия*: требует 33 ячейки программной памяти и выполняется за 333 командных цикла.

Обе подпрограммы оперируют с двумя 16-разрядными числами, одно из которых образуют байты ACCaLO и ACCaHI, а другое – ACCbLO и ACCbHI. Байты HI – старшие байты чисел (high). Результат размещается в четырех байтах ACCbHI, ACCbLO, ACCcHI и ACCcLO, где ACCbHI – старший байт произведения.

Листинг 4.3

```
*****
; Умножение 16x16 разрядов
; (компактная версия).
*****
; Умножение  ACCb (16 битов) * ACCa (16 битов) -> ACCb,ACCc (32 бита)
;
; (A) Первый операнд помещается в байты ACCaLO и ACCaHI (16 битов).
; (B) Второй операнд помещается в байты ACCbLO и ACCbHI (16 битов).
; (C) Вызов - D_тпру.
; (D) Результат помещается в байты ACCbHI, ACCbLO, ACCcHI, ACCcLO.
;
; Показатели:
; объем памяти: 33 ячеек;
; количество циклов: 333.
; Показатели указаны для версии без учета знаков, т.е. с командой Signed equ FALSE.
;
; Программа соответствует инструкции по применению AN526 фирмы Microchip.
```

```

;*****
;
ACCaLO equ 10 ;
ACCaHI equ 11 ;
ACCbLO equ 12 ;
ACCbHI equ 13 ;
ACCCLO equ 14 ;
ACCCHI equ 15 ;
ACCDLO equ 16 ;
ACCDHI equ 17 ;
temp equ 18 ;
sign equ 19 ;
Flags equ 20 ;

include "mpreg.h"
org 0

;*****
SIGNED equ TRUE ; TRUE для версии с учетом знаков,
; FALSE для версии без учета знаков.

;*****
; Сложение с двойной точностью (ACCb + ACCa -> ACCb).
;
D_add bcf Flags,CARRY ; Сброс переноса
movf ACCaLO,w ; Сложение (ACCb + ACCa -> ACCb).
addwf ACCbLO ;
btfsc STATUS,CARRY ; Прибавление переноса.
incf ACCbHI
btfsc STATUS,CARRY
bsf Flags,CARRY
movf ACCaHI,w
addwf ACCbHI
btfsc Flags,CARRY
bsf STATUS,CARRY
retlw 0

;*****
; Умножение с двойной точностью (16x16 -> 32).
; (ACCb * ACCa -> ACCb, ACCc): результат представляется в 32 битах,
; старшие разряды помещаются в байты ACCb (ACCbHI, ACCbLO),
; младшие - в ACCc (ACCCHI, ACCcLO).
;
D_mpyS ; Результат помещается в ACCb(16 Msb) и ACCc(16 lsb).
IF SIGNED
call S_SIGN
ENDIF

;
call setup

mloop rrf ACCdHI ; Правый сдвиг.

```

```

    rrf      ACCdLO      ; Правый сдвиг.
    btfsc    STATUS,CARRY ; Проверка бита переноса.
    call     D_add       ; Вызов переменной D_add.
    rrf      ACCbHI      ; Правый сдвиг.
    rrf      ACCbLO      ; Правый сдвиг.
    rrf      ACCcHI      ; Правый сдвиг.
    rrf      ACCcLO      ; Правый сдвиг.
    decfsz   temp        ; Декрементирование и проверка на 0.
    goto     mloop       ; Повтор.
;

```

```

    IF      SIGNED
    btfss   sign,MSB
    retlw   0
    comf    ACCCLO      ; Инвертирование байта ACCbLO.
    incf    ACCCLO
    btfsc   STATUS,Z_bit
    decf    ACCcHI
    comf    ACCcHI
    btfsc   STATUS,Z_bit
neg_B     comf    ACCbLO      ; Инвертирование байта ACCbLO.
    incf    ACCbLO
    btfsc   STATUS,Z_bit
    decf    ACCbHI
    comf    ACCbHI      ; Инвертирование байта ACCbHI.
    retlw   0
    ELSE
    retlw   0
    ENDIF
;
;*****
;

```

```

setup     .movlw   .16
          movwf    temp
          movf     ACCbHI,w      ; Перемещаем байт ACCb в ACCd.
          movwf    ACCdHI
          movf     ACCbLO,w
          movwf    ACCdLO
          clrf     ACCbHI
          clrf     ACCbLO
          retlw    0
;
;*****
;

```

```

neg_A     comf     ACCaLO ; Изменение знака (дополнение) ACCaLO.
          incf     ACCaLO
          btfsc    STATUS,Z_bit
          decf     ACCaHI
          comf     ACCaHI

```

```

        retlw  0
;
;*****
; Раздел, ассемблируемый только для случая знакового умножения;
        IF      SIGNED
;
S_SIGN   movf   ACCaHI, W
        xorwt   ACCbHI, W
        movwf   sign
        btfss   ACCbHI, MSB ; Если MSB = 1, изменение знака байта ACCbHI.
        goto    chek_A      ;
;
        comf    ACCbLO      ; Отрицание байта ACCbLO.
        incf    ACCbLO
        btfsc   STATUS, Z_bit
        decf    ACCbHI
        comf    ACCbHI
;
chek_A   btfss   ACCaHI, MSB ; Если MSB = 1, изменение знака байта ACCaHI.
        retlw   0
        goto    neg_A
;
        ENDIF
;*****
; Тестовая программа.
;*****
; Загрузка констант в байты ACCa и ACCb для теста.
;
main     movlw   1
        movwf   ACCaHI
        movlw   OFF ; Загружаем константу 01FF в байт ACCa.
;
        movwf   ACCaLO
        movlw   07F
        movwf   ACCbHI
        movlw   OFF ; Загружаем константу 7FFF в байт ACCb.
        movwf   ACCbLO ;
;
        call    D_mpyS ; Результат (ACCb, ACCc) = 00FF 7E01.
;
self     goto    self
;
        org     PIC54
        goto    main
        END

```

Листинг 4.4

```

;*****
; Умножение 16х16 разрядов
; (быстрая версия).
;*****
; Умножение ACCb (16 битов) * ACCa (16 битов) -> ACCb, ACCc (32 бита).
; (A) Первый операнд помещается в байты ACCaLO и ACCaHI (16 битов).
; (B) Второй операнд помещается в байты ACCbLO и ACCbHI (16 битов).
; (C) Вызов D_MPYR.
; (D) Результат помещается в байты ACCbHI, ACCbLO, ACCcHI, ACCcLO.
;
; Показатели:
; объем памяти: 240;
; количество циклов: 233.
;
; Показатели даны для версии беззнакового умножения, т.е. с командой SIGNED equ FALSE.
;
; Программа соответствует инструкции по применению AN526 фирмы Microchip.
;*****

ACCaLO equ 10
ACCaHI equ 11
ACCbLO equ 12
ACCbHI equ 13
ACCcLO equ 14
ACCcHI equ 15
ACCdLO equ 16
ACCdHI equ 17
temp equ 18
sign equ 19
Flags equ 20
;
; include "mpreg.h"
; org 0
;*****
SIGNED equ FALSE ; TRUE для знаковой версии.
; FALSE для версии без знаков.
;*****
; Макрос умножения.
;
mulMac MACRO
LOCAL NO_ADD

rrf ACCdHI ; Сдвиг вправо.
rrf ACCdLO
;

```



```

; *****
;
; Умножение с двойной точностью (16 x 16 -> 32).
; (ACCb * ACCa -> ACCb,ACCc): результат 32 бита,
; старшие байты результата (16Msb) помещаются в байт ACCb (ACCbHI,ACCbLO),
; младшие байты результата (16Lsb) помещаются в байт ACCc (ACCcHI,ACCcLO).

```

```

IF      SIGNED
CALL    S_SIGN
ENDIF
call    setup

```

[illegible]

```

IF      SIGNED
btfss  sign,MSB
retlw  0
comf   ACCcLO      ; Изменение знака байта ACCaLO.
incf   ACCcLO
btfsc  STATUS,Z_bit
decf   ACCcHI
comf   ACCcHI
btfsc  STATUS,Z_bit
neg_B  comf   ACCbLO      ; Изменение знака байта ACCbLO.
incf   ACCbLO
btfsc  STATUS,Z_bit
decf   ACCbHI
comf   ACCbHI
retlw  0
ELSE
retlw  0
ENDIF

;
; *****
;
setup  movlw  .16
movwf  temp
movf   ACCbHI,w  ; Перемещение содержимого байта ACCb в ACCd.
movwf  ACCdHI
movf   ACCbLO,w
movwf  ACCdLO
clrf   ACCbHI
clrf   ACCbLO
retlw  0

;
; *****
;
neg_A  comf   ACCaLO      ; Изменение знака байта ACCa (-ACCa -> ACCa).
incf   ACCaLO
btfsc  STATUS,Z_bit
decf   ACCaHI
comf   ACCaHI
retlw  0

;
; *****
;
; Раздел, ассемблируемый только для случая знаковой операции.
;
IF      SIGNED
S_SIGN movf   ACCaHI,w
xorwf  ACCbHI,w
movwf  sign

```

```

    btfss ACCbHI,MSB ; Если MSB = 1, меняем знак в байте ACCb.
    goto  chek_A
    comf  ACCbLO      ; Изменяем знак байта ACCbLO.
    incf  ACCbLO
    btfsc STATUS,Z_bit
    decf  ACCbHI
    comf  ACCbHI
;
chek_A  btfss ACCaHI,MSB ; Если MSB = 1, меняем знак байта ACCaHI.
        retlw  0
        goto  neg_A
;
        ENDIF
;
;*****
; Тестовая программа.
;*****
; Загрузка констант в ACCa и ACCb для теста.
;
loadAB  movlw  1
        movwf ACCaHI
        movlw  0FFH      ; Загрузка константы 01FF в ACCa.
        movwf ACCaLO
;
        movlw  07FH
        movwf ACCbHI
        movlw  0FFH      ; Загрузка константы 7FFF в ACCb.
        movwf ACCbLO
        retlw  0
;
main    nop
;
        call  loadAB      ; Результат ACCb * ACCa -> (ACCb, ACCc).
        call  D_mpyF      ; В этом случае (ACCb, ACCc) = 00FF 7E01.
;
self    goto  self
;
        org   PIC54
        LIST  p=16c54
        goto  main
        END

```

Деление 16-разрядных чисел

Эта программа осуществляет деление двух 16-разрядных чисел и возвращает результат в двух 16-разрядных словах, одно из которых представляет частное, а другое – остаток. Речь идет, таким образом, о целочисленном делении. Программы используют возможности

условного ассемблирования, имеющиеся в ассемблере MPALC фирмы Microchip. Чтобы создать программу беззнакового деления, достаточно ассемблировать строку:

Signed equ FALSE

А чтобы получать программу деления чисел со знаками, надо заметить эту строку:

Signed equ TRUE

Быстрая и компактная версии программы даны в листингах 4.5 и 4.6:

- ♦ быстрая версия требует 370 ячеек памяти и выполняется за 263 командных цикла микроконтроллера;
- ♦ компактная версия требует 37 ячеек памяти и выполняется за 310 циклов.

В обеих программах делимое размещается в байтах ACCbHI и ACCbLO, а делитель – в ACCaHI и ACCaLO. Байты HI соответствуют старшим разрядам. Частное размещается в байтах ACCbHI и ACCbLO, а остаток – в ACCcHI и ACCcLO.

Листинг 4.5

```

;*****
; Деление 16-разрядных чисел
; (компактная версия).
;*****
; Деление: ACCb (16 бит) / ACCa (16 бит) -> ACCb (16 бит).
; Остаток помещается в ACCc (16 бит).
;
; (A) Делитель помещается в байт ACCaHI и ACCaLO (16 бит).
; (B) Делимое помещается в байт ACCbHI и ACCbLO (16 бит).
; (C) Вызов D_div.
; (D) Частное (16 бит) помещается в байты ACCbHI и ACCbLO.
; (E) Остаток (16 бит) помещается в байты ACCcHI и ACCcLO.
;
; Показатели:
; объем памяти: 037;
; количество циклов: 310.
; Показатели указаны для версии без знаков, то есть со строкой SIGNED equ FALSE.
;
; Программа соответствует инструкции по применению AN526 фирмы Microchip.
;*****
;
ACCaLO equ 10
ACCaHI equ 11
ACCbLO equ 12
ACCbHI equ 13
ACCcLO equ 14

```



```

nogo    rlf    ACCbLO
        rlf    ACCbHI
        decfsz temp    ; Цикл по всем разрядам.
        goto   dloop
;
        IF     SIGNED
        btfss  sign, msb    ; Проверка знака.
        retlw  0
        goto   neg_B    ; Меняем знак байта ACCa (-ACCa -> ACCa).
        ELSE
        retlw  0
        ENDIF
;
; *****
setup    movlw  .16    ; Для 16-ти повторов.
        movwf  temp
        movf   ACCbHI, w    ; Содержимое байта ACCb перемещается в ACCd.
        movwf  ACCdHI
        movf   ACCbLO, w
        movwf  ACCdLO
        clrf   ACCbHI
        clrf   ACCbLO
        retlw  0
;
; *****
neg_A    comf   ACCaLO    ; Меняем знак ACCa (-ACCa -> ACCa).
        incf   ACCaLO
        btfsc  STATUS, Z_bit
        decf   ACCaHI
        comf   ACCaHI
        retlw  0
;
; *****
; Раздел для деления со знаками.
;
        IF     SIGNED
;
S_SIGN   movf   ACCaHI, W
        xorwf  ACCbHI, W
        movwf  sign
        btfss  ACCbHI, msb    ; Если msb = 1, меняем знак байта ACCb.
        goto   chek_A
;
        comf   ACCbLO    ; Меняем знак байта ACCb.
        incf   ACCbLO
        btfsc  STATUS, Z_bit

```

```

        decf     ACCbHI
        comf     ACCbHI
;
chek_A   btfss   ACCaHI,MSB ; Если MSB = 1, то меняем знак байта ACCa.
        retlw   0
        goto    neg_A
;
        ENDIF
;
; *****
; Тестовая программа.
; *****
; Загрузка констант в байты ACCa и ACCb для теста.
;
main     movlw   1
        movwf   ACCaHI
        movlw   OFF      ; Загрузка константы 01FF в байт ACCa.
        movwf   ACCaLO   ;
;
        movlw   07F
        movwf   ACCbHI
        movlw   OFF      ; Загрузка константы 7FFF в байт ACCb.
        movwf   ACCbLO   ;
;
        call    D_divS    ; Остаток помещается в байтACCc.
;                          ; Здесь ACCb = 0040 и ACCc = 003F.
self     goto    self
;
        org     PIC54
        goto    main
        END

```

Листинг 4.6

```

; *****
; Деление 16 на 16 бит
; (быстрая версия).
; *****
; Деление: ACCb (16 бит) / ACCa (16бит) -> ACCb (16 бит).
; Остаток помещается в байты ACCc (16 битов).
;
; (A) Делитель помещается в байты ACCaHI и ACCaLO (16 бит).
; (B) Делимое помещается в байты ACCbHI и ACCbLO (16 бит).
; (C) Вызов D_div.
; (D) Частное (16 бит) помещается в байты ACCbHI и ACCbLO.
; (E) Остаток (16 бит) помещается в байты ACCcHI и ACCcLO.
;

```

```
; Показатели:
; объем памяти: 370;
; количество циклов: 263.
; Показатели указаны для версии без знаков, то есть со строкой SIGNED equ FALSE.
;
; Программа соответствует инструкции по применению AN526 фирмы Microchip.
```

```
;*****
;
ACCaLO equ 10
ACCaHI equ 11
ACCbLO equ 12
ACCbHI equ 13
ACCcLO equ 14
ACCcHI equ 15
ACCdLO equ 16
ACCdHI equ 17
temp equ 18
sign equ 19
;
include "nipreg.h"
org 0
;*****
SIGNED equ FALSE ; TRUE для версии с учетом знаков.
; FALSE для версии без учета знаков.
;*****
; Макрос деления.
;
divMac MACRO
LOCAL NOCHK
LOCAL NOGO
bcf STATUS,CARRY
rlf ACCdLO
rlf ACCdHI
rlf ACCcLO
rlf ACCcHI
movf ACCaHI,w
subwf ACCcHI,w ; Проверка, a > c?
btfss STATUS,Z_bit
goto NOCHK
movf ACCaLO,w
subwf ACCcLO,w ; Если младшие байты равны, проверка старших байтов.
NOCHK btfss STATUS,CARRY ; Проверка переноса.
goto NOGO
movf ACCaLO,w ; Разность (a - c) помещается в c.
subwf ACCcLO
btfss STATUS,CARRY
```



```

        decf     ACCcHI
        movwf    ACCaHI,w
        subwf    ACCcHI
        bsf      STATUS,CARRY; Установка бита переноса.
NOGO    rlf      ACCbLO
        rlf      ACCbHI
;
        ENDM
;*****
; Деление с двойной точностью (16 / 16 -> 16).
; (ACCb / ACCa -> ACCb, остаток помещается в ACCc.)
; Частное помещается в ACCb (ACCbHI,ACCbLO), а остаток в ACCc (ACCcHI, ACCcLO).
;
; До вызова этой подпрограммы надо убедиться, что делимое больше, чем делитель.
;
setup   movlw    .16           ; Для 16-ти переносов.
        movwf    temp
        movf     ACCbHI,w     ; Перемещение байта ACCb в ACCd.
        movwf    ACCdHI
        movf     ACCbLO,w
        movwf    ACCdLO
        clrf     ACCbHI
        clrf     ACCbLO
        retlw    0
;
;*****
neg_A   comf     ACCaLO       ; Меняем знак байта ACCa (-ACCa -> ACCa).
        incf     ACCaLO
        btfsc    STATUS,Z_bit
        decf     ACCaHI
        comf     ACCaHI
        retlw    0
;
;*****
D_divF  IF      SIGMED
        CALL     S_SIGN
        ENDIF
;
        call     setup
        clrf     ACCcHI
        clrf     ACCcLO
;
; Вызов макроса divMac 16 раз.
;

```

```
divMac
divMac
divMac
divMac
divMac
divMac
divMac
divMac
divMac
divMac
divMac
divMac
divMac
divMac
divMac
```

```
IF      SIGNED
btfss  sign,MSB ; Проверка знака.
retlw  0
goto   neg_B    ; Меняем знак байта ACCa (-ACCa -> ADCa).
ELSE
retlw  0
ENDIF
```

```
*****
; Раздел скомпонован только для случая деления со знаками.
```

```
IF      SIGNED
;
S_SIGN  movf    ACCaHI,W
        xorwf   ACCbHI,W
        movwf   sign
        btfss   ACCbHI,MSB ; Если старший бит равен единице,
                           ; меняем знак байта ACCb.
        goto    chek_A

        comf     ACCbLO ; Изменение байта ACCb.
        incf     ACCbLO
        btfsf    STATUS,Z_bit
        decf     ACCbHI
        comf     ACCbHI

;
chek_A  btfss   ACCaHI,MSB ; Если младший бит равен единице,меняем знак байта ACCa.
        retlw   0
        goto    neg_A

;
ENDIF
```

```

;*****
;                               Тестовая программа.
;*****
; Загрузка констант в байты ACCa и ACCb для тестирования.
main    movlw    1
        movwf    ACCaHI
        movlw    OFF        ; Загрузка константы 01FF в байт ACCa.
        movwf    ACCaLO    ;
;
        movlw    07F
        movwf    ACCbHI
        movlw    OFF        ; Загрузка константы 7FFF в байт ACCb.
        movwf    ACCbLO
        call    D_divF      ; Остаток помещается в байт ACCc.
                           ; Здесь ACCb = 0040 и ACCc = 003F.
;
self    goto     self
;
        org      PIC54
        LIST      p=16c54
        goto     main
        END

```

Сложение и вычитание 16-разрядных чисел

Эта программа осуществляет сложение и вычитание двух 16-разрядных чисел и возвращает 16-разрядный результат. Она может использоваться другими арифметическими программами. Программа представлена в листинге 4.7 и имеет следующие показатели:

- ◆ сложение требует 7 ячеек программной памяти и выполняется за 8 командных циклов;
- ◆ вычитание требует 14 ячеек памяти и выполняется за 17 командных циклов.

Операнды размещаются в байтах ACCbHI, ACCbLO и ACCaHI, ACCaLO, при вычитании вычисляется разность байтов ACCb и ACCa. Результат размещается в байтах ACCbHI и ACCbLO.

Листинг 4.7

```

;*****
; Сложение и вычитание с двойной точностью.
;*****
; Сложение: ACCb (16 бит) + ACCa (16 бит) -> ACCb. (16 бит).
; (A) Первый операнд помещается в байты ACCaLO и ACCaHI (16 бит).
; (B) Второй операнд помещается в байты ACCbLO и ACCbHI (16 бит).
; (C) Вызов D_add.

```

; (D) Результат помещается в байты ACCbLO и ACCbHI (16 бит).

; Показатели:

; размер памяти: 07;

; количество циклов: 08.

; Вычитание: ACCb (16 бит) - ACCa (16 бит) -> ACCb (16 бит).

; (A) Первый операнд помещается в байты ACCaLO и ACCaHI (16 бит).

; (B) Второй операнд помещается в байты ACCbLO и ACCbHI (16 бит.)

; (C) Вызов D_add.

; (D) Результат помещается в байты ACCbLO и ACCbHI (16 битов).

; Показатели:

; объем памяти: 14;

; количество циклов: 17.

; Программа соответствует инструкции по применению AN526 фирмы Microchip.

ACCaLO equ 10

ACCaHI equ 11

ACCbLO equ 12

ACCbHI equ 13

include "mpreg.h"

org 0

; Вычитание с двойной точностью (ACCb - ACCa -> ACCb).

D_sub call neg_A ; Отрицание байта ACCa, затем сложение.

; Сложение с двойной точностью (ACCb + ACCa -> ACCb).

D_add movf ACCaLO,w

addwf ACCbLO ; Прибавление младшего байта.

btfsc STATUS,CARRY; Добавление переноса.

incf ACCbHI

movf ACCaHI,w

addwf ACCbHI ; Прибавление старшего байта.

retlw 0

neg_A comf ACCaLO ; Меняем знак байта ACCa (-ACCa -> ACCa).

incf ACCaLO

btfsc STATUS,Z_bit

decf ACCaHI

```

    comf    ACCaHI
    retlw   0
;
;*****
;    Тестовая программа.
;*****
; Загрузка констант в ACCa и ACCb для тестирования.
;
loadAB    movlw    1
           movwf    ACCaHI
           movlw    OFF           ; Загрузка 01FF в ACCa.
           movwf    ACCaLO
           movlw    07F
           movwf    ACCbHI
           movlw    OFF           ; Загрузка 7FFF в ACCb.
           movwf    ACCbLO
           retlw    0
;
main      pop
;
           call     loadAB        ; ACCb + ACCa -> ACCb.
           call     D_add         ; Здесь ACCb = 81FE.
           call     loadAB        ; ACCb - ACCa -> ACCb.
           call     D_sub         ; Здесь ACCb = 7E00.
;
self      goto     self
;
           org      PIC54
           goto     main
           END

```

Операции с плавающей запятой

Эти программы являются наиболее сложными даже для программистов, уже привычных к ассемблеру. Напомним, что число с плавающей запятой представляется мантиссой и порядком (показателем степени).

В предложенных программах предполагается, что операнды кодируются следующим образом: мантисса – шестнадцатью битами с учетом знака и порядок – восьмью битами с учетом знака.

Алгоритмы программ сложения, вычитания и умножения схематично показаны на рис. 4.2 и 4.3.

Эти программы оптимизированы (что видно из листинга 4.8) как по объему кода, так и по времени выполнения. Программы занимают следующий объем памяти:

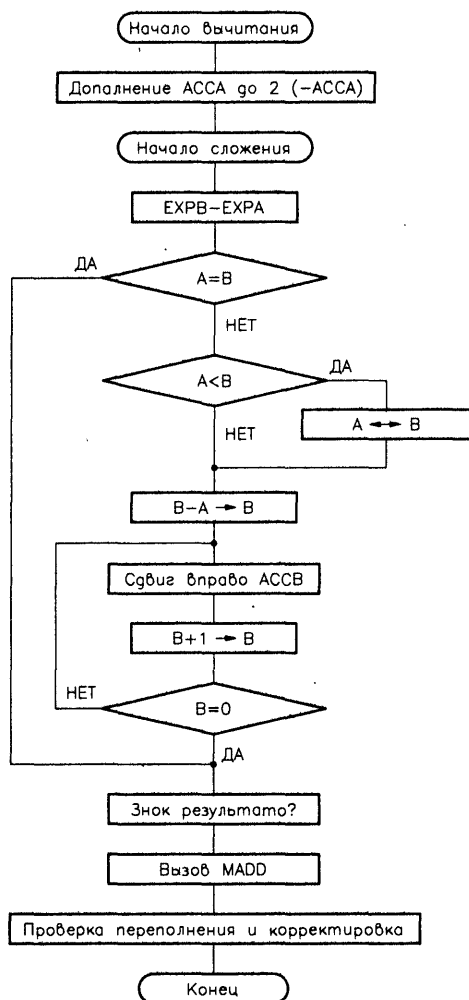


Рис. 4.2

Алгоритм сложения и вычитания чисел с плавающей запятой

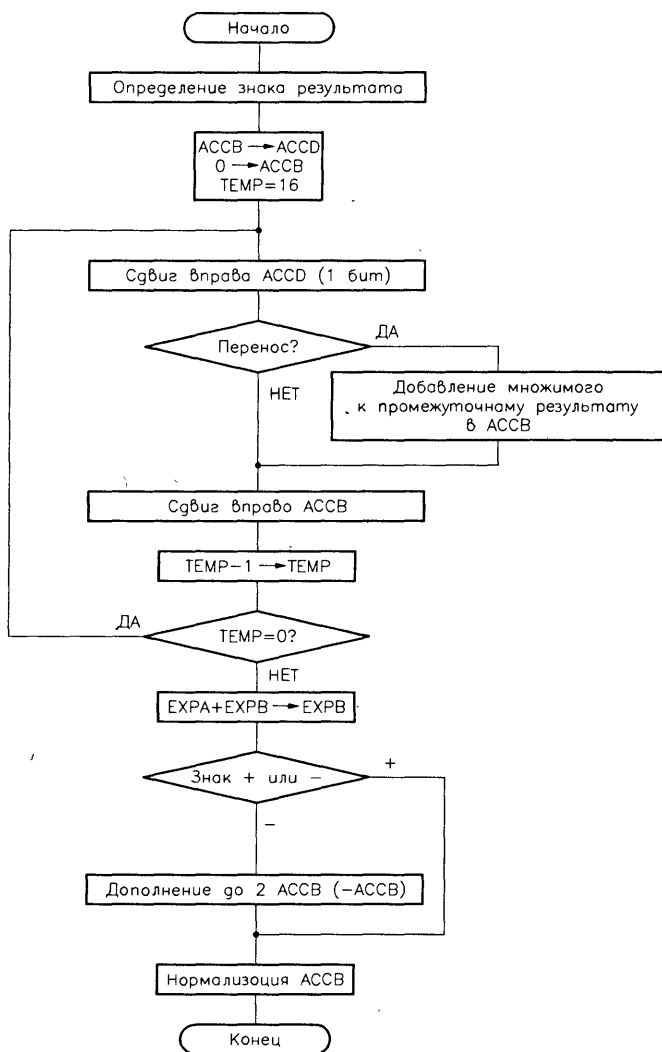


Рис. 4.3

Алгоритм умножения чисел с плавающей запятой

- ♦ сложение – 55 ячеек;
- ♦ вычитание – 61 ячейку;
- ♦ умножение – 102 ячейку.

Количество командных циклов, в течение которых выполняются данные программы, не указывается, поскольку они зависят от «размера» обрабатываемых чисел.

Программы используют в качестве мантиссы одного из операндов байты ACCaHI, ACCaLO, а в качестве порядка – байт EXPa. Соответственно для второго операнда – байты ACCbHI, ACCbLO и EXPb. Мантисса результата предоставляется в байтах ACCbHI и ACCbLO, порядок – в EXPb. При вычитании вычисляется разность байтов ACCb и ACCa. При умножении возможно, хотя это не было целью, получить мантиссу форматом в 32 бита, для чего достаточно ассемблировать программу либо со строкой:

Mode 16 equ TRUE

либо

Mode 16 equ FALSE

32-разрядная мантисса размещается в регистрах ACCbHI, ACCbLO, ACCbHI и ACCbLO, а 8-разрядный порядок – в EXPb.

Листинг 4.8

```

; *****
; Сложение, вычитание, умножение с плавающей запятой.
; *****
;
; Сложение: ACCb (16 бит) + ACCa (16 бит) -> ACCb (16 бит).
; (A) Мантисса первого операнда помещается в байты ACCaLO и ACCaHI (16 бит),
; порядок - в EXPa (8 бит).
; (B) Мантисса второго операнда помещается в байты ACCbLO и ACCbHI (16 бит),
; порядок - в EXPb (8 бит).
; (C) Вызов F_add.
; (D) Мантисса результата помещается в байты ACCbLO и ACCbHI (16 бит),
; порядок - в EXPb (8 разрядов).
;
; Объем занимаемой памяти: 55 ячеек.
; *****
;
; Вычитание: ACCb (16 бит) - ACCa (16 бит) -> ACCb (16 бит)
; (A) Мантисса первого операнда помещается в байты ACCaLO и ACCaHI
; (16 бит), порядок - в EXPa (8 бит).

```



```
; (В) Мантисса второго операнда помещается в байты ACCbLO и ACCbHI (16 бит),
; порядок - в EXPb (8 бит).
; (С) Вызов F_sub.
; (D) Мантисса результата помещается в байты ACCbLO и ACCbHI (16 бит),
; порядок - в EXPb (8 разрядов).
;
; Объем занимаемой памяти: 61 ячейка.
```

```
*****
```

```
; Умножение:
; ACCb (16 битов) EXP(b) * ACCa (16 бит) EXPa -> ACCb (16 битов) EXPb.
; (A) Мантисса первого операнда помещается в байты ACCaLO и ACCaHI (16 бит),
; порядок - в EXPa (8 бит).
; (В) Мантисса второго операнда помещается в байты ACCbLO и ACCbHI (16 бит),
; порядок - в EXPb (8 бит).
; (С) Вызов F_mpy.
; (D) Мантисса результата помещается в байты ACCbLO и ACCbHI (16 бит),
; порядок - в EXPb (8 разрядов).
;
; Получение 32-разрядного результата (мантиссы) возможно при замене строки
; Model6 equ TRUE на Model6 equ FALSE.
; В этом случае 32-разрядная мантисса результата будет
; размещена в байты ACCbHI, ACCbLO, ACCcHI, ACCcLO, а порядок - в EXPb.
;
; Объем программной памяти: 102 ячейки.
; Программа соответствует инструкции по применению AN526 фирмы Microchip.
```

```
*****
```

```
ACCaLO equ 10
ACCaHI equ 11
EXPa equ 12
ACCbLO equ 13
ACCbHI equ 14
EXPb equ 15
ACCcLO equ 16
ACCcHI equ 17
ACCdLO equ 18
ACCdHI equ 19
temp equ 1A
sign equ 1B
```

```
include "mpreg.h"
```

```
Model6 equ TRUE ; FALSE для умножения с 32-разрядной мантиссой.
org 0
```

```

;*****
; Вывчитание чисел с плавающей запятой (ACCb - ACCa -> ACCb)
;
F_sub      call    neg_A      ; Изменение знака байта ACCa и сложение.
;
F_add      movf     EXPa,w
           subwf    EXPb,w    ; Выбор наибольшего порядка.
           btfsc   STATUS,Z_bit
           goto    padd      ; Порядки равны.
           btfsc   STATUS,CARRY
           call    F_swap    ; Если A > B, то меняем их местами.
           movf    EXPa,w
           subwf    EXPb
scloop     call    shftSR
           incfsz   EXPb
           goto    scloop
           movf     EXPa,w
           movwf    EXPb
padd       movf     ACCaHI,w
           iorwf    ACCbHI,w
           movwf    sign
           call     D_add     ; Сложение с двойной точностью.
           btfss   sign,MSB
           btfss   ACCbHI,MSB
           retlw    0
           bcf     STATUS,CARRY
           incf     EXPb
           goto    shftR
;
;*****
; Вывчитание с двойной точностью (ACCb - ACCa -> ACCb).
;
D_sub      call    neg_A      ; Изменение знака байта ACCa и сложение.
;
D_add      movf     ACCaLO,w  ; Сложение (ACCb + ACCa -> ACCb).
           addwf    ACCbLO    ; Сложение младших байтов.
           btfsc   STATUS,CARRY; Прибавляем перенос.
           incf     ACCbHI
           movf     ACCaHI,w
           addwf    ACCbHI    ; Сложение старших байтов.
           retlw    0
;
;*****
shftSR     bcf     STATUS,CARRY
           btfsc   ACCbHI,MSB
           bsf     STATUS,CARRY

```

```

shftR      rrf      ACCbHI
           rrf      ACCbLO
           retlw    0
;
shftSL     bcf      STATUS,CARRY
;
           if      Model6
           rlf      ACCcLO
           rlf      ACCcHI
           endif
;
           rlf      ACCbLO
           rlf      ACCbHI
           bcf      ACCbHI,MSB
           btfsc    STATUS,CARRY
           bsf      ACCbHI,MSB
           retlw    0
;
;*****
; Умножение с плавающей запятой:
; ACCb (16 битов) EXP(b) * ACCa (16 битов) EXPa -> ACCb (16 битов) EXPb.
;
;
F_мпу
           rrf      ACCdLO
           btfsc    STATUS,CARRY ; Сложение необходимо?
           call     D_add
           rrf      ACCbHI
           rrf      ACCbLO
           rrf      ACCcHI
           rrf      ACCcLO
           decfsz    temp          ; Цикл по всем битам.
           goto     mloop

           movf      EXPa,w
           addwf     EXPb
;
           IF      Model6
           movf      ACCbHI
           btfss    STATUS,Z_bit
           goto     finup          ; Если байт ACCbHI не равен 0.
           movf      ACCbLO
           btfss    STATUS,Z_bit
           goto     Shft08         ; Если байт ACCbLO не равен 0 и ACCbHI = 0.
           movf      ACCcHI,w
           movwf     ACCbHI        ; Если ACCb = 0, тогда
                                   ; содержимое байта ACCc перемещается в ACCb.
           movf      ACCcLO,w

```

```

movwf ACCbLO
movlw .16
addwf EXPb
goto finup
;
Shft08 movf ACCbLO, w
movwf ACCbHI
movf ACCcHI, w
movwf ACCbLO
movlw .8
addwf EXPb
;
ENDIF
;
finup btfss sign,MSB
goto F_norm
;
decf ACCcLO ; Изменение знака байта ACCc.
comf ACCcLO
btfsc STATUS,Z_bit
decf ACCcHI
comf ACCcHI
btfsc STATUS,Z_bit
;
neg_B decf ACCbLO ; Изменение знака байта ACCb.
comf ACCbLO
btfsc STATUS,Z_bit
decf ACCbHI
comf ACCbHI
;
goto F_norm
;
;
;
;
*****
setup movlw .16 ; Для 16-ти сдвигов.
movwf temp
movf ACCbHI, w ; Содержимое байта ACCb перемещается в ACCd.
movwf ACCdHI
movf ACCbLO, w
movwf ACCdLO
clrf ACCbHI
clrf ACCbLO ; Обнуление байта ACCb (ACCbLO и ACCbHI).
retlw 0
;
;
;
*****

```

```

;
neg_A    comf    ACCaLO      ; Изменение знака байта ACCa (-ACCa -> ACCa).
         incf    ACCaLO
         btfsc   STATUS,Z_bit
         decf    ACCaHI
         comf    ACCaHI
         retlw   0
;

```

```

;*****
S_SIGN   movf    ACCaHI,W
         xorwf   ACCbHI,W
         movwf   sign
         btfss   ACCbHI, MSB ; Если MSB = 1, меняем знак байта ACCb.
         goto    chek_A
;

```

```

         comf    ACCbLO      ; Изменение знака байта ACCb.
         incf    ACCbLO
         btfsc   STATUS,Z_bit
         decf    ACCbHI
         comf    ACCbHI
;

```

```

;
chek_A   btfss   ACCaHI,MSB  ; Если MSB = 1, меняем знак байта ACCa.
         retlw   0
         goto    neg_A
;

```

```

;*****
; Подпрограмма нормализации.
;

```

```

; Нормализует байт ACCb для использования в операциях с плавающей запятой.
; Эта подпрограмма максимизирует мантиссу и уменьшает порядок,
; что позволяет достичь максимальной точности.
;

```

```

F_norm   movf    ACCbHI
         btfss   STATUS,Z_bit
         goto    C_norm
         movf    ACCbLO
         btfsc   STATUS,Z_bit
         retlw   0
C_norm   btfsc   ACCbHI,6
         retlw   0
         call    shftSL
         decf    EXPb
         goto    C_norm
;

```

```

; *****
;   Меняем местами байты ACCa и ACCb
;   [(ACCa, EXPa) <-> (ACCb, EXPb)].
;
F_swap
    movf    ACCaHI, w
    movwf   temp
    movf    ACCbHI, w      ; ACCaHI <-> ACCbHI.
    movwf   ACCaHI
    movf    temp, w
    movwf   ACCbHI
    movf    ACCaLO, w
    movwf   temp

;
    movf    ACCbLO, w      ; ACCaLO <-> ACCbLO.
    movwf   ACCaLO
    movf    temp, w
    movwf   ACCbLO

;
    movf    EXPa, w
    movwf   temp
    movf    EXPb, w      ; EXPa <-> EXPb.
    movwf   EXPa
    movf    temp, w
    movwf   EXPb

;
    retlw   0

;
; *****
;   Тестовая программа.
; *****
;   Загрузка констант в байты (ACCa, EXPa) и (ACCb, EXPb) для тестирования.
;
loadAB    movlw    1
          movwf    ACCaHI
          movlw    0FF      ; Загрузка константы 01FF EXP(4) в байт ACCa.
          movwf    ACCaLO
          movlw    04
          movwf    EXPa
          movlw    07F
          movwf    ACCbHI
          movlw    0FF      ; Загрузка константы 7FFF EXP(6) в байт ACCb.
          movwf    ACCbLO
          movlw    06

```

```

movwf  EXPb
retlw  0
;
main   nop
;
call   loadAB      ; ACCb(EXPb) + ACCa(EXPa) -> ACCb(EXPb).
call   F_add        ; Здесь ACCb = 403F, EXPb = 07.
;
call   loadAB      ; ACCb(EXPb) - ACCa(EXPa) -> ACCb(EXPb).
call   F_sub        ; Здесь ACCb = 7F7F, EXPb = 06.
call   loadftAB     ; ACCbf(EXPb1) * ACCa(EXPa) -> ACCb(EXPb).
call   F_mpy        ; Здесь ACCb = FF7E, EXPb = 12.
;
self   goto  self
;
org    PIC54
goto   main
END

```

Преобразование двоично-десятичных кодов в двоичные

Числа, закодированные в двоично-десятичном коде, часто используют, например, при управлении внешними цифровыми индикаторами, в том числе и семисегментными светодиодными. Листинг 4.9 демонстрирует программу преобразования двоично-десятичных чисел в двоичную форму.

Программа преобразует 5-разрядные десятичные числа, представленные в коде, в 16-разрядные двоичные числа, при этом использует 30 ячеек памяти и выполняется в течение 121 командного цикла. Число в формате размещается в регистрах R0, R1, R2, причем старший десятичный разряд располагается в правом (младшем) полубайте R0. 16-разрядное двоичное число, полученное в результате преобразования, помещается в регистрах H_byte и L_byte, где H_byte – старший байт.

Листинг 4.9

```

;*****
; Преобразование BCD-кода в двоичный.
;
; Данная программа преобразует 5-разрядное десятичное число,
; представленное в BCD-коде, в 16-разрядный двоичный код.
; Число размещается в байтах R0, R1, R2. Старший десятичный разряд
; располагается в правом полубайте R0.

```

; Двоичное число представляется в байтах H_byte и L_byte.

; Показатели:

; используемая память: 30;

; количество командных циклов: 121.

; Программа соответствует инструкции по применению AN526 фирмы Microchip.

H_byte equ 10

L_byte equ 11

R0 equ 12

R1 equ 13

R2 equ 14

H_temp equ 15 ; Временный регистр.

L_temp equ 16 ; Временный регистр.

INCLUDE "mpreg.h"

mpyl0b andlw 0F
addwf L_byte
btfsc STATUS,CARRY

incf H_byte
mpyl0a bcf STATUS,CARRY ; Умножение на 2.

rlf L_byte,w
movwf L_temp
rlf H_byte,w ; (H_temp,L_temp) = 2 * N.
movwf H_temp

bcf STATUS,CARRY ; Умножение на 2.

rlf L_byte

rlf H_byte

bcf STATUS,CARRY ; Умножение на 2.

rlf L_byte

rlf H_byte

bcf STATUS,CARRY ; Умножение на 2.

rlf L_byte

rlf H_byte ; (H_byte,L_byte) = 8 * N.

movf L_temp,w

addwf L_byte

btfsc STATUS,CARRY


```

    incf    H_byte
    movf    H_temp,w
    addwf   H_byte
    retlw   0          ; (H_byte, L_byte) = 10 * N.
;
;
BCDtoB    clrf    H_byte
           movf    R0,w
           andlw   0F
           movwf   L_byte
           call    mpy10a          ; Результат равен 10a + b.
           swapf   R1,w
           call    mpy10b          ; Результат равен 10[10a + b].
           movf    R1,w
           call    mpy10b          ; Результат равен 10[10[10a + b] + c].
           swapf   R2,w
           call    mpy10b          ; Результат равен 10[10[10a + b] + c] + d].
           movf    R2,w
           andlw   0F
           addwf   L_byte
           btfsc   STATUS,CARRY
           incf    H_byte          ; Результат равен
                                   ; 10[10[10[10a + b] + c] + d] + e.
           retlw   0          ; Преобразование закончено.
;
; *****
; Тестовая программа.
; *****
main       movlw   06
           movwf   R0          ; R0 = 06.
           movlw   55
           movwf   R1          ; R1 = 55.
           movlw   35
           movwf   R2          ; R2 = 35 (R0, R1, R2 = 6, 55, 35).
;
           call    BCDtoB      ; После преобразования H_byte = FF
                                   ; и L_byte = FF.
;
self       goto    self
;
           org     1FF
           goto    main
;
END

```

Преобразование двоичных кодов в двоично-десятичные

Эта программа осуществляет преобразование, обратное выполняемому предыдущей программой. Программа имеет две версии, одна из которых преобразует 16-разрядное двоичное число в 5-разрядное десятичное, представленное в BCD-коде, а другая – преобразование 8-разрядного двоичного числа в две цифры BCD-кода.

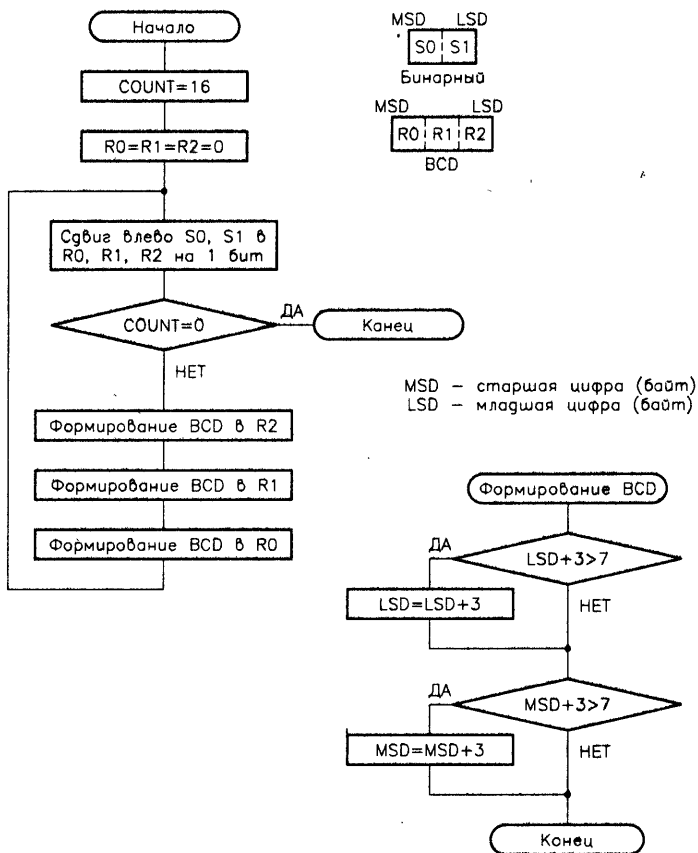


Рис. 4.4

Алгоритм преобразования двоичных чисел в двоично-десятичные

Алгоритм преобразования представлен на рис. 4.4, а код программы – в листингах 4.10 и 4.11. Показатели двух подпрограмм:

- ♦ преобразование 8-разрядного кода требует 10 ячеек памяти и выполняется за 81 командный цикл (максимум);
- ♦ преобразование 16-разрядного кода требует 30 ячеек памяти и выполняется за 719 командных циклов.

В случае преобразования 8-разрядного кода бинарное число, которое надо преобразовать, помещается в регистр W, а результат возвращается в регистрах MSD и LSD, причем старшая цифра располагается в MSD.

В случае преобразования 16-разрядного кода двоичное число, которое надо преобразовать, помещается в регистры H_byte и L_byte со старшим байтом в H_byte. Полученное в результате число в двоично-десятичном коде помещается в R0, R1 и R2. Цифра старшего разряда располагается в правом (младшем) полубайте R0.

Листинг 4.10

```
; *****
; Преобразование двоичных чисел в двоично-десятичные.
; Эта программа преобразует двоичное 8-разрядное число,
; содержащееся в регистре W, в двухразрядное десятичное число,
; представленное в BCD-коде. Младшая цифра формируется в регистре LSD,
; старшая - в MSD.
;
; Показатели:
; объем памяти: 10;
; количество командных циклов: 81 (W = 63 hex, т. е. 99 в десятичном представлении).
;
; Программа соответствует инструкции по применению AN526 фирмы Microchip.
; *****
LSD      equ      10
MSD      equ      11
;
;      INCLUDE "mpreg.h"
;
BinBCD   clrf      MSD
          movwf     LSD
gtenth   movlw     10
          subwf     LSD,W
          BTFSS     STATUS,CARRY
          goto      over
          movwf     LSD
          incf      MSD
          goto      gtenth
over     retlw     0
```

Тестовая программа

```

;*****
;
main    movlw    63      ; W = 63 hex.  $W = 63_{10} = 01100011_2$ 
        call    BinBCD   ; После преобразования MSD = 9 и LSD = 9
self    goto     self    ; (63 в hex = 99 в десятичном).
;
        org     1FF
        goto    main
;
        END
    
```

Листинг 4.11

```

;*****
; Преобразование двоичных кодов в двоично-десятичные.
; Эта программа преобразует 16-разрядное двоичное число в 5-разрядное десятичное,
; представленное в BCD коде.
; Исходное двоичное число располагается в H_byte и L_byte.
; помещается в байты R0, R1 и R2.
; Полученное в результате десятичное число
; Старшая цифра содержится в правом полубайте R0.
;
; Показатели:
; объем памяти: 35;
; количество командных циклов: 885.
;
; Программа соответствует инструкции по применению AN526 фирмы Microchip.
;*****
;
count    equ     16
temp     equ     17
H_byte   equ     10
L_byte   equ     11
R0       equ     12
R1       equ     13
R2       equ     14
;
        include "mpreg.h"
;
B2_BCD   bcf     STATUS,0 ; Сброс бита переноса.
        movlw   .16
        movwf   count
        clrf    R0
        clrf    R1
        clrf    R2
loop16   rlf     L_byte
        rlf     H_byte
    
```

```

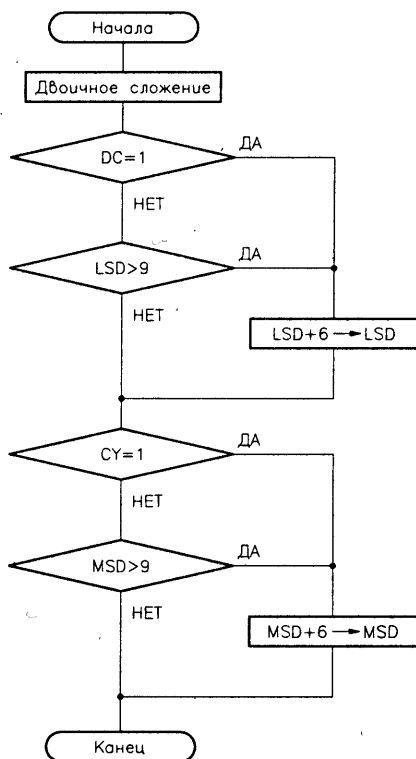
    rlf    R2
    rlf    R1
    rlf    R0
;
    decfsz count
    goto   adjDEC
    retlw  0
;
adjDEC: movlw    R2
        movlw    FSR
        call     adjBCD
        movlw    R1
        movwf    FSR
        call     adjBCD
;
        movlw    R0
        movwf    FSR
        call     adjBCD
;
        goto     loop16
;
adjBCD: movlw    3      ; W = 3 = 0011
        addwf    0,W
        movwf    temp
        btfsc    temp,3      ; Результат больше, чем 7?
        movwf    0
        movlw    30h      ; W = 30 = 0011 0010
        addwf    0,W
        movwf    temp
        btfsc    temp,7      ; Результат больше, чем 7?
        movwf    0      ; Сохранение в MSD.
        retlw    0
;
;*****
; Тестовая программа.
;*****
;
main:   movlw    0FFh
        movwf    H_byte      ; Задаем двоичное FFFF (16 бит).
        movwf    L_byte      ;
        call     B2_BCD      ; После преобразования: R0,R1,R2 = 06,55,35.
;
self:   goto     self
;
        org     1FFh
        goto     main
;
END

```

Сложение и вычитание чисел в двоично-десятичных кодах

Если ваше приложение работает в коде BCD и использует операции сложения и вычитания, целесообразнее осуществлять их непосредственно в двоично-десятичном коде, чем преобразовывать два раза (сперва BCD в двоичный, а затем обратно). В этом разделе вам предлагаются две программы, вычисляющие сумму и разность двух чисел без знака.

Алгоритмы программ приведены на рис. 4.5 и 4.6, а тексты – в листингах 4.12 и 4.13.

**Рис. 4.5**

Алгоритм сложения двоично-десятичных кодов

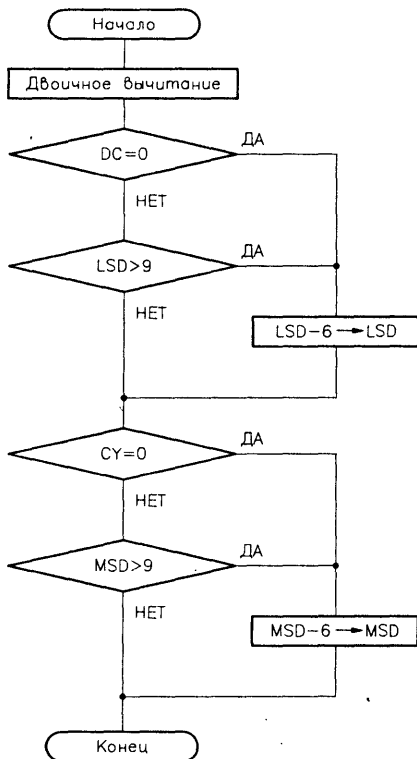


Рис. 4.6

Алгоритм вычитания чисел в двоично-десятичном коде (BCD)

Показатели программ:

- ◆ сложение требует 29 ячеек памяти и выполняется за 23 командных цикла (максимум);
- ◆ вычитание требует 31 ячейку памяти и выполняется за 21 командный цикл (максимум).

Два числа, которые надо сложить или вычесть, содержатся в байтах Num_1 и Num_2, а результат помещается в Num_2. При вычитании вычисляется разность байтов Num_2 и Num_1. В случае переполнения при выполнении операций перенос фиксируется в Num_1.

Листинг 4.12

```

;*****
; Сложение двоично-десятичных чисел без знаков.
; Слагаемые располагаются в байтах Num_1 и Num_2.

```

; Результат помещается в Num_2. Перенос помещается в Num_1.

; Показатели:

; требуемая память: 25;

; количество командных циклов: 23 (максимум).

; Программа соответствует инструкции по применению AN526 фирмы Microchip.

Num_1 equ 8 ; Первое слагаемое.

result equ 8

Num_2 equ 9 ; Второе слагаемое.

O_flow equ 9

include "mpreg.h"

BCDAdd movf Num_1,w
addwf Num_2 ; Двоичное сложение.

clrf Num_1

rlf Num_1

btfsc STATUS,DC; DC = 0?

goto adjust ; Корректировка младшей цифры (Lsd).

movlw 6

addwf Num_2 ; LSD > 9?

btfsc STATUS,CARRY

incf Num_1

btfss STATUS,DC; Проверка десятичного переноса.

subwf Num_2 ; LSD < 9.

goto over1

adjust movlw 6

addwf Num_2

over1 movlw 60 ; Добавление числа 6 к старшей цифре (MSD).

addwf Num_2

btfsc STATUS,CARRY

goto over3

btfss Num_1,0

subwf Num_2

retlw 0

over3 movlw 1

movwf Num_1

retlw 0

; Тестовая программа.

main movlw 99

movwf Num_1 ; Num_1 = 99.


```

        movlw    99
        movwf    Num_2          ; Num_2 = 99.
;
call     BCDAdd                ; После сложения Num_2 = 98
                                ; и Num_1 = 01 (99 + 99 = 198).
;
self     goto     self
        org      1FF
        goto     main
;
        END

```

Листинг 4.13

```

; *****
; Вычитание двоично-десятичных чисел без знаков.
; Два числа DCB для вычитания помещаются в байты Num_111111 и Num_2.
; Результат (Num_2 - Num_1) помещается в Num_2.
; Перенос помещаются в Num_1.
;
; Показатели:
; требуемая память: 31;
; количество командных циклов: 21 (максимум).
;
; Программа соответствует инструкции по применению AN526 фирмы Microchip.
; *****
Num_1    equ     8              ; Размещение байта Num_1.
result   equ     8
Num_2    equ     9              ; Размещение байта Num_2.
O_flow   equ     9
;
        include  "mpreg.h"
;
BCDSub   movf     Num_1,w
        subwf    Num_2
        clrf     Num_1
        rlf      Num_1
        btfss    STATUS,DC
        goto     adjst1
        btfss    Num_2,3        ; Корректировка младшей цифры (LSD)
                                ; результата.
        goto     Over_1
        btfsc     Num_2,2
        goto     adjst1        ; Корректировка LSD результата.
        btfss     Num_2,1      ; Бит 1 Num_2 = 1 ?

```

```

adjst1    goto    Over_1      ; Если да, переходим к старшей цифре (MSD).
          movlw   6
          subwf   Num_2
Over_1     btfss   Num_1,0     ; CY = 0?
          goto    adjst2      ; Да, уточнение MSD результата.
          clrf    Num_1
          btfss   Num_2,7     ; Нет, проверяем: MSD > 9?
          retlw   0
          btfsc   Num_2,6
          goto    adjst2
          btfss   Num_2,5
          retlw   0
adjst2     movlw   60          ; Добавление цифры 6 к MSD.
          subwf   Num_2
          clrf    Num_1
          btfss   STATUS,CARRY ; Проверка переполнения.
          retlw   0
          movlw   1
          movwf   Num_1
Over       retlw   0
;
; *****
; Тестовая программа.
; *****
;
main       movlw   23
          movwf   Num_1       ; Num_1 = 23.
          movlw   99
          movwf   Num_2       ; Num_2 = 99.
          call    BCDSUB      ; После вычитания Num_2 = 76 (99 - 23) и Num_1 = 0
                               ; (указывает, что результат больше нуля).
;
          movlw   99
          movwf   Num_1       ; Num_1 = 99.
          movlw   0
          movwf   Num_2       ; Num_2 = 0.
;
          call    BCDSUB      ; После вычитания Num_2 = 1 и Num_1 = 1
                               ; (показывает, что результат меньше нуля).
;
self       goto    self
;
          org     1FF
          goto    main
;
          END

```

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРЕРЫВАНИЙ МИКРОКОНТРОЛЛЕРОВ 16C5X

Подсемейство 16C5X PIC-микроконтроллеров не обладает аппаратными средствами прерываний, что ограничивает их применение.

В этом разделе рассказано, как реализовать псевдопрерывание с помощью программы, реагирующей на изменение состояния линий портов. Получаемые в результате показатели вполне приемлемы, так как достигается время реакции от 10 до 20 мкс.

Принцип действия предлагаемой программы следующий: каждая линия параллельного порта уподоблена входу прерывания, и изменение ее состояния может инициировать начало подпрограммы обработки прерывания, индивидуальной для каждой линии. Переход к соответствующей подпрограмме может осуществляться посредством модификации содержимого программного счетчика (PC) PIC-микроконтроллера 16C5X, модификация же происходит по результату считывания состояния порта. Например, когда в качестве входов прерывания использованы две линии параллельного порта, фрагмент программы может иметь следующий вид:

MOVWF CONDTN, W	; Загружаем в регистр W текущее состояние порта.
ANDLW 3	; Маскируем шесть старших битов.
ADDWF 2, 1	; Прибавляем содержимое регистра W к PC.
GOTO MAIN	; Возвращаемся к основной программе, если нет
	; логической единицы ни на одном из входов.
GOTO INT1	; Если единица подана на линию 0 порта.
GOTO INT2	; Если единица подана на линию 1 порта.
GOTO INT3	; Если единица подана на линии 0 и 1 одновременно.

В данном случае состояние двух псевдовходов прерывания используется для формирования двухразрядного двоичного числа, которое добавляется к содержимому PC для того, чтобы с его помощью перейти (goto) к подпрограмме обработки соответствующего прерывания.

Обычно все гораздо сложнее, так как во многих приложениях необходимо обеспечивать быструю реакцию на прерывание, в противном случае механизм прерывания не будет эффективным. Кроме того, требуется проводить опрос линий прерывания в строго определенные интервалы времени.

В общем случае для программной организации подобной системы прерываний можно использовать два механизма: разделение главной программы на программные секции (разделы) и таймер реального времени.

Соответствующий алгоритм представлен на рис. 4.7. Таймер реального времени (RTCC) служит счетчиком истекшего времени. Переходы к подпрограммам прерываний и различным разделам главной программы реализованы с использованием принципа таблицы переходов, который был рассмотрен ранее.

При отсутствии прерывания в каждом цикле алгоритма текущее содержимое регистра переходов добавляется к содержимому программного счетчика (PC), и благодаря этому осуществляется переход к соответствующему разделу главной программы. После выполнения данного раздела содержимое регистра переходов

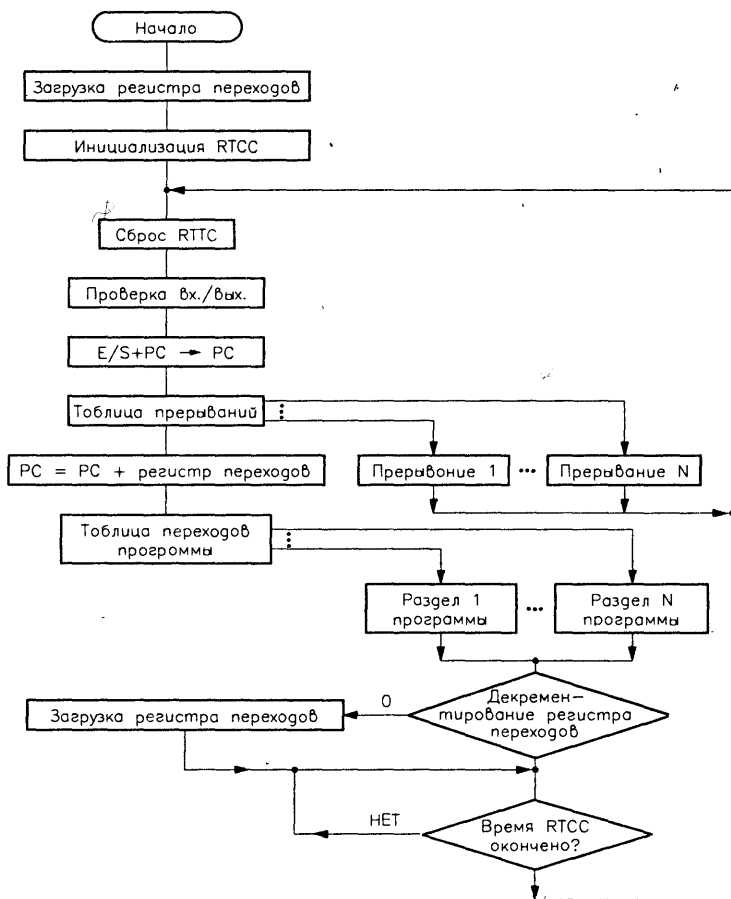


Рис. 4.7

Алгоритм программной реализации системы прерываний

декрементируется, и цикл повторяется. Таким образом поочередно будут исполняться все разделы.

Чтобы определить период опроса линий прерывания, можно использовать таймер. Так, по приведенному алгоритму новый опрос линий прерывания будет производиться после завершения счета таймера. Интервал опроса может быть определен и табличным способом:

```

MOVLW TEST          ; Необходимое число командных циклов.
SUBWF RTCC,W        ; Оставшееся число командных циклов.
ADDWF 2,1           ; Добавляем это число к содержимому PC.
NOP
...
GOTO DEBUT          ; Количество NOP равно содержимому TEST.
                   ; Максимальное время ожидания истекло,
                   ; возвращение к главной программе.

```

Единственный параметр, который требует определения, это TEST. Его значение зависит от продолжительности выполнения всех разделов программы (в командных циклах).

В листинге 4.14 представлен пример программы, реализующей приведенный алгоритм. Она рассчитана на опрос двух линий прерывания с разбивкой главной программы на четыре раздела и содержит восемь дополнительных инструкций, необходимых для завершения отсчета времени.

Листинг 4.14

```

; Программная реализация прерываний.
LIST P=16C54.
;
; Принцип программной реализации системы прерываний.
; Соответствует инструкции по применению AN514 фирмы Microchip.
;
BRANCH EQU 8
CNDTN EQU 9
IO EQU 0A
TEMP EQU 0B

SETUP CLRF CNDTN
      MOVLW 4
      MOVWF BRANCH ; Четыре раздела в главной программе.
      MOVLW 8

```

```

OPTION          ; Конфигурируем RTCC для счета
                ; командных циклов.

START  CLRF     1          ; Сброс RTCC.
       MOVF     6,W        ; Считывание входов/выходов.
       MOVWF    IO
       IORWF    CNDTN,W    ; Определяем переход в таблице прерываний.
       MOVWF    TEMP      ; Прерывание при изменении
       MOVF     CNDTN,W    ; уровня линии с 0 на 1.
       SUBWF    TEMP, 1    ; Уравнение для номера прерывания:
       MOVF     10,W       ; (ES + CNDTN) - CNDTN = ПЕРЕРЫВАНИЕ,
       MOVWF    CNDTN     ; где ES - текущее состояние входов
       MOVF     TEMP,W     ; и CNDTN - предыдущее состояние.
       ANDLW    3         ; Маскируем шесть старших битов.
       ADDWF    2,1        ; Прибавляем состояние входов к PC.
       GOTO     MAIN      ; Для состояния = 00.
       GOTO     INT1      ; Для состояния = 01.
       GOTO     INT2      ; Для состояния = 10.
       GOTO     INT3      ; Для состояния = 11.

INT1    NOP           ; Программа первого прерывания.
       GOTO     START

INT2    NOP           ; Программа второго прерывания.
       GOTO     START

INT3    NOP           ; Программа третьего прерывания.
       GOTO     START

MAIN    MOVF     BRANCH,W  ; Прибавление текущего содержимого
       ADDWF    2,1        ; регистра переходов к PC.
       NOP
       GOTO     MAIN4     ; Таблица переходов главной программы.
       GOTO     MAIN3
       GOTO     MAIN2
       GOTO     MAIN1

MAIN1   NOP           ; Главная программа, раздел 1.
       GOTO     BRNCHK

MAIN2   NOP           ; Главная программа, раздел 2.
       GOTO     BRNCHK

MAIN3   NOP           ; Главная программа, раздел 3.
       GOTO     BRNCHK

MAIN4   NOP           ; Главная программа, раздел 4.
       GOTO     BRNCHK

BRNCHK  DECFSZ   BRANCH, 1 ; Декрементация регистра переходов и тест на 0.
       GOTO     TIMCHK

```

```

        MOVLW    4
        MOVWF    BRANCH      ; Загрузка регистра переходов.

TIMCHK  MOVLW    D'41'        ; Проверка: RTCC равно 50 (50 - 7)
        SUBWF    1,W          ; Определение времени ожидания.
        ADDWF    2,1          ; Добавление времени ожидания к PC.
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        GOTO     START
        END

```

ПРИНЦИП МНОГОЗАДАЧНОСТИ

В некоторых случаях необходимо, чтобы микроконтроллер работал в режиме многозадачности. Это бывает нужно, например, когда при выполнении устройством некоторых функций ЦПУ долго находится в состоянии ожидания. «Потерянное время» целесообразно использовать для выполнения других функций.

Возможность прямой модификации содержимого программного счетчика, которой обладает PIC-микроконтроллер (см. предыдущий раздел), может быть применена и для поддержки многозадачного режима.

На рис. 4.8 показана простая схема, для управления которой написана следующая программа. PIC-микроконтроллер 16C54 заставляет мигать восемь светодиодов с различной скоростью, при этом управление миганием каждого из них является отдельной задачей.

Как видно из листинга 4.15, принцип построения программы достаточно прост. Счетчик задач task инкрементируется при каждом вызове очередной задачи. Сам вызов (переход) осуществляется табличным способом за счет добавления содержимого счетчика задач к содержимому программного счетчика PC. Таблица перехода отправляет в адрес начала каждой задачи.

Данная программа имеет некоторые ограничения. Она не позволяет определить для каждой задачи ни точное, ни максимальное время выполнения. Решить проблему можно с помощью таймера реального времени (RTCC), позволяющего выяснить время выполнения каждой задачи. Затем на основе этих значений будет легко определить суммарное время выполнения всех задач.

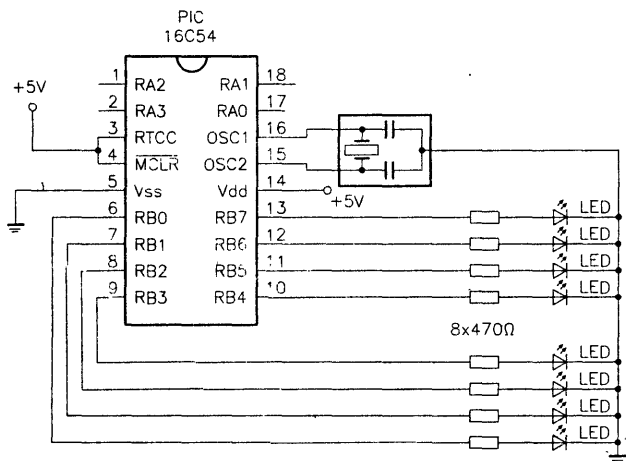


Рис. 4.8

Схема, демонстрирующая работу PIC-микроконтроллера
в многозадачном режиме

Листинг 4.15

```

; Пример функционирования в многозадачном режиме.
; Соответствует инструкции по применению фирмы Parallax.
; Эта программа - пример организации многозадачной работы.
; Она управляет восьмью задачами, которые заставляют мигать
; с различной частотой восемь светодиодов, связанных с RB0 - RB7. Скорость (темп)
; мигания определяется переменной величиной ticks.
;
; Определение констант.
LEDs      =      rb
;
; Определение переменных величин.
org      8
task      ds      1      ; Число задач.
tasks     ds      1      ; Системный таймер.
time0     ds      1      ; Таймер 0.
time1     ds      1      ; Таймер 1.
time2     ds      1      ; Таймер 2.
time3     ds      1      ; Таймер 3.
time4     ds      1      ; Таймер 4.
time5     ds      1      ; Таймер 5.
time6     ds      1      ; Таймер 6.
time7     ds      1      ; Таймер 7.
; Определение типа используемого PIC-микроконтроллера.
device    pic16c54, xt_osc, wdt_off, protect_off
reset     start
; Начало программы в ПЗУ с адреса 0.
org      0
    
```



```

start    mov     !rb, #00000000b ; Все линии порта RB - выходы.
          mov     task, #7         ; Определение числа задач.
          clr     ticks            ; Сброс системного таймера.
          clr     LEDs            ; Сброс светодиодов (LEDs).

system   inc     task             ; Следующая задача.
          cjne    task, #8, :cont  ; Все задачи выполнены?
          clr     task            ; Если да, сброс счетчика задач
          inc     ticks           ; и приращение системного таймера.
:cont    mov     w, task
          jmp     pc+w            ; Табличный переход
          jmp     task0           ; к задаче N.
          jmp     task1
          jmp     task2
          jmp     task3
          jmp     task4
          jmp     task5
          jmp     task6
          jmp     task7

task0    cjne    ticks, #255, :cont ; Счет до 255.
          inc     time0           ; Приращение таймера 0.
          cjne    time0, #3, :cont ; Счет до 3.
          clr     time0          ; Сброс таймера 0 и изменение
          xor     LEDs, #00000001b ; состояния светодиодов.
:cont    jmp     system

task1    cjne    ticks, #255, :cont
          inc     time1
          cjne    time1, #8, :cont
          clr     time1
          xor     LEDs, #00000010b
:cont    jmp     system

task2    cjne    ticks, #255, :cont
          inc     time2
          cjne    time2, #6, :cont
          clr     time2
          xor     LEDs, #00000100b
:cont    jmp     system

task3    cjne    ticks, #255, :cont
          inc     time3
          cjne    time3, #11, :cont
          clr     time3
          xor     LEDs, #00001000b
:cont    jmp     system

```

```
task4.    cjne    ticks, #255, :cont
          inc     time4
          cjne    time4, #12, :cont
          clr     time4
          xor     LEDs, #00010000b
:cont     jmp     system

task5     cjne    ticks, #255, :cont
          inc     time5
          cjne    time5, #4, :cont
          clr     time5
          xor     LEDs, #00100000b
:cont     jmp     system

task6     cjne    ticks, #255, :cont
          inc     time6
          cjne    time6, #23, :cont
          clr     time6
          xor     LEDs, #01000000b
:cont     jmp     system

task7     cjne    ticks, #255, :cont
          inc     time7
          cjne    time7, #9, :cont
          clr     time7
          xor     LEDs, #10000000b
:cont     jmp     system
```

РАСШИРЕНИЕ СТЕКОВОЙ ПАМЯТИ МИКРОКОНТРОЛЛЕРОВ 16C5X

Стековая память подсемейства 16C5X PIC-микроконтроллеров имеет только два уровня. Поэтому данные микроконтроллеры не позволяют выполнять программы с уровнем вложенности подпрограмм более двух, поскольку при этом теряются адреса возврата.

Программа, которая вам предлагается, показывает, как реализовывать стековую память с пятью уровнями, разрешающую обращение к пяти вложенным подпрограммам. Этот пример, конечно, может быть расширен, но поскольку размер оперативной памяти (RAM) PIC-микроконтроллеров 16C5X невелик, советуем вам при разработке приложений просчитывать максимальное число вложенных подпрограмм, предусмотренных в вашей программе, чтобы наилучшим образом приспособить предлагаемый пример к конкретным требованиям.

Листинг 4.16 демонстрирует, как можно использовать в программе макрокоманду NCALL вместо обычной CALL. Эта макрокоманда сохраняет содержимое программного счетчика PC в стековой памяти и затем выполняет обычную команду вызова подпрограммы CALL. В конце каждой подпрограммы вместо типовой команды возврата (например, RETLW k) нужно поместить строку GOTO RTRN. RTRN – это подпрограмма, выполняющая функцию, обратную макрокоманде NCALL. Она загружает программный счетчик значением, полученным из расширенного стека, и таким образом осуществляет действительный возврат из подпрограммы.

Чтобы пример был более наглядным, листинг включает главную программу, начинающуюся с макрокоманды START, которая вызывает три вложенные подпрограммы TOM, DICK и HARRY.

Внимание! Макрокоманда NCALL использует регистр косвенной адресации FSR (регистр f4). Если этот регистр задействован в вашей программе, нужно его сохранить, прежде чем запускать макрокоманду NCALL.

Листинг 4.16

```

;*****
; Программа управления многоуровневой стековой памятью
; для обеспечения вложенности более двух подпрограмм.
; В этой демонстрационной программе вызываемые подпрограммы
; не выполняют никаких действий (используются команды NOP).
; Программа соответствует инструкции по применению AN527 фирмы Microchip.
;*****
;
PC      equ      2
FSR     equ      4
;
;      org      8
STACK  res      5      ; Определяем объем стековой памяти равным пяти.
;
;      org      01ff
;      goto    START
;      org      0
;
INIT    movlw    STACK  ; Загружает "STACK" в качестве указателя.
        movwf    FSR    ; /
        goto    START  ; /

```

```

;*****
; Определение макроса NCALL для использования
; вместо команды CALL.
;
NCALL      macro    LABEL
            movf     PC,w      ; Сохранить PC в-стеке.
            movwf    0         ; /
            incf     FSR       ; Инкрементируем указатель стека.
            goto     LABEL     ; Переход к подпрограмме.
            endm

; Возвращение после NCALL
;
RTRN       decf     FSR       ; Указатель на последнем месте стека.
            movlw    3
            addwf    0,w
            movwf    PC       ; Перезагрузка PC.
;
;*****
;
START      nop
            NCALL    TOM
            nop       ; Код главной программы.
            nop       ; /
            sleep

;
TOM         nop
            NCALL    DICK
            nop       ; Код подпрограммы TOM.
            goto     RTRN

;
DICK        nop
            NCALL    HARRY
            nop       ; Код подпрограммы DICK.
            goto     RTRN

;
HARRY       nop       ; Код подпрограммы HARRY.
            nop       ; /
            goto     RTRN
;
            end

```

ПЕРЕДАЧА АСИНХРОННОЙ ПОСЛЕДОВАТЕЛЬНОСТИ ПРИ ОТСУТСТВИИ ПОСЛЕДОВАТЕЛЬНОГО ПОРТА

За исключением 16C63, 16C65, 16C73 и 16C74 микроконтроллеры рассматриваемых в данной книге подсемейств не имеют последовательного асинхронного интерфейса. Однако он очень необходим, когда требуется взаимодействие с терминалом или другим информационным оборудованием.

В этом разделе будет рассказано, как программным способом реализовать передачу и прием асинхронной последовательности данных, используя любую линию параллельного порта. Поскольку PIC-микроконтроллеры подсемейства 16CXX являются быстродействующими, при определенных условиях может быть достигнута скорость передачи 19200 бод. Многие микроконтроллеры, также реализующие эту функцию с помощью программного обеспечения, ограничены скоростью 4800 и даже иногда 2400 бод!

Типичный вид байтовой посылки через асинхронный последовательный интерфейс представлен на рис. 4.9. В примере предполагается, что работа ведется в 8-битном режиме передачи данных и без контроля четности, чтобы немного упростить задачу.

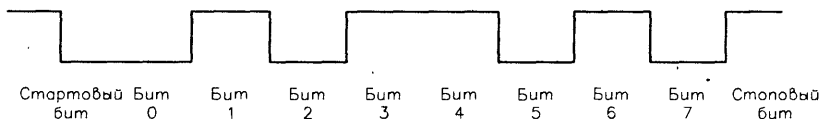


Рис. 4.9

Вид байтовой посылки при асинхронной передаче

После стартового бита следуют восемь битов данных (младшие разряды впереди) и за ними один или несколько стоп-битов.

В качестве примера реализации рассмотрим схему, показанную на рис. 4.10, которая осуществляет прием асинхронной последовательности и индикацию принятого кода в двоичной форме с помощью восьми светодиодов, подключенных к порту В. Соответствующая программа представлена в листинге 4.17.

После конфигурирования портов эта программа входит в цикл ожидания, предназначенный для обнаружения передачи стартового бита. Затем программа ожидает в течение времени, равного половине битовой посылки, чтобы зарегистрировать логическое значение принятого сигнала в середине теоретического битового интервала.


```

;
start_delay  mov     delay_cntr, #half_bit
:loop       nop
            djnz     delay_cntr, :loop
            ret
    
```

Если стартовый бит обнаружен, осуществляем прием данных, повторяя восемь раз следующие операции:

- ◆ ожидание времени регистрации бита;
- ◆ считывание бита и запись его в бит переноса, а затем в байт приема с помощью команды сдвига вправо;
- ◆ декрементирование счетчика битов;
- ◆ повторение процесса до тех пор, пока этот счетчик не станет равным нулю.

Затем программа посылает полученный байт на светодиоды.

Правильная работа этой программы зависит только от константы `bit_K`. Эта константа определяется частотой тактового генератора используемого PIC-микроконтроллера и скоростью передачи принимаемой последовательности. В табл. 4.1 приводятся используемые значения для четырех различных частот тактового генератора и для четырех наиболее распространенных скоростей передачи асинхронных последовательностей. Как видно из таблицы, при использовании генератора на 8 МГц с этой подпрограммой можно работать на скорости до 19200 бод.

Таблица 4.1

Значение константы `bit_K` в зависимости от скорости передачи асинхронной последовательности и частоты тактового генератора

Частота таймера (МГц)	Скорость передачи данных						
	300 (3,33 мкс)	600 (1,66 мкс)	1200 (833 мкс)	2400 (417 мкс)	4800 (208 мкс)	9600 (104 мкс)	19200 (52 мкс)
1	206	102	50	24	–	–	–
2	–	206	102	50	24	–	–
4	–	–	206	102	50	24	–
8	–	–	–	206	102	50	24

Передача асинхронной последовательности данных также не вызывает затруднений. Схема рис. 4.11 и листинг 4.18 демонстрируют интересный пример, который помимо подпрограмм передачи данных содержит главную программу, передающую слово Parallax.

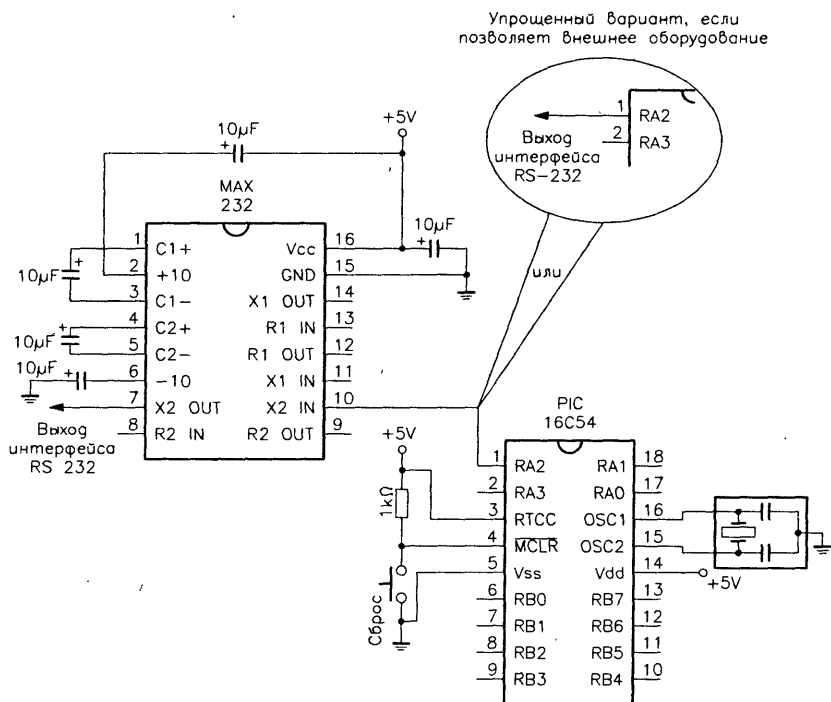


Рис. 4.11

Схема программной реализации передачи асинхронной последовательности

Программа включает две основные подпрограммы: Again и xmt_byte. Первая устанавливает счетчик битов, считывает буквы, подлежащие передаче из таблицы string, и помещает их в регистр W. Подпрограмма xmt_byte выполняет собственно передачу. Она начинается тем, что выдает логический ноль на выбранный для передачи выход, чтобы сгенерировать стартовый бит, а затем в цикле осуществляет следующие операции:

- ◆ сдвиг вправо байта, подлежащего передаче (при котором текущий передаваемый бит пересылается в бит переноса);
- ◆ пересылку бита переноса в соответствующий разряд параллельного порта;
- ◆ отсчитывает интервал передачи бита;
- ◆ уменьшает значение счетчика битов;
- ◆ повторяет процесс, пока значение счетчика не станет равно нулю.

Как и в случае приема, передача сигналов основывается только на правильных значениях константы `bit_K`. В обеих программах (приема и передачи сигналов) значения константы `bit_K` идентичны, поэтому и в этом примере вы можете использовать табл. 4.1.

Листинг 4.18

```
; Передача асинхронной последовательности.
; Соответствует инструкции по применению фирмы Parallax.
; Эта программа выдает данные в виде асинхронной последовательности через RA2.
; Скорость передачи определена константой bit_K.
; Формат передачи - 8 битов данных и 1 стоп-бит без бита четности.
;
bit_K      =      24      ; Значение выбирается в зависимости от скорости.
serial_out =      ra.2

; Определение переменных величин.

org      8

delay_cntr ds      1      ; Счетчик времени.
bit_cntr   ds      1      ; Число переданных битов.
msg_cntr   ds      1      ; Смещение в строке (таблице).
xmt_byte   ds      1      ; Байт, который надо передать.

; Программа в ПЗУ с адреса = 0.

org      0

; Определение PIC-микроконтроллера.
; Внимание! Используйте только версии
; с кварцевым генератором для лучшей временной стабильности.

device pic16c54,xt_osc,wdt_off,protect_off
reset begin

begin      mov      !ra, #00000000b; Все разряды порта A - выходы.
           mov      msg_cntr, #0    ; Последовательность 9 букв (0 - 8).
```

```

:again      mov     bit_cntr, #8      ; Счетчик битов равен 8.
            mov     w,msg_cntr      ; Указатель в последовательности.
            call    string          ; Считывание очередного символа.
            mov     xmt_byte,w      ; Символ, который надо передать.
            clrb    serial_out      ; Заменять на sktb serial_out,
                                   ; если передача без инверсии.
            call    bit_delay       ; Стартовый бит.
:xmit       rr      xmt_byte        ; Ротация бита, который надо передать в С.
            movb    serial_out,c    ; Заменять movb serial_out,/c
                                   ; если прямое соединение.
            call    bit_delay
            djnz    bit_cntr,:xmit  ; Передача окончена?
            setb    serial_out      ; Заменить clrb serial_out,
                                   ; если передача без инверсии.
            call    bit_delay       ; Стоп-бит.
            inc     msg_cntr        ; Инкрементирование указателя строки.
            cjbe    msg_cntr, #8,:again
            ; Переданы все символы?
:endless    jmp     :endless        ; Бесконечный цикл, выход при сбросе.

```

; Чтобы изменить скорость передачи, измените значение bit_K.

```

bit_delay  mov     delay_cntr, #bit_K
:loop      nop
            djnz    delay_cntr, :loop
            ret
string jmp  pc+w      ; Последовательность букв "Parallax".
            retw     'P', 'a', 'r', 'a', 'l', 'l', 'a', 'x', '10'

```

Г Л А В А 5

ГОТОВЫЕ РЕШЕНИЯ

В этой главе:

Часы с будильником

Реализация шины I²C

Четырехканальный вольтметр
со светодиодной индикацией

Микрокомпьютер, программируемый на Basic

В предыдущих главах уже рассматривалось немало примеров исполнения PIC-микроконтроллеров подсемейства 16CXX и соответствующее программное обеспечение. Теперь вам предлагается несколько конкретных устройств.

Эти устройства были выбраны по двум причинам. Во-первых, они представляют практический интерес и, во-вторых, используют один или несколько методов, не описанных в предыдущих главах.

ЧАСЫ С БУДИЛЬНИКОМ

Ниже приведен полный исходный текст соответствующей программы, поскольку без этого наш рассказ потерял бы часть своего смысла.

Программное обеспечение для часов с будильником включает в себя несколько очень часто употребляемых подпрограмм:

- ◆ подпрограмму управления матричной клавиатурой, работающую совместно с 4-разрядным цифровым индикатором от одних линий микроконтроллера;
- ◆ подпрограмму динамической индикации;
- ◆ подпрограмму отсчета времени, использующую таймер реального времени (RTCC) и кварцевый синхрогенератор микроконтроллера.

Схема устройства представлена на рис. 5.1. Что касается клавиатуры и индикатора, то их схема практически идентична той, что приведена в главе 3. Если вы посчитаете, что несмотря на относительно высокое значение сопротивления резисторов R12 – R15 существует слишком сильное влияние клавиатуры на индикатор, вы можете использовать схему, представленную на рис. 5.2, которая благодаря диодам устраняет эту связь.

Частота тактового генератора микроконтроллера равна 4,096 МГц. Она стабилизируется кварцевым резонатором и позволяет для часов получить частоту, период которой является кратной частью секунды. Частота командных циклов микроконтроллера с данным генератором составляет 1,024 МГц. Задав коэффициент деления предварительного делителя RTCC равным 32, получаем частоту 32 КГц, которая, если установить модуль счета RTCC равным 96, позволяет получить период в 5 мс. Эти сигналы будут использованы для отсчета секунд и далее – минут и часов, а также для управления индикацией и клавиатурой, как видно из временной диаграммы, приведенной на рис. 5.3.



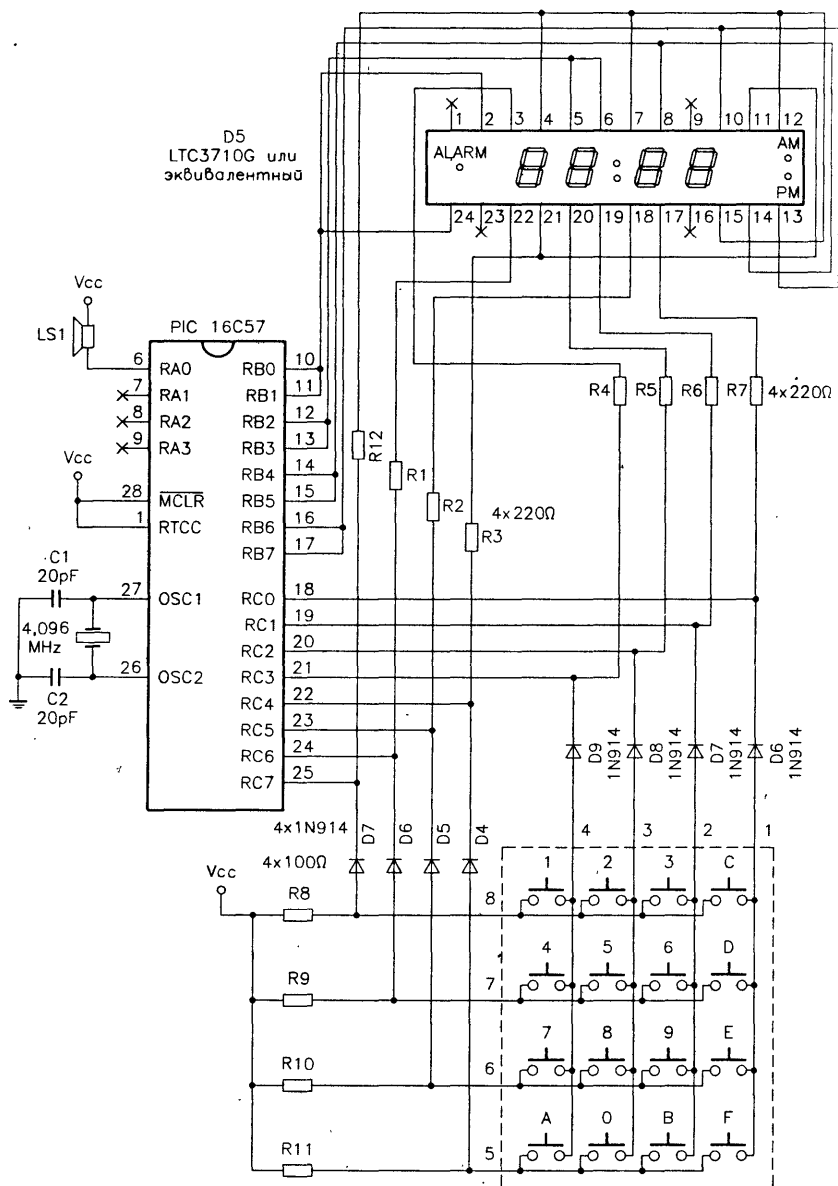


Рис. 5.2

Принципиальная схема будильника с развязкой клавиатуры через диоды

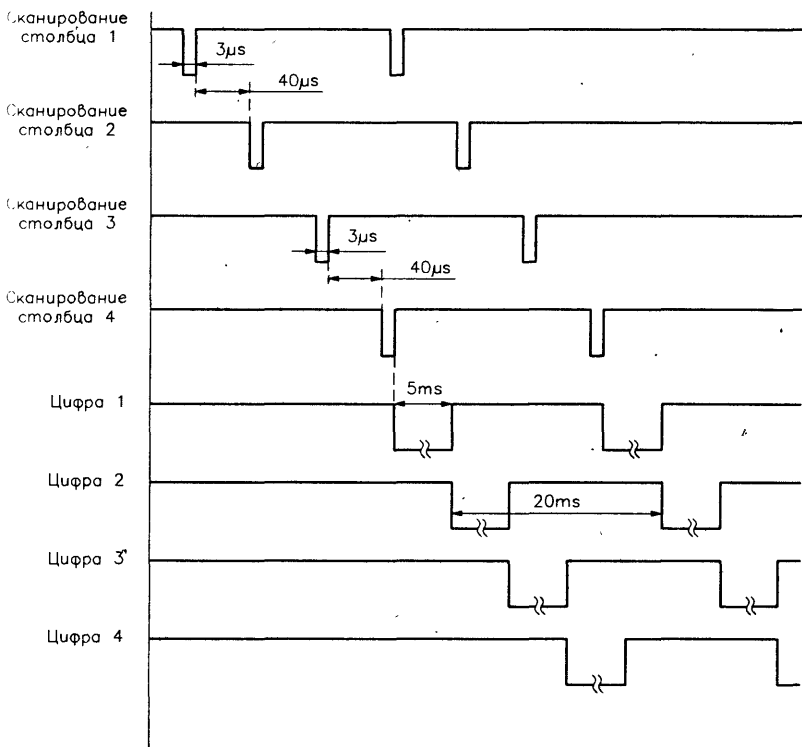


Рис. 5.3

Временные диаграммы опроса клавиатуры и индикации

Каждый разряд индикатора отображается в течение 5 мс с периодом 20 мс. Сканирование клавиатуры осуществляется импульсами длительностью 3 мкс (по столбцам), с паузами в 40 мкс.

Алгоритм программы представлен на рис. 5.4 и не требует особых комментариев; отметим только, что данное устройство достаточно эффективно использует возможности PIC-микроконтроллера 16СХХ. Действительно, большинство функциональных циклов выполняются менее чем за 5 мс, что согласуется с циклом работы таймера RTCC.

Внешний вид клавиатуры устройства представлен на рис. 5.5.

С помощью функциональных клавиш устанавливаются различные режимы работы устройства.

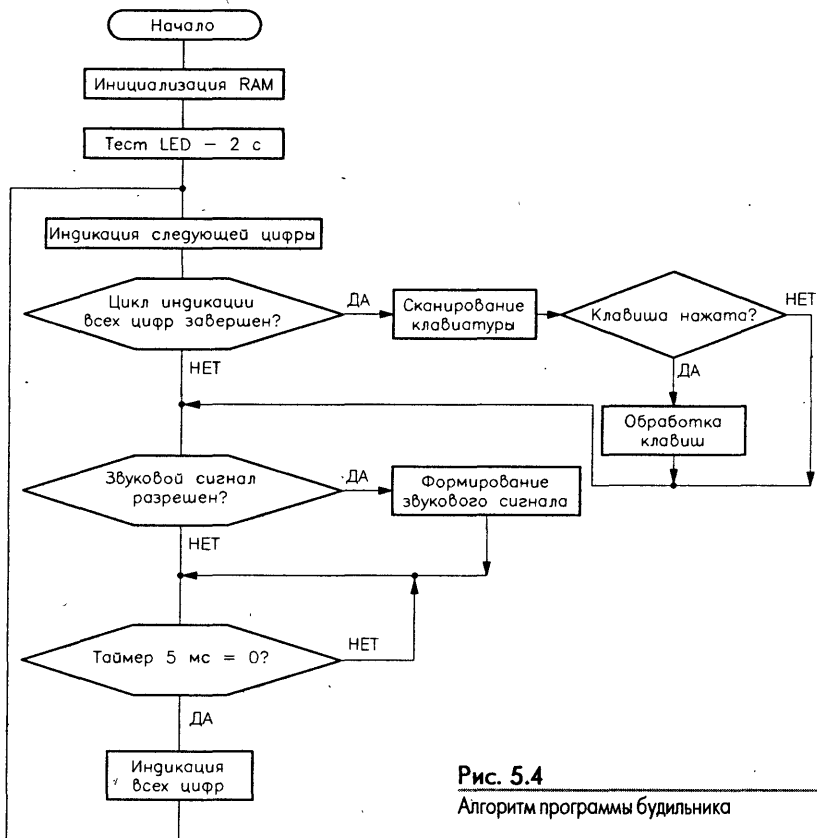


Рис. 5.4

Алгоритм программы будильника

1	2	3	DISABLE ALARM
4	5	6	AM/PM
7	8	9	CLEAR ALARM
ALARM TIME		CLEAR ENTRY	SET

Рис. 5.5

Внешний вид клавиатуры

Установка текущего времени

Нажать клавишу SET, затем с помощью цифровых клавиш последовательно установить десятки часов, единицы часов, десятки минут, единицы минут. Нажатием клавиш AM/PM и SET подтвердить выполненные установки.

Отображение времени будильника

Нажать клавишу ALARM. Время, когда прозвонит будильник, высвечивается в течение 5 с.

Программирование времени будильника

Нажать клавишу ALARM, затем SET, так чтобы светодиод *alarm* индикатора начал мигать. Потом последовательно набрать десятки часов, единицы часов, десятки минут, единицы минут. С помощью клавиш AM/PM и SET подтвердить выполненные установки.

Разрешение/отмена режима будильника

Нажать один или два раза клавишу ALARM. При каждом нажатии устройство переходит из одного режима в другой. Текущее состояние будильника указывается светодиодом с надписью *alarm*.

Временная приостановка звукового сигнала будильника

Нажать клавишу DISABLE ALARM. Звуковой сигнал будет прерван на десять минут. Эта функция предназначена для «ленивых», которые хотят понежиться в постели лишних десять минут!

Отмена звукового сигнала будильника

Нажать CLEAR ALARM. *Внимание!* Это действие окончательно, и звуковой сигнал повторится только на следующий день в тот же час.

Отмена набора

Нажать клавишу CLEAR ENTRY в режиме установки времени или установки времени будильника. Предшествующий нажатию клавиши CLEAR ENTRY набор будет аннулирован и восстановятся ранее заданные значения.

Программа, приведенная в листинге 5.1, реализует все вышеперечисленные функции.

Листинг 5.1

TITLE "Часы с будильником".

LIST P = 16C57

; ;
; Соответствует инструкции по применению AN529 фирмы Microchip.; ;
; Определение констант.

PIC57 EQU 7FFH

; ;
; *****
; Кварцевый генератор с частотой 4,096 МГц. Коэффициент деления предварительного
; делителя равен 32, что позволяет получить период приращения RTCC 31;25 мкс.
; Модуль счета RTCC равен 96, период - 5 мс.

MSEC5 EQU D'96

; *****
C EQU 0
BEP EQU 0
RTATS EQU 0
DC EQU 1
HR10 EQU 1
Z EQU 2
HR EQU 2
MIN10 EQU 3
MIN EQU 4
FLASH EQU 4
PA0 EQU 5
KEY_BEEP EQU 5
AMPM EQU 5
PA1 EQU 6
FD EQU 0
KEY_HIT EQU 6
ALED EQU 6
AM_PM EQU 7
COLON EQU 3
ALRMLED EQU 2
SERVICED EQU 7
ALONOF EQU 0
INAL EQU 1
SILNC EQU 2
INAA EQU 3
INKEYBEP EQU 5; ;
; Определение регистров RAM.

RTCC EQU 1

PC EQU 2

```

STATUS      EQU 3
FSR          EQU 4
PORT_A       EQU 5
PORT_B       EQU 6
PORT_C       EQU 7
;
; Определение регистров часов реального времени.
;
MSTMR        EQU 8      ; Счетчик миллисекунд.
STMR          EQU 9      ; Счетчик секунд.
MTMR          EQU 0A     ; Счетчик минут.
HTMR          EQU 0B     ; Счетчик часов.
;
; Определение регистров будильника.
MALARM        EQU 0C     ; Минуты будильника.
HALARM        EQU 0D     ; Часы будильника.
;
; Определение регистров ввода данных.
MENTRY        EQU 0E     ; Минуты.
HENTRY        EQU 0F     ; Часы.
;
; *****;
; Определение битов состояния и их функций.
;
FLAG          EQU 10
;
;   BIT # 7|6|5|4|3|2|1|0|
;   -----|---|---|---|---|---|---|
;
;       X|X|X|X|X|X|0|0| -> Режим часов (REAL TIME MODE - RTM).
;       X|X|X|X|X|X|0|1| -> Режим будильника (ALARM TIME MODE - ATM).
;       X|X|X|X|X|X|1|0| -> Режим ввода данных (DATA ENTRY MODE - DEM).
;       X|X|X|X|X|X|1|1| -> Режим проверки (TEST MODE - TM).
;       X|X|X|X|X|Y|X|X| -> Светодиод ALARM вкл./выкл.
;       X|X|X|X|Y|X|X|X| -> Двоеточие вкл./выкл.
;       X|X|X|Y|X|X|X|X| -> Мигание.
;       X|X|Y|X|X|X|X|X| -> Звуковой сигнал (KEY_BEEP).
;       X|Y|X|X|X|X|X|X| -> KEY_HIT (0/1).
;       Y|X|X|X|X|X|X|X| -> Служебный код.
;
;   X = произвольное значение.
;   Y = соответствует описанию (0/1).
;
TEMP          EQU 11
DIGIT          EQU 12
NEW_KEY        EQU 13
KEY_NIBL       EQU 14
DEBOUNCE       EQU 15
MIN_SEC        EQU 16      ; Счетчик минут/секунд.
ENTFLG         EQU 17
;
; Флаги режима ввода (регистр ENTFLG).
;
;   BIT # 7|6|5|4|3|2|1|0|
;   -----|---|---|---|---|---|---|
;
;       X|X|X|X|X|X|X|Y| -> Ввод времени/значений будильника.

```

```

; X|X|X|X|X|Y|X| -> Десятки часов установлены.
; X|X|X|X|X|Y|X|X| -> Единицы часов установлены.
; X|X|X|X|Y|X|X|X| -> Десятки минут установлены.
; X|X|X|Y|X|X|X|X| -> Единицы минут установлены.
; X|X|Y|X|X|X|X|X| -> INKEYBER.
; X|Y|X|Y|X|X|X|X| -> Не используется.
; Y|X|X|X|X|X|X|X| -> Не используется.

```

ALFLAG EQU 18 ; Регистр флагов будильника.

```

; BIT #7|6|5|4|3|2|1|0|
; X|X|X|X|X|X|X|Y| -> ALONOF. - - -
; X|X|X|X|X|X|Y|X| -> INAL. - - -
; X|X|X|X|X|Y|X|X| -> SILNC. - - -
; X|X|X|X|Y|X|X|X| -> INAA. - - -
; X|X|X|Y|X|X|X|X| -> Не используется.
; X|X|Y|X|X|X|X|X| -> Не используется.
; X|Y|X|Y|X|X|X|X| -> Не используется.
; Y|X|X|X|X|X|X|X| -> Не используется.

```

AAFLAG EQU 19

AATMR EQU 1A

Определение функций портов.

PORT_A:

```

; BIT 0 -> Выход на BEEPER (активный низкий).
; BIT 1-3 -> Не используются.

```

PORT_B: все выходы.

```

; BIT 0&4 -> Общий катод старшей цифры (MSB) & светодиод ALARM.
; BIT 1&5 -> Общий катод второй цифры & и двоеточие.
; BIT 2&6 -> Общий катод третьей цифры & и светодиод PM.
; BIT 3&7 -> Общий катод младшей цифры (LSB) & светодиод AM.

```

PORT_C:

При отображении все порты - выходы, управляющие сегментами.

При сканировании клавиатуры COL - выходы, ROW - входы.

```

; BIT 0 -> сегмент A & столбец (COL) 4.
; BIT 1 -> сегмент B & столбец (COL) 3.
; BIT 2 -> сегмент C & столбец (COL) 2.
; BIT 3 -> сегмент D & столбец (COL) 1.
; BIT 4 -> сегмент E & строка (ROW) 4.
; BIT 5 -> сегмент F & строка (ROW) 3.
; BIT 6 -> сегмент G & строка (ROW) 2.
; BIT 7 -> Общий анод всех сигнализаторов (ALARM, PM, AM, двоеточие)
; & строка (ROW) 1.

```

```

;
;   ORG      0
START
;   GOTO     INIT_CLK      ; Инициализация часов.
;
;
;   Тест индикации, все светодиоды зажжены на две секунды.
;
TEST_HARDWARE
;   MOVLW    D'02'        ; Горят в течение двух секунд.
;   MOVWF    MIN_SEC      ; /
;
;
TEST_LOOP
;   MOVF     MIN_SEC, W    ; Пересылаем в регистр W.
;   BTFSC    STATUS, Z     ; Если не равно 0, пропускаем,
;   GOTO     NORM_TIME     ; иначе переходим.
;   CALL     UPDATE_DISPLAY ; Обновить индикацию.
;   BSF      STATUS, PA0    ; Банк 1.
;   CALL     UPDATE_TIMERS  ; Ожидание и обновление.
;   BCF      STATUS, PA0    ; Банк 0.
;   GOTO     TEST_LOOP     ; Возвращаемся к началу цикла.
;
NORM_TIME
;   BCF      FLAG, 0        ; В реальном времени.
;   BCF      FLAG, 1
;
TIME_LOOP
;   CALL     UPDATE_DISPLAY
;   BSF      STATUS, PA1    ; Банк 2.
;   CALL     SERVICE_KEYS
;   BSF      STATUS, PA0    ; Банк 3.
;   CALL     SOUND_AA       ; Звуковой сигнал.
;   BCF      STATUS, PA1    ; Банк 1.
;   CALL     UPDATE_TIMERS  ; Ожидание и обновление таймеров.
;   BCF      STATUS, PA0    ; Банк 0.
;   BCF      STATUS, PA1    ; /
;   MOVF     FLAG, W        ; Флаги перемещаем в регистр W.
;   ANDLW    B'000000011'   ; /
;   XORLW    B'000000001'   ; /
;   BTFSC    STATUS, Z     ; Если не 0, то пропускаем.
;   CALL     RESET_ATM
;   GOTO     TIME_LOOP
;
;
RESET_ATM
;   MOVF     MIN_SEC, W     ; Загружаем MIN_SEC в регистр W.
;   ANDLW    B'00001111'   ; /
;   BTFSS    STATUS, Z     ; Если 0, пропускаем,
;   RETLW    0             ; в противном случае возвращаемся.

```

```
BCF    FLAG,0      ; Устанавливаем режим часов.
BCF    FLAG,ALRMLED ; Гасим СИД.
BTFSCL ALFLAG,ALONOF ; Проверяем ALONOF.
BSF    FLAG,ALRMLED ; Включаем светодиод ALARM.
RETLW  0           ; Возвращаемся.
```

UPDATE_DISPLAY

```
MOVLW  B'00000000' ; Обнуляем выходы сегментов.
MOVWF  PORT_C       ; /
MOVLW  B'00111111' ; Последняя цифра?
XORWF  PORT_B,0     ; /
BTFSCL STATUS,Z     ; Если не 0, то пропускаем.
GOTO   SCAN_KP      ; В противном случае сканируем клавиатуру.
```

UP_DSP_1

Выбор цифры индикации.

```
COMF   PORT_B,0     ; Загружаем содержимое порта В в регистр W с инверсией.
BTFSCL STATUS,Z     ; Если цифра не выбрана,
MOVLW  B'11000000'  ; тогда W = 11000000.
MOVWF  TEMP         ; Сохраняем в регистре TEMP.
COMF   TEMP         ; Инвертируем.
BSF    STATUS,C      ; Устанавливаем перенос.
RLF    TEMP         ; Сдвиг влево.
BTFSCL STATUS,C     ; Если C = 1, то пропускаем.
RLF    TEMP         ; В противном случае три раза сдвигаем влево
RLF    TEMP         ; с переносом.
MOVF   TEMP,0       ; Пересылаем в регистр W
MOVWF  PORT_B       ; и затем в порт В.
```

Формирование сигналов сегментов цифры.

```
MOVLW  MTMR         ; Загрузить FSR с MTMR.
MOVWF  FSR          ; /
MOVF   FLAG,0       ; Пересылка содержимого регистра FLAG в регистр W.
ANDLW  B'00000011'  ; Маскируем по шаблону.
MOVWF  TEMP         ; Сохраняем в регистре TEMP.
XORLW  B'00000011'  ; Инвертируем по шаблону.
BTFSCL STATUS,Z     ; Если не 0, то пропускаем,
GOTO   DO_TM        ; иначе переходим в режим тестирования.
BCF    STATUS,C      ; Обнуляем бит переноса.
RLF    TEMP         ; Левый сдвиг регистра TEMP.
MOVF   TEMP,0       ; Пересылка в регистр W.
ADDWF  FSR           ; Изменяем указатель косвенной адресации.
CALL   GET_7_SEG     ; Получаем семисегментный код.
```

```

MOVWF DIGIT      ; Пересылаем в DIGIT.
CALL MASK_ANNC   ; Вызываем MASK_ANNC.
BTFSC FLAG,FLASH ; Если флага FLASH нет, тогда пропускаем,
CALL CHK_HALF_SEC ; иначе вызываем CHK_HALF_SEC.
MOVF DIGIT,0      ; Пересылаем код символа
MOVWF PORT_C      ; в порт C.
RETLW 0           ; Возвращаемся.
;

```

DO_TM

```

MOVLW B'11111111' ; Включаем все сегменты.
MOVWF PORT_C       ; /
RETLW 0            ; Возвращаемся.
;

```

CHK_HALF_SEC

```

BTFSS FLAG, COLON ; Если двоеточие включено, то возвращаемся,
MOVLW B'00000000' ; Выдаем 0 на все линии порта C.
MOVWF DIGIT
RETLW 0
;

```

; Посредством косвенной адресации (с использованием FSR) формируем
; табличным способом семисегментные коды для отображения минут.

GET_7_SEG

```

COMF PORT_B,0      ; Инвертируем значение PORT_B и помещаем в W.
ANDLW B'11110000'  ; Маскируем младшую тетраду.
BTFSC STATUS,Z      ; Проверяем на 0.
INCF FSR            ; Если не 0, то инкрементируем FSR.
MOVF F0,0           ; Содержимое F0 помещаем в W,
MOVWF TEMP          ; а затем в TEMP.
COMF PORT_B,0      ; Инвертированное содержимое порта B пересылаем в W.
ANDLW B'11110000'  ; Маскируем младшую тетраду.
BTFSC STATUS,Z      ; Если не 0 (D1/2), то пропускаем,
BCF TEMP,AM_PM      ; иначе обнуляем бит AM/PM регистра TEMP.
COMF PORT_B,0      ; Инвертированное содержимое порта B пересылаем в W.
ANDLW B'11001100'  ; Маскируем, если D2 или D4.
BTFSC STATUS,Z      ; Если не 0 (D2 или D4), то пропускаем.
SWAPF TEMP          ; Иначе меняем местами тетрады регистра TEMP.
MOVLW B'00001111'  ; Маскируем старшую тетраду.
ANDWF TEMP,0
ADDWF PC            ; Результат добавляем к программному счетчику PC
; для адресации одной из строк таблицы.
RETLW B'00111111'  ; Семисегментный код цифры 0.
RETLW B'00000110'  ; Семисегментный код цифры 1.
RETLW B'01011011'  ; Семисегментный код цифры 2.

```



```

RETLW B'01001111' ; Семисегментный код цифры 3.
RETLW B'01100110' ; Семисегментный код цифры 4.
RETLW B'01101101' ; Семисегментный код цифры 5.
RETLW B'01111101' ; Семисегментный код цифры 6.
RETLW B'00000111' ; Семисегментный код цифры 7.
RETLW B'01111111' ; Семисегментный код цифры 8.
RETLW B'01100111' ; Семисегментный код цифры 9.

```

; Эта подпрограмма сканирует клавиатуру 4x4.

; Если клавиша нажата, KEY_HIT переходит в 1 и номер клавиши помещается в NEW_KEY.

; Если никакая клавиша не нажата, OFF помещается в NEW_KEY и KEY_HIT и переходит в 0.

SCAN_KP

```

BTFSC FLAG,KEY_HIT ; Клавиша "обслуживается"?
GOTO UP_DSP_1 ; Если да, пропускаем обычную процедуру,
MOVLW B'11111111' ; иначе выдаем 1 на все линии
MOVWF PORT_B ; порта В.
MOVLW B'11110111' ; Устанавливаем ключ COL на низкий уровень.
MOVWF TEMP ; Сохраняем в регистр TEMP.

```

SKP1

```

MOVLW B'00000000' ; Все линии порта С определяем как выходы.
TRIS PORT_C ; /
MOVF TEMP,W ; Пересылаем TEMP в регистр W.
ANDLW B'00001111' ; Опрашиваем клавиатуру.
MOVWF PORT_C ; /
MOVLW B'11110000' ; Старшие 4 линии порта С - входы,
TRIS PORT_C ; младшие - выходы.
MOVF TEMP,W ; Восстанавливаем предыдущее значение
; линий порта С.
MOVWF PORT_C ; /
MOVF PORT_C,W ; Считываем текущее значение в регистр W.
ANDLW B'11110000' ; Маскируем младшую тетраду.
XORLW B'11110000' ; Проверяем, была ли нажата клавиша,
BTFSS STATUS,Z ; Если нет - пропускаем,
GOTO DET_KEY ; да - определяем нажатую клавишу.

```

SKP3

```

BSF STATUS,C ; Установить бит переноса.
RRF TEMP ; Опросить следующий столбец.
BTFSC STATUS,C ; Если все выполнено, пропускаем.
GOTO SKP1
CLRF NEW_KEY ; Устанавливаем NEW_KEY = FF.
DECF NEW_KEY ; /

```

SKP2

```

CLRF PORT_C ; Обнуляем регистр PORT_C
MOVLW B'00000000' ; и задаем все линии порта С как выходы.
TRIS PORT_C ; /
GOTO UP_DSP_J ; Возвращаемся.

```

DET_KEY

; Нажатая клавиша обнаружена.

```

INCF    NEW_KEY,W      ; Клавиша отжата?
BTFS    STATUS,Z       ;
GOTO    SKP2           ; Если нет, то возвращаемся.
MOVF    PORT_C,W       ; Считываем номер строки клавиатуры.
IORLW   B'00001111'    ;
ANDWF   TEMP,W         ;
MOVWF   NEW_KEY        ; Переписываем в регистр W.
CALL    GET_KEY_VAL    ; Получаем значение клавиши.
MOVWF   NEW_KEY        ;
BSF     FLAG,KEY_HIT   ; Устанавливаем флажок "Клавиша нажата".
GOTO    SKP2           ; Возвращаемся.

```

; Эта подпрограмма декодирует код клавиши.

```

ONE     EQU    77
TWO     EQU    7B
THREE   EQU    7D
C       EQU    7E
FOUR    EQU    0B7
FIVE    EQU    0BB
SIX     EQU    0BD
D       EQU    0BE
SEVEN   EQU    0D7
EIGHT   EQU    0DB
NINE    EQU    0DD
E       EQU    0DE
A       EQU    0E7
ZERO    EQU    0EB
B       EQU    0ED
F       EQU    0EE

```

GET_KEY_VAL

```

ANDLW   B'00001111'    ; Маскируем старший полубайт.
MOVWF   KEY_NIBL       ; Сохраняем.
MOVLW   4               ; Задаем счет до 4.
MOVWF   TEMP           ; /

```

GKV1

```

BSF     STATUS,C       ; Устанавливаем бит переноса.
RRF     KEY_NIBL       ; Сдвигаем KEY_NIBL.
BTFS    STATUS,C       ; Проверяем перенос.
GOTO    GET_HI_KEY     ; Перходим.
DECF    TEMP           ; Уменьшаем значение счетчика.
GOTO    GK11..         ; Повторяем.

```

GO_RESET

```

BSF     STATUS, PA0      ; Банк 3.
BSF     STATUS, PA1      ; /
GOTO    SYS_RESET        ; Сброс.

GET_HI_KEY
DECf    TEMP              ; Декрементируем TEMP.
SWAPF   NEW_KEY, W        ; Меняем местами полубайты.
ANDLW   B'00001111'       ; /
MOVWF   KEY_NIBL          ; Сохраняем.
MOVF    TEMP, W           ; Определяем смещение для таблицы переходов.
ADDWF   PC                ; Модифицируем программный счетчик.
GOTO    GET147A           ; Переходим по таблице.
GOTO    GET2580           ; /
GOTO    GET369B           ; /
GOTO    GETCDEF           ; /

;
GET147A
MOVLW   4                  ; Задаем счет до 4.
GETCOM
MOVWF   TEMP
GETCOM1
BSF     STATUS, C          ; Устанавливаем бит переноса.
RRF     KEY_NIBL           ; Сдвигаем KEY_NIBL.
BTFSs   STATUS, C          ; Проверяем перенос.
GOTO    KEY_TBL           ; Переходим к таблице.
DECFSZ  TEMP              ; Декрементируем счетчик и проверяем на 0.
GOTO    GETCOM1           ; Если не 0, то повторяем,
GOTO    GO_RESET          ; иначе сбрасываем.

;
GET2580
MOVLW   8                  ; Задаем счет до 8.
GOTO    GETCOM

;
GET369B
MOVLW   D'12'             ; Задаем счет до 12.
GOTO    GETCOM

;
GETCDEF
MOVLW   D'16'             ; Задаем счет до 16.
GOTO    GETCOM

;
KEY_TBL
DECf    TEMP              ; Декрементируем TEMP.
MOVF    TEMP, W           ; Переписываем в W.
ADDWF   PC                ; Переходим по таблице.
RETLW   1                  ; Клавиша 1.
RETLW   4                  ; Клавиша 4.
RETLW   7                  ; Клавиша 7.
RETLW   0A                ; Клавиша A.

```

```
RETLW 2 ; Клавиша 2.
RETLW 5 ; Клавиша 5.
RETLW 8 ; Клавиша 8.
RETLW 0 ; Клавиша 0.
RETLW 3 ; Клавиша 3.
RETLW 6 ; Клавиша 6.
RETLW 9 ; Клавиша 9.
RETLW 0B ; Клавиша В.
RETLW 0C ; Клавиша С.
RETLW 0D ; Клавиша D.
RETLW 0E ; Клавиша E.
RETLW 0F ; Клавиша F.
```

MASK_ANNC

```
MOVLW B'11111100' ; Первая цифра?
XORWF PORT_B,0 ; /
BTFSC STATUS,Z ; Если нет, то пропускаем,
GOTO MASK_ALARM ; иначе переходим к MASK_ALARM.
MOVLW B'11110011' ; Вторая цифра?
XORWF PORT_B,0 ; /
BTFSC STATUS,Z ; Если нет, то пропускаем,
GOTO MASK_COLON ; иначе переходим к MASK_COLON.
MOVLW B'11001111' ; Третья цифра?
XORWF PORT_B,0 ; /
BTFSC STATUS,Z ; Если нет, то пропускаем,
GOTO MASK_PM ; иначе переходим к MASK_PM.
```

MASK_AM

```
INCF FSR ; Инкрементируем FSR.
BTFSS FO,AM_PM ; Проверяем разряд AM_PM, если он равен 0, значит - AM.
BSF DIGIT,7 ; Устанавливаем 7 разряд (MSB).
GOTO BLNK_LEAD_0 ; Продолжаем.
```

MASK_PM

```
INCF FSR ; Инкрементируем FSR.
BTFSS FO,AM_PM ; Проверяем разряд AM_PM, если он равен 1, значит - PM.
BSF DIGIT,7 ; Устанавливаем старший разряд (MSB).
GOTO BLNK_LEAD_0 ; Продолжаем.
```

MASK_ALARM

```
BTFSC FLAG,ALRMLED ; Если 1, то включаем светодиод (ALARM).
BSF DIGIT,7 ; /
GOTO BLNK_LEADJ ; Продолжаем.
```

MASK_COLON

```
BTFSC FLAG,COLON ; Если 1, то включаем двоеточие (COLON).
BSF DIGIT,7 ; /
GOTO BLNK_LEAD_0 ; Продолжаем.
```

BLNK_LEAD_0

```
MOVF FLAG,W ; Перемещаем флаги в регистр W.
```

```

ANDLW  B'00000011' ; Маскируем (IN DEM).
XORLW  B'00000010' ; Проверяем.
BTPSC  STATUS,Z     ; Если 0, то выполняем,
RETLW  0             ; иначе возвращаемся.
MOVLW  B'11111100'  ; Если первая цифра,
XORWF  PORT_B,0      ; /
BTFSS  STATUS,Z     ; то пропускаем.
RETLW  0             ; Возвращаемся,
MOVLW  B'00111111'  ; иначе маскируем G и ANUNC
ANDWF  DIGIT,0       ; и пересылаем в регистр W.
xorlw  B'00111111'  ; Проверяем на 0.
BTFSS  STATUS,Z     ; Если 0, то пропускаем.
RETLW  0             ; Возвращаемся,
MOVLW  B'10000000'  ; иначе выключаем D1.
ANDWF  DIGIT        ; /
RETLW  0             ; Возвращаемся.

;
; Инициализация портов RTCC.
IMIT_CLK
MOVLW  B'00001111'  ; Активизируем.
MOVWF  PORT_A        ; /
MOVLW  B'00000000'  ; Определяем все линии порта A как выходы.
TRIS   PORT_A

;

MOVLW  B'11111111'  ; Устанавливаем все разряды порта B в 1.
MOVWF  PORT_B        ; /
MOVLW  B'00000000'  ; Определяем все линии порта B как выходы.
TRIS   PORT_B

;

MOVLW  B'00000000'  ; Все разряды порта C устанавливаем в 0.
MOVWF  PORT_C        ; /
MOVLW  B'00000000'  ; Все линии порта C определяем как выходы.
TRIS   PORT_C        ; /

MOVLW  B'00000100'  ; Определяем коэффициент деления
MOVWF  OPTION        ; предварительного делителя частоты.

MOVLW  MSEC5         ; RTCC = 5 мс.
MOVWF  RTCC          ; /
CLRF   MSTMR         ; Обнуляем MSTMR (счетчик миллисекунд),
CLRF   STMR          ; STMR (секунд),
CLRF   MTMR          ; и MTMR (минут).
MOVLW  12H           ; Устанавливаем 12 часов.
MOVWF  HTMR          ; /
MOVWF  HALARM        ; Устанавливаем будильник на 12 часов.
CLRF   MALABM        ; /
MOVLW  B'00000011'  ; Устанавливаем режим тестирования.

```

```
MOVWF FLAG      ; /
CLRF ALFLAG     ; Сбрасываем все флаги.
CLRF AAFLAG     ; /
CLRF ENTFLG     ; /
GOTO TEST_HARDWARE
```

; Все подпрограммы, относящиеся к таймеру, помещены выше адреса 200.

```
ORG 0200
```

UPDATE_TIMERS

```
MOVF RTCC,W      ; RTCC перемещаем в регистр W.
BTFS STATUS,Z    ; Проверяем на 0.
GOTO UPDATE_TIMERS ; Если не 0 - повторяем,
MOVLW MSEC5      ; иначе RTCC = 5 мс.
MOVWF RTCC       ; /
INCF MSTMR       ; Инкрементируем счетчик миллисекунд.
BTFS FLAG,KEY_HIT ; Если клавиша не нажата, пропускаем,
GOTO CHK_DE_BOUNCE ; иначе переходим.
```

UP_TMR_1

```
MOVF FLAG,W      ; Режим будильника?
ANDLW B'000000U' ; /
XORLW B'000000U' ; /
BTFS STATUS,Z    ; Проверяем на 0, если да, то пропускаем
GOTO UP_TMR_2    ;
BSF FLAG,ALRMLED ; и включаем светодиод ALARM.
BSF FLAG,COLON   ; Включаем светодиоды двоеточия.
MOVLW D'100      ; Организуем мигание с периодом 1/2 с
SUBWF MSTMR,0    ; /
BTFS STATUS,C    ; /
BCF FLAG,ALRMLED ; светодиода ALARM.
GOTO UP_TMR_3    ; Переходим к UP_TMR_3.
```

UP_TMR_2

```
BSF FLAG,COLON   ; Включаем светодиоды двоеточия.
MOVLW D'100      ; Задаем режим мигания.
SUBWF MSTMR,0    ; /
BTFS STATUS,C    ; /
BCF FLAG,COLON   ; /
```

UP_TMR_3

```
MOVF MSTMR,0     ; Пересылаем MSTMR в W.
XORLW D'200      ; = 200 ?
BTFS STATUS,Z    ; /
RETLW 0          ;
```

; Приращение счетчика секунд.

```
CLRF MSTMR       ; Обнуляем MS_TMR.
MOVF MIN_SEC,W   ; Пересылаем MIN_SEC в регистр W.
```

```

ANDLW B'00001111' ; Маскируем минуты.
BTFSS STATUS,Z ; Проверяем на 0. Если 0, то пропускаем.
DECF MIN_SEC ; Декрементируем счетчик секунд.
MOVLW STMR ; Загружаем S_TMR в регистр косвенного адреса.
MOVWF FSR ; /
CALL INC_60 ; Вызываем INC_60 (инкрементируем секунды).
IORLW 0 ; Проверяем на 0.
BTFSS STATUS,Z ; Если 0, то пропускаем,
GOTO CHK_AL_TIM ; иначе - продолжаем.

```

```

;
; Приращение счетчика минут.
;

```

```

SWAPF MIN_SEC ; Меняем местами тетрады регистра MIN_SEC.
MOVF MIN_SEC,W ; Перемещаем в регистр W.
ANDLW B'00001111' ; Маскируем секунды.
BTFSS STATUS,Z ; Если не 0, пропускаем,
DECF MIN_SEC ; иначе декрементируем счетчик минут.
SWAPF MIN_SEC ; Снова меняем местами тетрады.
CALL CHK_SILNC_TIM ; Переходим к подпрограмме.
MOVLW MTMR ; Инкрементируем счетчик минут.
MOVWF FSR ; /
CALL INC_60 ; /
IORLW 0 ; Проверяем на 0.
BTFSS STATUS,Z ; Если 0, пропускаем.
GOTO CHK_AL_TIM ; Проверяем время индикации.

```

```

;
; Приращение счетчика часов.
;

```

```

MOVLW HTMR ; Загружаем адрес счетчика часов
MOVWF FSR ; в регистр косвенного адреса.
CALL INC_HR ; Инкрементируем счетчик часов.

```

```

;
CHK_AL_TIM
BTFSS ALFLAG,ALONOF ; Проверяем флаг ALONOF, если 0 - выходим.
RETLW 0 ; /
BTFSC ALFLAG,SILNC ; Если изменений нет, то возвращаемся.
RETLW 0
BTFSC ALFLAG,INAL ; Выполнено?
GOTO CHK_1_MIN ; Переходим к CHK_1_MIN.
RETLW 0 ; Если да, то возвращаемся.
MOVF HALARM,W ; Сравниваем время будильника (часов)
XORWF HTMR,W ; с реальным временем.
BTFSS STATUS,Z ; Если равны, то переходим к сравнению минут,
RETLW 0 ; иначе возвращаемся.
MOVF MALARM,W ; Сравниваем минуты.
XORWF MTMR,W ; /
BTFSS STATUS,Z ; /
RETLW 0 ; /

```

```

MOVF    STMR,W      ; Счетчик секунд равен нулю?
BTFSS   STATUS,Z     ; Если да, то пропускаем команду.
RETLW   0            ; Если нет, возвращаемся.
BSF     ALFLAG,INAL  ; Устанавливаем сигнальный флаг будильника.
MOVLW   10           ; Устанавливаем таймер на интервал в 1 минуту.
MOVWF   MIN_SEC      ; /
RETLW   0

```

;
CHK_1_MIN

```

SWAPF   MIN_SEC,W    ; Меняем местами тетрады минут и секунд.
ANDLW   B'00001111'  ; Проверяем минуты.
BTFSS   STATUS,Z     ; Если 0, то пропускаем команду.
RETLW   0            ; Возвращаемся.
BCF     ALFLAG,INAL  ; Обнуляем бит INAL.
BCF     ALFLAG,INAA  ; Обнуляем бит INAA.
BSF     PORT_A,BEP   ; Прекращаем звуковой сигнал.
RETLW   0

```

;
INC_60

```

INCF    F0           ; Инкрементируем регистр F0.
MOVF    F0,0         ; Пересылаем в регистр W.
ANDLW   B'00001111'  ; Маскируем старшие биты.
XORLW   B'00001010'  ; Если значение равно 10, то обнуляем,
BTFSS   STATUS,Z     ; /
RETLW   1            ; иначе возвращаем 1.
MOVLW   B'11110000'  ; Маскируем младшие разряды.
ANDWF   F0           ; /
SWAPF   F0           ; Меняем местами тетрады регистра F0.
INCF    F0           ; Инкрементируем.
MOVF    F0,0         ; Пересылаем в регистр W.
SWAPF   F0           ; Обратно меняем тетрады.
XORLW   D'6         ; Сравниваем с числом 6.
BTSS    STATUS,Z     ; Если 0, то пропускаем команду,
RETLW   1            ; иначе возвращаем число, не равное 0 (NZ).
CLRF    F0           ; Обнуляем F0.
RETLW   0            ; Возвращаем 0.

```

;
CHK_SILNC_TIM

```

BTFSS   ALFLAG,SILNC ; Проверяем, были ли изменения.
RETLW   0            ; Если нет, пропускаем.
SWAPF   MIN_SEC,W    ; Вводим MIN_SEC в регистр W с перестановкой тетрад.
ANDLW   B'00001111'  ; Маскируем секунды.
BTFSS   STATUS,Z     ; Если да,
RETLW   0            ; возвращаемся.
BCF     ALFLAG,SILNC ; Обнуляем бит SILNC.
MOVLW   10           ; Устанавливаем таймер на 1 мин.
MOVWF   MIN_SEC      ; /
RETLW   0

```


CHK_DE_BOUNCE

```

    BTFSC  ENTFLG, INKEYBEP ; Звуковой сигнал включен?
    GOTO   CHK_DEB_1       ; Если да, то декрементируем таймер.
    BTFSS  FLAG, KEY_BEEP  ; Флаг звукового сигнала установлен?
    GOTO   CHK_SERV        ; Если нет, то проверяем, обслуживается ли он.
    BTFSC  ALFLAG, INAA    ; Проверяем, установлен ли флаг INAA.
    GOTO   CHK_BEP_ON      ; Если да, проверяем, включен ли звук.

```

CHK_DEB_1

```

    BSF    ENTFLG, INKEYBEP ; Устанавливаем флаг.
    MOVF   DEBOUNCE, W      ; Пересылка в регистр W.
    BTFSC  STATUS, Z        ; Если не 0, пропускаем,
    MOVLW  D'20             ; иначе задаем 100 мс.
    MOVWF  DEBOUNCE         ; /
    BCF    PORT_A, BEP      ; Включаем звуковой сигнал.
    DECFSZ DEBOUNCE         ; Декрементируем, пока не 0.
    GOTO   UP_TMR_1         ; Возвращаемся.
    BSF    PORT_A, BEP      ; Отключаем таймер.

```

CHK_SERV

```

    CLRF   DEBOUNCE
    BSF    PORT_A, BEP
    BTFSS  FLAG, SERVICED   ; Если обслуживается, то попускаем команду.
    GOTO   UP_TMR_1         ; Возвращаемся.
    BCF    FLAG, SERVICED   ; Обнуляем флажки.
    BCF    FLAG, KEY_HIT    ; /
    BCF    FLAG, KEY_BEEP   ; /
    BCF    ENTFLG, INKEYBEP ; /
    GOTO   UP_TMR_1         ; Возвращаемся.

```

CHK_BEP_ON

```

    BTFSS  PORT_A, BEP      ; Проверяем бит BEP.
    GOTO   UP_TMR_1         ; Если 0, то ожидаем.
    GOTO   CHK_DEB_1        ; Возвращаемся.

```

INC_HR

```

    INCF   FO               ; Инкрементируем счетчик часов.
    MOVF   FO, W            ; Пересылаем в регистр W.
    MOVWF  TEMP             ; Сохраняем в TEMP.
    ANDLW  B'00001111'     ; Маскируем старшую тетраду
    XORLW  D'10             ; и сравниваем со значением 10.
    BTFSS  STATUS, Z        ; Если равно, то пропускаем команду,
    COTO   INC_AM_PM        ; иначе переходим к INC_AM_PM.
    MOVLW  B'00010000'     ; Загружаем 1 в старшую тетраду.
    MOVWF  FO
    GOTO   RESTORE_AM_PM    ; Восстанавливаем AM/PM.

```

INC_AM_PM

```

    BCF    FO, AM_PM        ; Обнуляем бит AM/PM.
    MOVF   FO, W            ; Пересылаем в W.

```

```

XORLW 12H ; Сравниваем со значением 12 HEX.
BTFSS STATUS,Z ; Если равно, то пропускаем следующую команду.
GOTO CHK_13 ; Иначе переходим к CHK_13.
BTFSS TEMP,AM_PM ; Проверяем флаг AM_PM,
GOTO SET_AM_PM ; если он не установлен, устанавливаем.
BCF FO,AM_PM ; Сбрасываем флаг.
RETLW 0 ; Возвращаемся.
SET_AM_PM
BSF FO,AM_PM ; Устанавливаем флаг AM_PM.
CHK_13
MOVF FO,W ; Пересылаем в регистр W.
XORLW 13H ; Сравниваем с числом 13H.
BTFSS STATUS,Z ; При равенстве пропускаем следующую команду.
GOTO RESTORE_AM_PM
SET_1_HR
MOVLW B'00000001' ; Загружаем 1.
MOVWF FO
RESTORE_AM_PM
BTFSC TEMP,AM_PM ; Пропускаем команду, если AM.
BSF FO,AM_PM ; Устанавливаем флаг AM_PM.
RETLW 0
;
ORG 400
;
; Определение функций клавиш.
;
ALARM_KEY EQU 0A
CE_KEY EQU 0B
SNOOZE_KEY EQU 0C
AM_PM_KEY EQU 0D
CLR_ALARM_KEY EQU 0E
SET_KEY EQU 0F
;
SERVICE_KEYS
BTFSS FLAG,KEY_HIT ; Если нет нажатой клавиши,
RETLW 0 ; возвращаемся.
BTFSC FLAG,SERVICED ; Если обрабатывается,
RETLW 0 ; возвращаемся.
BSF FLAG,SERVICED ; Установить флаг обработки.
MOVF FLAG,W ; Определяем режим.
ANDLW B'00000011' ; /
BTFSC STATUS,Z ; Если 00, тогда режим часов.
GOTO RTMKS ; Переводим клавиатуру в режим часов.
MOVWF TEMP ; Сохраняем в регистре TEMP.
DECFSZ TEMP ; Декрементируем регистр TEMP.
GOTO SK1 ; Если 0, пропускаем.
GOTO ATMKS ; Если 01, задаем режим будильника.

```

```

DECFSZ TEMP      ; Декрементируем TEMP.
RETLW 0           ; Если 11, возвращаемся.
GOTO DEMKS       ; Если 10, переходим в режим ввода данных.

```

```

; Обработка клавиш в реальном времени.

```

```

RTMKS

```

```

CALL CHK_AL_KEYS ; Проверяем все клавиши.
IORLW 0           ; Сравниваем с 0.
BTFSC STATUS,Z   ; Если 0,
RETLW 0           ; то возвращаем 0,
MOVLW SET_KEY    ; иначе загружаем SET_KEY.
XORWF NEW_KEY,W  ; Сравниваем с NEW_KEY,
BTFSC STATUS,Z   ; если не 0, то пропускаем.
GOTO SERV_SET_RTC ; Переходим к обработке клавиш режима часов.
MOVLW ALARM_KEY  ; Проверяем клавиши будильника?
XORWF NEW_KEY,W  ; /
BTFSC STATUS,Z   ; Если не 0, пропускаем,
GOTO SERV_ALARM_RTC ; иначе обрабатываем сигнал будильника.

```

```

IGNORE_KEY

```

```

RETLW 0           ; Возвращаемся.

```

```

; Обработка клавиш будильника.

```

```

ATMKS

```

```

CALL CHK_AL_KEYS ; Проверяем все клавиши.
IORLW 0           ; Сравниваем с 0.
BTFSC STATUS,Z   ; Если не 0,
RETLW 0           ; то пропускаем.
MOVLW SET_KEY    ; Клавиша нажата?
XORWF NEW_KEY,W  ; /
BTFSC STATUS,Z   ; Если не 0, то пропускаем.
GOTO SERV_SET_ATM
MOVLW ALARM_KEY  ; Проверяем клавишу ALARM.
XORWF NEW_KEY,W  ; Нажата?
BTFSC STATUS,Z   ; Если нет, пропустить,
GOTO SERV_ALARM_ATM ; иначе обработать.
GOTO IGNORE_KEY

```

```

; Обработка клавиш ввода данных.

```

```

DEMKS

```

```

CALL CHK_AL_KEYS ; Проверяем все клавиши.
IORLW 0           ; Сравниваем с 0.
BTFSC STATUS,Z   ; Если не 0, пропускаем,
RETLW 0           ; иначе возвращаемся.
MOVLW SET_KEY    ; Проверяем SET_KEY.
XORWF NEW_KEY,W  ; /
BTFSC STATUS,Z   ; Если не 0, пропускаем,
GOTO DEMKS_END   ; иначе заканчиваем.

```

```

MOVLW    CE_KEY      ; Если нажат ключ отмены набора,
XORWF    NEW_KEY, W  ; /
BTFSF    STATUS, Z   ; /
GOTO     DEMKS_END_1 ; заканчиваем ввод.
BTFSF    ENTFLG, HR10 ; Десятки часов установлены?
GOTO     ENT_HR_10    ; Если нет, ввести.
BTFSF    ENTFLG, HR   ; Единицы часов введены?
GOTO     ENT_HRS      ; Если нет, ввести.
BTFSF    ENTFLG, MIN10 ; Десятки минут введены?
GOTO     ENT_MIN_10   ; Если нет, ввести.
BTFSF    ENTFLG, MIN  ; Единицы минут введены?
GOTO     ENT_MIN      ; Если нет, ввести.
GOTO     ENT_AM_PM    ; Выполнить.

DEMK5_END
BTFSF    ENTFLG, RTATS ; Проверить бит RTATS.
GOTO     LD_RTM        ; Перейти к LD_RTM.
MOVF     MENTRY, W     ;
MOVWF    MALARM        ;
MOVF     HENTRY, W     ;
MOVWF    HALARM        ;
BCF      FLAG, ALRMLED ; Сбросить флаг ALRMLED.
BTFSF    ALFLAG, ALONOF ; Проверить флаг ALONOF (вкл./выкл.).
BSF      FLAG, ALRMLED ; Если установлен, то установить ALRMLED.

DEMK5_END_1
BCF      FLAG, 0        ; Устанавливаем режим часов.
BCF      FLAG, 1        ; /
BCF      FLAG, FLASH    ;

SERV_COM_RET
BSF      FLAG, KEY_BEEP
RETLW    0              ; Возвращаемся.

;
LD_RTM
MOVF     MENTRY, W     ; Загружаем время.
MOVWF    MTMR          ; /
MOVF     HENTRY, W     ; /
MOVWF    HTMR          ; /
CLRF     MSTMR         ; Обнуляем счетчик секунд
CLRF     STMR          ; /
GOTO     DEMK5_END_1   ; Возвращаемся.

;
ENT_HR_10
MOVF     NEW_KEY, W     ; NEW_KEY перемещаем в регистр W.
BTFSF    STATUS, Z     ; Если не 0, пропускаем.
GOTO     LD_HENTRY_0    ; Загружаем 0.
DECFSZ   NEW_KEY, 0     ; Если 1, пропускаем.
GOTO     IGNORE_KEY     ; Игнорируем нажатие клавиши.
BSF      HENTRY, 4      ; Установить в 1.
BSF      ENTFLG, HR10   ; Устанавливаем флаг HR10.

```

```

        GOTO    SERV_COM_RET    ; Переходим к следующему.
LD_HENTRY_0
        BCF     HENTRY, 4        ; Обнулить.
        BSF     ENTFLG, HR10
        GOTO    SERV_COM_RET    ;
ENT_HRS
        MOVLW   HENTRY           ; Используем косвенную адресацию.
        MOVWF   FSR              ; /
        BTFSC   HENTRY, 4        ; Проверяем бит на 0.
        GOTO    ALLOWO_2        ; Если 1, то устанавливаем десятки часов.
        MOVLW   D'10'           ; Проверяем от 0-9.
        SUBWF   NEW_KEY, W       ; /
        BTFSC   STATUS, C        ; Если переноса нет, пропускаем.
        GOTO    IGNORE_KEY      ; Игнорируем нажатие клавиши.
ENT_LO_COM1
        BSF     ENTFLG, HR       ; Устанавливаем флаг HR.
ENT_LO_COM
        MOVF    FO, W            ; Загружаем HRS.
        ANDLW   B'11110000'     ; Маскируем младшую тетраду.
        IORWF   NEW_KEY, W       ; Сравниваем с NEW_KEY.
        MOVWF   FO              ; Сохраняем.
        GOTO    SERV_COM_RET    ; Переходим к следующему.
ALLOWO_2
        MOVLW   D'3'            ; Разрешаем значения в интервале 0-2.
        SUBWF   NEW_KEY, W       ; /
        BTFSC   STATUS, C        ; Если меньше, чем 3, пропускаем.
        GOTO    IGNORE_KEY
        GOTO    ENT_LO_COM1     ; /
ENT_MIN_10
        MOVLW   MENTRY          ; Косвенная адресация.
        MOVWF   FSR              ; /
        MOVLW   D'6'            ; Разрешаем значения 0-5.
        SUBWF   NEW_KEY, W       ; /
        BTFSC   STATUS, C        ; Если перенос, пропускаем.
        GOTO    IGNORE_KEY      ; Игнорируем нажатие клавиши.
        SWAPF   FO, W            ; Меняем местами тетрады.
        ANDLW   B'11110000'     ; Маскируем младшую тетраду.
        IORWF   NEW_KEY, W       ; Проверяем нажатие.
        MOVWF   FO              ; Сохраняем.
        SWAPF   FO              ; Меняем тетрады.
        BSF     ENTFLG, MIN10
        GOTO    SERV_COM_RET    ; Продолжаем.
;
ENT_MIN
        MOVLW   MENTRY          ; Косвенная адресация.
        MOVWF   FSR              ; /
        MOVLW   D'10'           ; Разрешаем значения 0-9.

```

```

SUBWF NEW_KEY,W ; Проверяем выход за эти пределы.
BTFSC STATUS,C ; Если не вышли, пропускаем.
GOTO IGNORE_KEY ; Иначе игнорируем нажатие.
BSF ENTFLG,MIN ; Устанавливаем флаг.
GOTO ENT_LO_COM ; /

;
ENT_AM_PM
MOVLW AM_PM_KEY ; Нажата клавиша AM/PM?
XORWF NEW_KEY,W ; /
BTFSS STATUS,Z ; Если да, пропускаем.
GOTO IGNORE_KEY
BTFSS HENTRY,AM_PM ; Проверяем флаг AM_PM.
GOTO SETAMPM ; Если не установлен, установить.
BCF HENTRY,AM_PM ; Сбросить флаг.
GOTO SERV_COM_RET ; Завершить.

SETAMPM
BSF HENTRY,AM_PM ; Установить флаг.
GOTO SERV_COM_RET

;
SERV_SET_RTM
MOVF MTMR,W ; Пересылаем значения.
MOVWF MENTRY ; /
MOVF HTMR,W ; /
MOVWF HENTRY ; /

SERV_COM
MOVF FLAG,W ; Сохраняем в регистре W.
ANDLW B'00000001' ; Режим будильника или часов?
MOVWF ENTFLG ; Сохраняем в ENTFLG.
MOVLW B'11110010' ; Задаем 1 с.
IORWF FLAG ; /
BCF FLAG,0 ; /
RETLW 0

;
SERV_SET_ATM
MOVF MALARM,W ; Пересылаем значения.
MOVWF MENTRY ; /
MOVF HALARM,W ; /
MOVWF HENTRY ; /
BSF ALFLAG,ALONOF ; Устанавливаем флаг.
GOTO SERV_COM ; Переходим к SERV_COM.

SERV_ALARM_ATM
BTFSS ALFLAG,ALONOF ; Проверяем флаг вкл./выкл.
GOTO SET_ALONOF ; Устанавливаем флаг вкл./выкл.
BCF ALFLAG,ALONOF ; Сбрасываем флаг.
GOTO SERV_ATM_COM ; Переходим к SERV_ATM_COM.

SET_ALONOF
BSF ALFLAG,ALONOF ; Устанавливаем флаг.

SERV_ATM_COM

```

```

BSF     FLAG, KEY_BEEP    ; Звуковой сигнал.
MOVLW   B'11110000'      ; Сбрасываем счетчик секунд.
ANDWF   MIN_SEC           ; /
RETLW   0                 ; Возвращаемся.
;
SERV_ALARM_RTM
BSF     FLAG, KEY_BEEP    ; Устанавливаем флаг звукового сигнала.
BSF     FLAG, 0           ;
BCF     FLAG, 1           ; /
MOVLW   D'05'            ; Записываем 5 в регистр MIN_SEC.
MOVWF   MIN_SEC           ; /
RETLW   0                 ;
;
SERV_SNOOZE
MOVLW   0A0              ; Устанавливаем значение 10.
MOVWF   MIN_SEC           ; /
BSF     ALFLAG, SILNC     ; Устанавливаем флаг.
CLR_AL_COM
BSF     FLAG, KEY_BEEP    ; Устанавливаем флаг звукового сигнала.
CLRF    AATMR             ; Обнуляем таймер AATMR.
CLRF    AAFLAG            ; Сбрасываем флаг.
BCF     ALFLAG, INAA      ; Сбрасываем флаг INAA.
BSF     PORT_A, BEP       ; Отключаем звуковой сигнал.
RETLW   0                 ; Возвращаемся.
;
CHK_AL_KEYS
BTFS    ALFLAG, ALONOF    ; Будильник включен?
RETLW   1                 ; Если нет, то возвращаемся.
BTFS    ALFLAG, INAL      ; Проверяем флаг INAL.
RETLW   1                 ; Если нет, пропускаем.
MOVLW   CLR_ALARM_KEY     ; Сбрасываем время будильника?
XORWF   NEW_KEY, W        ; /
BTFS    STATUS, Z         ; Если нет, пропускаем.
GOTO    CLR_ALARM         ; Сбрасываем время будильника.
MOVLW   SNOOZE_KEY        ; Проверяем SNOOZE_KEY.
XORWF   NEW_KEY, W        ; /
BTFS    STATUS, Z         ; Если 0, пропускаем.
RETLW   1                 ;
GOTO    SERV_SNOOZE
;
CLR_ALARM
BCF     ALFLAG, INAL      ; Сбросить INAL.
BCF     ALFLAG, SILNC     ; Сбросить SILNC.
MOVLW   B'00001111'      ; Обнулить минуты.
ANDWF   MIN_SEC           ; /
GOTO    CLR_AL_COM
;
ORG     600

```

; Если сигнал разрешен, эта подпрограмма
; берет на себя управление генерацией звука.

SOUND_AA

```

BTFS    ALFLAG, INAL    ; Пропускаем, если включен сигнал тревоги.
RETLW   0                ; Возвращаемся.
BTFS    ALFLAG, SILNC   ; Проверяем SILNC.
RETLW   0                ; Возвращаемся.
BTFS    ENTFLG, INKEYBER; Проверяем INKEYBER.
GOTO    CHK_COLSN       ; Если установлен, переходим к CHK_COLSN.

```

SND_AA0

```

BTFS    ALFLAG, INAA    ; Пропускаем, если INAA = 1.

```

SND_AA_1

```

CALL    INIT_AA         ; Инициализируем все.
BTFS    AAFLAG, 0        ; Пропускаем, если выполнено.
GOTO    DO_CYCLO        ; Иначе выполняем цикл 1.
BTFS    AAFLAG, 1        ; Пропускаем, если выполнено.
GOTO    DO_CYCL1        ; Иначе выполняем цикл 2.
BTFS    AAFLAG, 2        ; Пропускаем, если выполнено.
GOTO    DO_CYCL2        ; Иначе выполняем цикл 3.
BTFS    AAFLAG, 3        ; Пропускаем, если выполнено.
GOTO    DO_CYCL3        ; Иначе выполняем цикл 4.
BTFS    AAFLAG, 4        ; Пропускаем, если выполнено.
GOTO    DO_CYCL4        ; Иначе выполняем цикл 5.
BTFS    AAFLAG, 5        ; Пропускаем, если выполнено.
GOTO    DO_CYCL5        ; Иначе выполняем цикл 6.
BTFS    AAFLAG, 6        ; Пропускаем, если выполнено.
GOTO    DO_CYCL6        ; Иначе выполняем цикл 6.
BTFS    AAFLAG, 7        ; Пропускаем, если выполнено.
GOTO    DO_CYCL7        ; Иначе выполняем цикл 7.
GOTO    SND_AA_1         ; Возвращаемся.

```

INIT_AA

```

CLRF    AAFLAG          ; Сбрасываем все флаги.
BSF     ALFLAG, INAA    ; Устанавливаем флаг INAA.
GOTO    PUT_ON_100      ; Задаем 100 мс.

```

DEC_AA_TMR

```

DEC     AATMR            ; Декрементируем таймер.
MOVF    AATMR, W         ; Пересылаем в регистр W.
BTFS    STATUS, Z        ; Проверяем на 0.
RETLW   1                ; Если не 0, возвращаем 1,
RETLW   0                ; иначе возвращаем 0.

```

DO_CYCLO

```

CALL    DEC_AA_TMR       ; Уменьшаем значение таймера.
BTFS    STATUS, Z        ; Если не 0, то возвращаемся.
RETLW   0                ;
BSF     AAFLAG, 0        ; Устанавливаем флаг "ГОТОВО".

```



```

PUT_OFF_100
    BSF     PORT_A, BEP      ; Выключаем звуковой сигнал
    MOVLW   D'20'           ; на 100 мс.
    MOVWF   AATMR           ; /
    RETLW   0

;
DO_CYCL1
    CALL    DEC_AA_TMR      ; Уменьшить значение таймера.
    BTFSS   STATUS, Z       ; Проверяем на 0.
    RETLW   0
    BSF     AAFLAG, 1       ; Устанавливаем флаг "ГОТОВО".

PUT_ON_100
    BCF     PORT_A, BEP     ; Отключаем звуковой сигнал
    MOVLW   D'20'           ; на 100 мс.
    MOVWF   AATMR           ; /
    RETLW   0

DO_CYCL2
    CALL    DEC_AA_TMR      ; Декрементируем таймер.
    BTFSS   STATUS, Z       ; Если не 0, то возвращаемся.
    RETLW   0               ; /
    BSF     AAFLAG, 2       ; Устанавливаем флаг "ГОТОВО".
    BSF     PORT_A, BEP     ; Отключаем звуковой сигнал
    MOVLW   D'100'          ; на 500 мс.
    MOVWF   AATMR           ; /
    RETLW   0

;
DO_CYCL3
    CALL    DEC_AA_TMR      ; Декрементируем таймер.
    BTFSS   STATUS, Z       ; Если не 0, то возвращаемся.
    RETLW   0               ; /
    BSF     AAPLAG, 3       ; Устанавливаем флаг "ГОТОВО".
    GOTO    PUT_ON_100      ; Выполняем следующий цикл.

;
DO_CYCL4
    CALL    DEC_AA_TMR      ; Декрементируем таймер.
    BTFSS   STATUS, S       ; Если не 0,
    RETLW   0               ; возвращаемся.
    BSF     AAFLAG, 4       ; Устанавливаем флаг "ГОТОВО".
    GOTO    PUT_OFF_100     ; Выполняем следующий цикл.

;
DO_CYCL5
    CALL    DEC_AA_TMR      ; Декрементируем таймер.
    BTFSS   STATUS, Z       ; Если не 0, то возвращаемся.
    RETLW   0               ; /
    BSF     AAFLAG, 5       ; Устанавливаем флаг "ГОТОВО".
    GOTO    PUT_ON_100      ; Выполняем следующий цикл.

;
DO_CYCL6

```

```

CALL    DEC_AA_TMR      ; Декрементируем таймер.
BTFSS   STATUS, Z       ; Если не 0, то возвращаемся.
RETLW   0               ; /
BSF     AAFLAG, 6       ; Устанавливаем флаг "ГОТОВО".
BSF     PORT_A, BEP     ; Отключаем звуковой сигнал
MOVLW   D'200           ; на 1000 мс.
MOVWF   AATMR           ; /
RETLW   0
;
DO_CYCL7
CALL    DEC_AA_TMR      ; Декрементируем таймер.
BTFSS   STATUS, Z       ; Возвращаемся, если не 0.
RETLW   0               ; /
BSF     AAFLAG, 7       ; Устанавливаем флаг "ГОТОВО".
GOTO    PUT_ON, 100     ; Выполняем следующий цикл.
;
CHK_COLSN
BTFSC   PORT_A, BEP     ; Если включен, то пропускаем,
GOTO    SND_AA_0        ; иначе возвращаемся.
MOVF    AATMR, W        ; Пересылаем значение AATMR в регистр W.
BTFSC   STATUS, Z       ; Проверяем на 0.
GOTO    LD_AAT_1        ; Загружаем в таймер.
DEC     AATMR           ; Декрементируем таймер.
RETLW   0               ; Возвращаемся.
LD_AAT_1
INCF    AATMR           ; Инкрементируем таймер.
RETLW   0               ; Возвращаемся.
;
ORG     PIC57
SYS_RESET
GOTO    START
;
END

```

РЕАЛИЗАЦИЯ ШИНЫ I²C

Этот пример был выбран по двум причинам. Во-первых, популярность шины I²C постоянно возрастает. Во-вторых, в данном случае рассматриваемый интерфейс реализуется микроконтроллерами 16C5X, лишенными встроенного последовательного интерфейса, и используются линии стандартного параллельного порта.

Название шины I²C представляет собой аббревиатуру от сочетания Inter Integrated Circuit (ИИ, или I²C). Эта стандартная последовательная шина была изначально предложена фирмой Philips, но сейчас микросхемы с ней выпускаются очень многими производителями.

Шина I²C использует только две сигнальные линии и обеспечивает обмен данными в последовательной форме со скоростью 100 Кбит/с (или 400 Кбит/с для самих последних версий). Это значение, конечно, далеко от скоростей передачи локальных информационных сетей, в которых скорость доходит до 10 Мбит/с и более.

Первоначально шина I²C применялась для звуковых или телевидеоприложений, для которых простота реализации была важнее, чем высокая пропускная способность.

Общие положения

Шина I²C имеет следующие основные характеристики:

- ◆ это последовательная двухпроводная шина с линией передачи данных SDA (Serial Data) и линией передачи тактовых синхроимпульсов SCL (Serial Clock);
- ◆ данные могут передаваться в обоих направлениях между любыми двумя абонентами;
- ◆ предусмотрено мультиуправление передачей данных, то есть инициировать обмен и управлять им может любое подключенное устройство;
- ◆ каждый абонент шины имеет свой персональный семиразрядный служебный адрес. Таким образом, к шине может быть подключено до 128 абонентов;
- ◆ пересылка каждого байта данных завершается сигналом подтверждения приема;
- ◆ шина может работать с максимальной скоростью 100 Кбит/с, причем ее можно автоматически замедлять, чтобы приспособиться к скорости более медленных устройств-абонентов;
- ◆ максимальное число абонентов шины ограничено допустимой емкостной нагрузкой шины, которая не должна превышать 400 пф. Реальное число зависит от характеристик микросхем, используемых для подключения к шине, и дискретных полупроводниковых компонентов;
- ◆ электрические уровни шины допускают использование микросхем, изготовленных по технологиям КМОП, НМОП или ТТЛ.

Рассмотрим принципы функционирования шины.

Принцип обмена данными

Принцип работы входных и выходных каскадов устройств, подключаемых к шине I²C, иллюстрируется рис. 5.6. Подключение к шине

и принципы работы приемных устройств комментариев не требуют. Что касается передатчиков, то для них обязательна конфигурация с открытым коллектором или с открытым стоком. Соединение передатчиков через шину эквивалентно их *проводному И*.

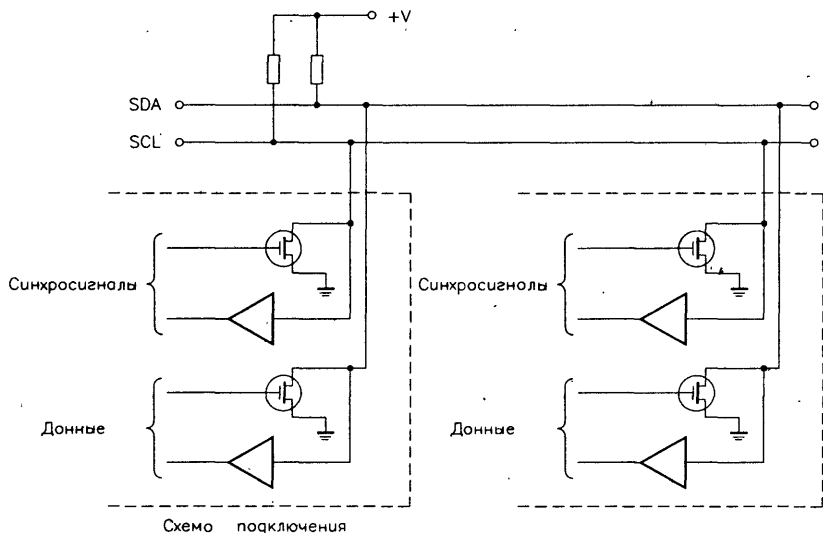


Рис. 5.6

Принципы подключения к шине I²C

Нагрузочные резисторы передатчиков линий SDA и SCL должны быть подсоединены к положительному напряжению. От этого напряжения зависят логические уровни сигналов шины. Мы будем говорить о положительной логике, при которой высокому уровню напряжения соответствует *логическая единица*, а низкому – *логический ноль*.

Если ни один из абонентов ничего не передает на шину, сигналы линий SDA и SCL имеют высокие уровни.

Данные, передаваемые по шине, считаются достоверными, когда сигнал на линии SCL имеет высокий уровень (рис. 5.7). Передатчик должен устанавливать данные на линию SDA при низком уровне линии SCL и поддерживать их в течение всего времени, когда уровень линии SCL высокий.

Так как передача осуществляется в последовательной форме, в посылках должна содержаться информация о ее начале и конце.

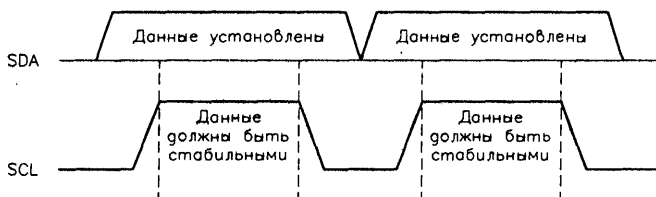


Рис. 5.7

Временные диаграммы передачи битов данных по шине I²C

Информация о начале называется здесь *стартовым условием* (старт-условие), а информация о конце – *стоповым условием* (стоп-условие).

Стартовому условию соответствует переход сигнала линии SDA от высокого уровня к низкому, в то время как сигнал линии SCL остается на высоком уровне. Стоповое условие выполняется, когда сигнал SDA переходит с низкого уровня к высокому, сигнал SCL при этом остается на высоком уровне.

Данные по шине передаются байтами. Старший бит каждого байта передается первым. За каждым байтом следует бит подтверждения от получателя. Временные диаграммы передачи данных по шине представлены на рис. 5.8.

Инициатором обмена является абонент, называемый *master*. Он формирует сигналы на линию SCL. Абонент, к которому обращается *master*, называется *slave*. Его задача – исполнить предписанные ему команды по приему и передаче данных.

Последовательность передачи следующая. Сначала *master* генерирует стартовое условие (рис. 5.8), затем осуществляет передачу семirazрядного служебного адреса, идентифицирующего абонента (*slave*), с которым он будет обмениваться информацией. Затем производится собственно передача данных байт за байтом. После пересылки байта принимающий абонент, которым может быть и *master* и *slave*, должен сформировать бит (девятый) подтверждения приема (низкий уровень).

Процесс может продолжаться до завершения передачи всех данных. Когда этот обмен заканчивается, *master* генерирует стоп-условие и шина освобождается.

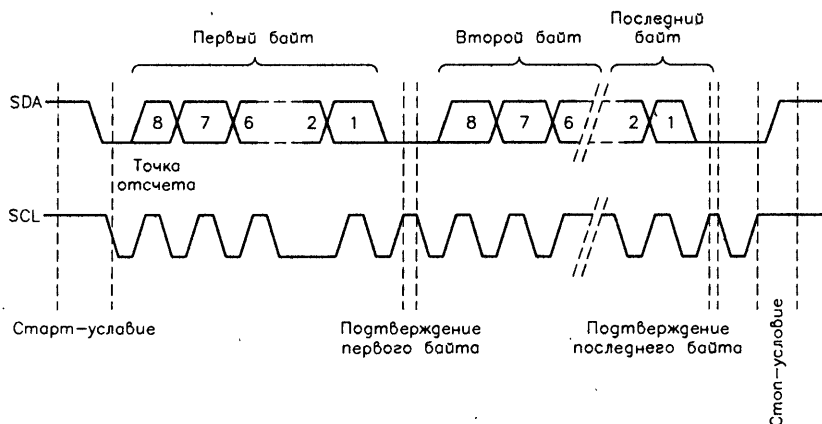


Рис. 5.8

Временные диаграммы передачи данных по шине I²C

Форматы передачи

Теперь мы знаем, как проходит обмен, и нам остается только рассмотреть формат передачи данных для того, чтобы понять, как осуществляется адресация и определение направления передачи данных.

На рис. 5.9 показано содержимое первого байта, который должен передаваться всегда в начале обмена. Семь старших битов первого байта содержат адрес получателя сообщения, что позволяет адресовать сообщение одному из 128 абонентов. Младший бит определяет

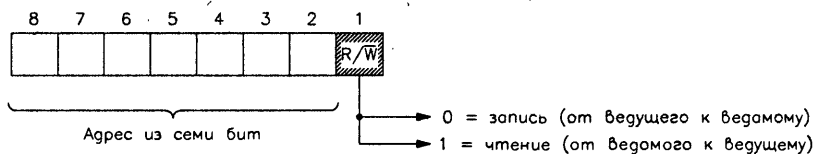


Рис. 5.9

Формат первого байта

операцию, которую собирается осуществить master – считывание или запись. Если этот бит равен нулю, master собирается записать данные в slave-устройство, т.е. посылает ему данные. Если бит равен единице – master собирается считать данные из slave-устройства, то есть собирается получить данные от подчиненного.

Когда master желает осуществлять несколько обменов с подчиненными устройствами, имеющими различные служебные адреса, ему не обязательно заканчивать первый обмен посылкой стоп-условия. Можно сразу начать новый обмен посылкой старт-условия, первое подчиненное устройство при этом автоматически отключится.

Наконец, существует процедура, называемая общим обращением, при которой служебный адрес, посланный абонентом master, равен нулю. Все микросхемы, подсоединенные к шине и способные ответить на такое общее обращение, должны это сделать, принимая данные, которые проходят в текущий момент по шине. Их действия зависят от бита команды (считывания/записи) первого байта. Действительно, если этот бит равен нулю, второй байт играет особую роль, которую мы не будем сейчас детально рассматривать. Скажем только, что благодаря этому любой из подчиненных абонентов получает возможность запрограммировать свой адрес с помощью программного обеспечения или аппаратных средств.

Когда этот бит равен единице, имеет место общее обращение, переданное абонентом master (например, контроллером клавиатуры). Такая микросхема неспособна генерировать адрес получателя информации, которому она должна ее посылать. В этих условиях следующий байт содержит адрес абонента master, чтобы интеллектуальное подчиненное устройство, например микроконтроллер, его признал и принял переданную им информацию.

Схема реализации интерфейса I²C

В этом примере PIC-микроконтроллер 16C54 используется как абонент шины I²C. Предлагаемая базовая схема может применяться в любом устройстве, работающем с интерфейсом I²C в режиме подчиненного устройства (slave).

Схема устройства представлена на рис. 5.10. Две линии порта A использованы в качестве линий SDA и SCL. Резистор R2 необязателен и должен быть установлен, только если его нет в master-устройстве шины.

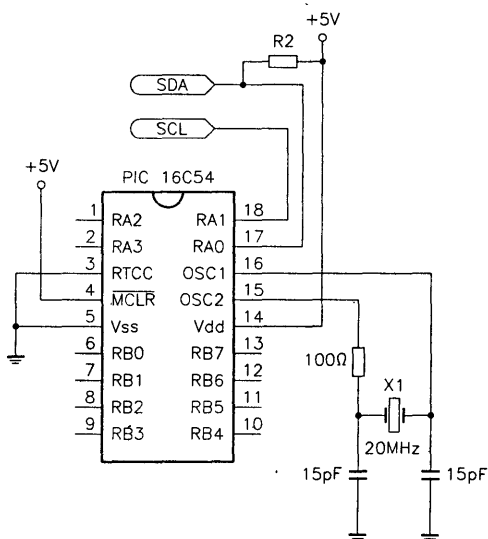


Рис. 5.10

Схема использования PIC 16C54 для реализации интерфейса I²C

Микросхема опознает служебный адрес, определяемый величиной `DEVICE_ADDRESS` (см. листинг 5.2), которая выбрана по умолчанию равной D6 (и D7 для считывания). Это значение предусмотрено фирмой Philips для ИС типа PCF8573 (часы реального времени – календарь).

Используемый PIC-микропроцессор 16C54 ведет себя как периферийное устройство шины I²C, содержащее восемь доступных через субадреса внутренних регистров. Он реализует также так называемый канал данных с нулевым субадресом, через который могут считываться последовательности символов.

Доступные через шину регистры в программе идентифицируются как I2CR0 – I2CR7, при этом I2CR0 соответствует субадресу 8. При обращении по нулевому субадресу микросхема выдает последовательность идентификации.

Алгоритм, представленный на рис. 5.11, иллюстрирует принцип работы этой программы, а ее текст приведен в листинге 5.2. Она может быть легко адаптирована к вашим приложениям с помощью соответствующих макрокоманд:

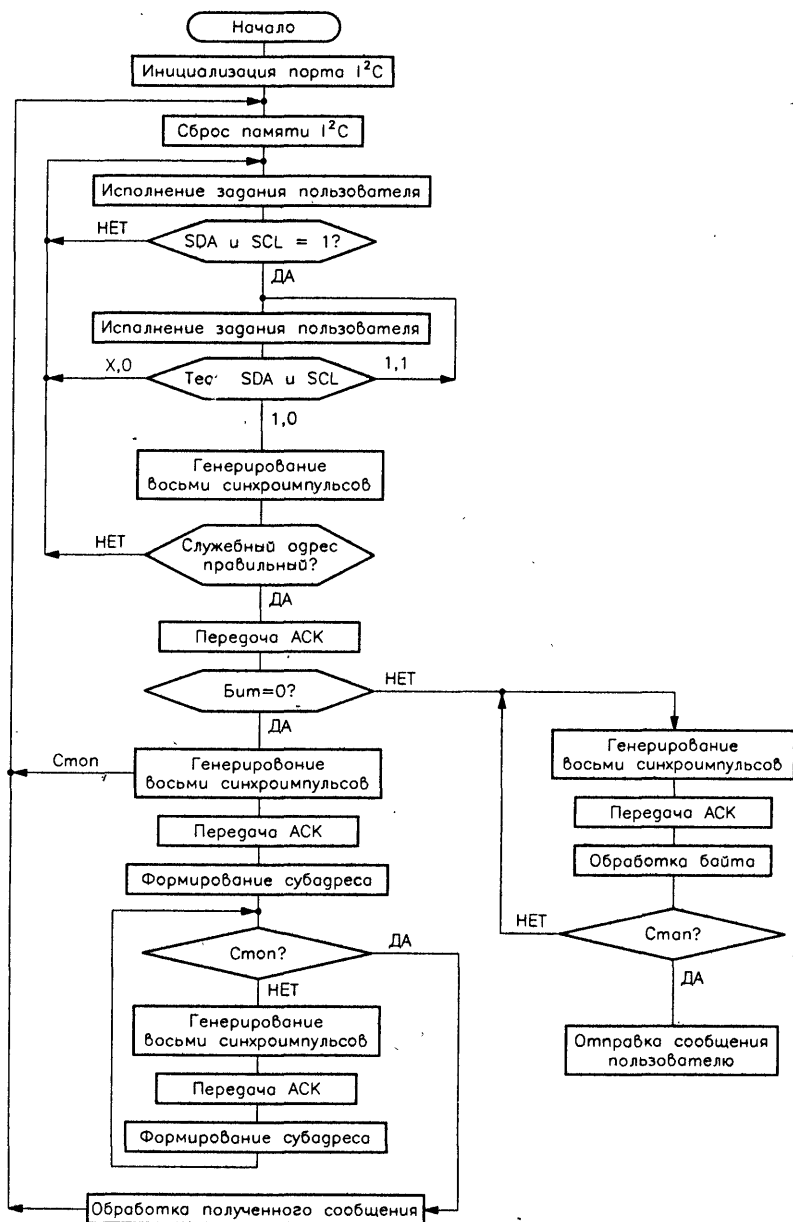


Рис. 5.11

Алгоритм программы для реализации шины I²C

- ◆ USER_MAIN – главная часть программы вашего приложения, размер которой не ограничен. Если она слишком большая, PIC не сможет сразу отвечать на некоторые обращения по шине, но в соответствии с принципом работы шины I²C они будут повторены master-устройством;
- ◆ USER_Q – предназначена для создания приложений, работающих в реальном времени. В большинстве случаев эта макрокоманда не используется, но если вы все же захотите ее применить, то ее инструкции не должны выполняться дольше 8 мкс;
- ◆ USER_MSG – макрокоманда, которую надо выполнять после получения запроса шины I²C. Она содержит код, необходимый для специфической обработки сообщения;
- ◆ USER_RECV – должна выполняться после получения байта. Она осуществляет обработку адресов;
- ◆ USER_XMIT – макрокоманда, которая готовит байт к передаче. В листинге, представленном в качестве примера, она обнаруживает нулевой субадрес и посылает так называемую последовательность идентификации.

Листинг 5.2

```
; LIST P=16C54 , C=80 , N=0, R=DEC.
;
; Программная реализация шины I2C.
; Соответствует инструкции по применению AN541 фирмы Microchip.
;
CPU      EQU    1654
SIM      EQU    0
      IF      (CPU==1654) || (CPU == 1655)
_RESVEC  EQU    01FFH      ; Стартовый адрес 16C54.
      ENDIF
      IF      CPU == 1656
_RESVEC  EQU    03FFH      ; Стартовый адрес 16C56.
      ENDIF
      IF      CPU == 1657
_RESVEC  EQU    07FFH      ; Стартовый адрес 16C57.
      ENDIF
; *****Вектор сброса*****
      ORG     _RESVEC      ;
RESVEC   ;
      GOTO    INIT        ;
; *****
; *****
; Макросы для обнуления, установки (в 1) битов и условных переходов,
; зависящих от состояния битов.
```

```

;*****
;* Синтаксис Описание
;* BIT label,bit,file ; Идентифицируем бит.
;* SEB label ; Устанавливаем бит в единицу.
;* CLB label ; Обнуляем бит.
;* SKBS label ; Пропускаем, если бит равен единице.
;* SKBC label ; Пропускаем, если бит равен нулю.
;* BBS label,address ; Переходим, если бит равен единице.
;* BBC label,address ; Переходим, если бит равен нулю.
;* CBS label,address ; Вызываем подпрограмму, если бит равен единице.
;* CBC label,address ; Вызываем подпрограмму, если бит равен нулю.
;*****
;
BIT MACRO label, bit, file ; Определение бита
label EQU file<<8|bit ; (macro).
ENDM
SEB MACRO label ; Установить бит.
BSF label>>8,label&7 ; (macro)
ENDM
CLB MACRO label ; Обнулить бит.
BCF label>>8,label&7 ; (macro)
ENDM
SKBS MACRO label ; Пропустить, если бит установлен (=1).
BTFSS label>>8,label&7 ; (macro)
ENDM
SKBC MACRO label ; Пропустить, если бит обнулен.
BTFSC label>>8,label&7 ; (macro)
ENDM
BBS MACRO label,address ; Перейти, если бит установлен.
BTFSC label>>8,label&7 ; (macro)
GOTO address ; (macro)
ENDM
BBC MACRO label, address ; Перейти, если бит обнулен.
BTFSS label>>8,label&7 ; (macro)
GOTO address ; (macro)
ENDM
CBS MACRO label, address ; Вызвать подпрограмму, если бит установлен.
Call label>>8,label&7 ; (macro)
ENDM
CBC MACRO label,address ; Вызвать подпрограмму, если бит обнулен.
Call label>>8,label&7 ; (macro)
ENDM

; ; Для сокращения исходного текста.
W EQU 0 ; Для указания рабочего регистра W.
w EQU 0 ; Для указания рабочего регистра W.
F EQU 1 ; Для указания в качестве приемника других регистров.
f EQU 1 ; Для указания в качестве приемника других регистров.

```

```

;*****
; * Объявление регистров.
;*****
ORG 0 ; Для объявления регистров.
ind RES 1 ; 0 - псевдорегистр для косвенной адресации (FSR).
RTCC RES 1 ; 1 - таймер реального времени.
PC RES 1 ; 2 - программный счетчик (PC).
STATUS RES 1 ; 3 - регистр состояния (STATUS).

; * Биты регистра состояния.
BIT B_C,0,STATUS ; Флаг переноса (перенос).
BIT B_DC,1,STATUS ; Флаг дополнительного переноса.
BIT B_Z,2,STATUS ; Флаг нуля.
BIT B_PD,3,STATUS ; Флаг режима пониженного потребления (sleep).
BIT B_TO,4,STATUS ; Флаг сброса по сторожевому таймеру (watchdog).
BIT B_PAO,5,STATUS ; Выбор банка регистров (младший разряд).
BIT B_PA1,6,STATUS ; Выбор банка регистров (старший разряд).
BIT B_PA2,7,STATUS ; Бит общего назначения (в некоторых микроконтроллерах
; используется для выбора банков регистров):
FSR RES 1 ; 4 - регистр косвенного адреса.
PORTA RES 1 ; 5 - регистр порта A (4 бита).
PORTB RES 1 ; 6 - регистр порта B.
IF (CPU = 1655) || (CPU = 1657)
PORTC RES 1 ; 7 - регистр порта C (только для PIC 16C54/56).
ENDIF

; Регистры, используемые данной программой.
I2CFLG RES 1 ; Регистр флагов I2C.
;
; Идентификаторы I2C.
BIT B_RD,0,I2CFLG ; 1 - чтение.
BIT B_UA,1,I2CFLG ; 0 - чтение служебного адреса.
BIT B_SA,2,I2CFLG ; 1 - чтение субадреса.
BIT B_ID,3,I2CFLG ; 1 - чтение последовательности идентификации.
;
I2CREG RES 1 ; Регистр ввода/вывода шины I2C.
I2CSUBA RES 1 ; Субадрес.
I2CBITS RES 1 ; Счетчик выданных битов.
;*****
; * Восемь псевдорегистров, доступных по субадресам 1-8.
; * Регистры считывания/записи.
;*****
I2CR0 EQU $ ; Субадрес 8.
RES 1 ; Восемь псевдорегистров.
I2CR1 EQU $ ; Субадрес 1.
RES 1
I2CR2 EQU $ ; Субадрес 2.
RES 1

```

```

I2CR3 EQU    $      ; Субадрес 3.
RES      1
I2CR4 EQU    $      ; Субадрес 4.
RES      1
I2CJ5 EQU    $      ; Субадрес 5.
RES      1
I2CJ6 EQU    $      ; Субадрес 6.
RES      1
I2CE7 EQU    $      ; Субадрес 7.
RES      1

```

; Константы, используемые программой.

```

DEVICE_ADDRESS EQU    0D6H      ; Служебный адрес устройства при записи по шине I2C,
                                   ; при чтении - 0D7.

```

; ** Определение порта A.

; ** Интерфейс I2C использует порт A.

; ** Линии данных SDA соответствует RA0.

```

TAREAD EQU    B'11110111'      ; Значение регистра TRISA для чтения.
TAWRITE EQU   B'11110110'      ; Значение регистра TRISA для записи.
TAINIT EQU    TAREAD           ; Начальное значение регистра TRISA.
                BIT    B_SDA,0,PORTA ; Разряд 0 порта A соответствует сигналу SDA
                BIT    B_SCL,1,PORTA ; Разряд 1 соответствует сигналу SCL.
; spare      B_???,2,PORTA ; Разряды 2 и 3 не используются.
; spare      B_???,3,PORTA ; /

```

; **

; ** Определение порта B (параллельный выход).

; **

```

TBINIT EQU    B'00000000'      ; Данное значение регистра TRISB задает все
                                   ; линии как выходные.
PBINIT EQU    B'11111111'      ; Начальное значение регистра данных
                                   ; порта B (PORTB init).

```

; * Макросы, предназначенные для программ пользователя.

; * Смотрите их описание в тексте книги.

; *

; * USER_MAIN: главная программа.

; * USER_Q: программа, выполняемая во время обмена, если он длится менее 8 мкс.

; * USER_MSG: программа обработки запроса I2C.

```

USER_MAIN MACRO

```

; *** Главная программа.

ENDM

```

USER_Q MACRO

```

; *** "Быстрая" программа (см. выше).

ENDM

```

USER_MSG      MACRO
;*** Обработка запроса шины I2C.
    ENDM
USER_RECV      MACRO
;
;*** Обработка полученного байта.
;*** В этом примере предназначенные данные посылаются под адресом 0 на порт b.
;
    BBC      B_ID, NXI_notid; Канал 0! Бит устанавливается, если начальный адрес 0.
    MOVFW    I2CREG          ; Принимаем биты.
    MOVWF    PORTB          ; и записываем их в PORTB.
    GOTO     IN_CONT
_NXI_notid
    ENDM
USER_XMIT      MACRO
;
;*** Подготовка байта к передаче.
;*** В этом примере посылается последовательность идентификации.
;
    BBC      B_ID, NXO_notid ; Канал 0! Бит устанавливается, если начальный адрес 0.
    CALL     GETID           ; Вызываем подпрограмму получения следующего байта
                              ; последовательности идентификации (ID)
    GOTO     OUT_CONT        ; и посылаем его.
_NXO_notid
    ENDM

;*****
; Начало программы.
;*****
    ORG      0
;*****
GETID
    MOVFW    I2CSUBA          ; Загружаем субадрес в регистр W.
    ANDLW    07H             ; Не более восьми значений.
    ADDWF    PC, f
;*****
; * Идентифицирующая последовательность.
;*****
    RETLW    "P"
    RETLW    "I"
    RETLW    "C"
    RETLW    "I"
    RETLW    "2"
    RETLW    "C"
    RETLW    0
    RETLW    0
;*****
; Интерфейс шины I2C использует порт A.
; Линии SDA соответствует порт RA0 для упрощения программы.

```

```

;*****
; ** INIT
; ** Старт при аппаратном сбросе.
; **
;*****
INIT                ; Включение питания
;*****
; ** RESET
; ** Старт при программном сбросе.
; **
;*****
RESET              ; Программный сброс.
    MOVLW    TAINIT ; Инициализация портов.
    TRIS     PORTA
    MOVLW    TBINIT
    TRIS     PORTB
    MOVLW    PBINIT
    MOVWF    PORTB
;*****
; Основной цикл ожидания.
;
;*****
I2CWAIT
    CLRWD    ; Сброс сторожевого таймера.
    CLB      B_UA ; Инициализация флагов состояния.
    CLB      B_SA ; Инициализация флагов состояния.
    CLB      B_RD ; Инициализация флагов состояния.
loop1
    CLRWD    ; Сброс сторожевого таймера.
    USER_MAIN ; Режим ожидания, выполняется программа пользователя.
    SKBC     B_SDA ; Ожидаем, пока SDA&SCL = H.
loop2
    SKBS     B_SCL ;
    GOTO     loop1 ; Прерываем ожидание.
    CLRWD    ; Сброс сторожевого таймера.
    USER_MAIN ; Исполняется программа пользователя.
; ** Ожидание старта.
;
    SKBC     B_SCL ; Если низкий уровень CLK, пропускаем.
    SKBC     B_SDA ; Низкий уровень SDA... Старт!
    GOTO     loop2
; ** Старт-условие обнаружено! – Ожидание первого бита!
;
loop3
    BBS      B_SDA, I2CWAIT ; Выходим, если на линии SDA появился высокий уровень,
                           ; но до того, как уровень сигнала SCL стал низким.
    BBS      B_SCL, loop3 ; Ожидаем, пока SCL = 0.

```

• NEXTBYTE

```

    CLRWDI          ; Сброс сторожевого таймера.
    MOVLW 1          ; Инициуем прием байта.
    MOVWF I2CREG
; ** Передача битов!
; ** SCL = 0 - значит, данные могут изменяться.
; ** Для этого есть 4 мкс.
loop4
    USER_0
loop4A
    BBC B_SCL,loop4A ; Ожидаем, пока SCL = 1.
;
; *** SCL = 1. Сдвиг битов.
;
    RRF PORTA,W      ; Перемещаем RAO в C.
    RLF I2CREG,F      ; Перемещаем перенос в регистр I2CREG.
    SKPNC             ; Пропустить, если не C.
    GOTO ACK_I2C      ; Подтверждение приема байта.
    BTFSC I2CREG,0    ; Пропускаем, если бит данных был равен 0.
    GOTO ii_1         ; Этот бит был установлен.
ii_0
    BBC B_SCL,loop4   ; Ожидаем, пока SCL = 1.
    SKBS B_SDA        ; Если SDA 0 переходит в 1, то это стоп-условие.
    GOTO ii_0
I2CSTOP
    USER_MSG         ; Сообщение о завершении процесса!
    GOTO I2CWAIT      ; Возвращаемся в главную программу (цикл).
ii_1
    BBC B_SDA,I2CWAIT ; Если SDA 1 переходит в 0, то это старт-условие.
    BBC B_SCL,loop4   ; Переходим, если SCL = 0.
    GOTO ii_1
ACK_I2C
    BBC B_UA,ACK_UA   ; Проверяем адрес устройства.
    BBS B_SA,ACK_SA   ; Чтение вторичного адреса.
; ****
; ** Обработка полученного байта (до подтверждения).
; ** Не подтверждать, если байт не обработан.
; ****
    USER_RECV
    MOVLW 07H         ; Загружаем счетчик.
    ANDWF I2CSUBA,f    ; Ограничиваем модуль счета.
    MOVLW I2CRO       ; Псевдорегистры.
    ADDWF I2CSUBA,W    ; Смещение от начала буфера.
    INCF I2CSUBA       ; Следующий субадрес.
    MOVWF FSR          ; Косвенный адрес.
    MOVWF I2CREG
    MOVWF ind          ; Поместить данные в регистр.
IN_CONT              ; Точка продолжения.

```


ACKloop

```

BBS    B_SCL,ACKloop ; Ожидаем, пока SCL = 0.
CLB    B_SDA          ; Подтверждаем прием.
MOVLW  TAWRITE
TRIS   PORTA
CLB    B_SDA          ; Устанавливаем подтверждение приема.

```

ACKloop2

```

USER_0 ; "Быстрая" программа пользователя.
BBC    B_SCL,ACKloop2 ; Ожидаем, пока SCL = 1.

```

ACKloop3

```

USER_0
BBS    B_SCL,ACKloop3 ; Ожтдаем, пока SCL = 0.
MOVLW  TAREAD         ; Заканчиваем подтверждение приема.
TRIS   PORTA
BBC    B_RD,NEXTBYTE  ; Пропускаем, если чтение
                        ; (подтверждаем только прием адреса).

```

```

;*****
; Чтение по шине I2C.
; Здесь должна быть размещена программа пользователя.
; Подпрограмма считывает данные, определенные I2CSUBA, и передает их на шину I2C.
; Эта версия ограничивает число последовательно считываемых байтов восьмью
; за счет конъюнкции (И) номера регистра с 7. Если необходимо передавать
; большее количество байтов, код может быть изменен.
;*****

```

NEXTOUT

```

;*****<<< Поместить следующий байт в регистр I2C. >>>*****
USER_XMIT ; Программа подготовки очередного байта для передачи (например,
           ; программа, формирующая последовательность идентификации).
MOVLW  07H ; Устанавливаем счетчик.
ANDWF  I2CSUBA,f ; Ограничиваем модуль счета.
MOVLW  I2CRO ; Псевдорегистры.
ADDWF  I2CSUBA,W ; Смещение от начала буфера.
INCF   I2CSUBA ; Следующий субадрес.
MOVWF  FSR ; Косвенный адрес.
MOVWF  ind ; Получаем данные.

```

OUT_CONT

```

MOVWF  I2CREG
; ** Инициализация счетчика битов.
MOVLW  8 ; Устанавливаем счетчик битов.
MOVWF  I2CBITS

```

```

; ** Пересылка битов.
; ** SCL = 0 - значит, изменяем данные.
;

```

iiOUT_loop_1

```

RLF    I2CREG,F ; Левый сдвиг данных. Старший бит помещается вперед.
SKPNC  ; Если перенос равен нулю, то пропускаем.

```

```

GOTO    iiOUT_1          ; Выдаем следующую единицу.
CLB     B_SDA            ; Выдаем нуль.
MOVLW   TAWRITE
TRIS    PORTA
CLB     B_SDA

iiOUT_0
    CLRWDOT              ; Сбрасываем сторожевой таймер.
    USER_Q

iiOUT_loop_02
    BBC     B_SCL,iiOUT_loop_02 ; Ожидаем, пока SCL = 1.
    USER_Q

iiOUT_loop_03
    BBS     B_SCL,iiOUT_loop_3  ; Ожидаем, пока SCL = 0.
    DECFSZ  I2CBITS            ; Инкрементируем счетчик битов.
    GOTO    iiOUT_loop0        ; Переходим к передаче последнего бита.
    MOVLW   TAREAD             ; После передачи последнего бита устанавливаем 1
                                ; для подтверждения (ACK).

    TRIS    PORTA
    GOTO    iiOUT_ack          ; Получаем подтверждение (ACK).

iiOUT_loop_0
    RLF     I2CREG,f           ; Левый сдвиг данных.

; Старший бит (MSB) переносится вперед.

    SKPC              ; Пропускаем, если перенос равен 0.
    GOTO    iiOUT_0        ; Выводим следующий 0.
    MOVLW   TAREAD         ; Устанавливаем 1.
    TRIS    PORTA

iiOUT_1
    CLRWDOT              ; Обнуляем сторожевой таймер.
    USER_Q

iiOUT_loop_12
    BBC     B_SCL,iiOUT_loop_12 ; Ожидаем, пока SCL = 1.
    USER_Q

iiOUT_loop_13
    BBS     B_SCL,iiOUT_loop_13 ; Ожидаем, пока SCL = 1.
    DECFSZ  I2CBITS            ; Считаем биты.
    GOTO    iiOUT_loop_1      ; Повторяем до последнего бита, равного 1.

iiOUT_ack
    INCF    I2CSUBA          ; Получить подтверждение.
                                ; Следующий субадрес.

iiOUT_Loop_a2
    BBC     B_SCL,iiOUT_loop_a2 ; Ожидаем, пока SCL = 1.
    BBS     B_SDA,I2CWAIT     ; Если нет подтверждения - ждать до перезапуска!

; Подготовка следующего байта.

iiOUT_Loop_a3

```

```

BBC      B_SCL,NEXTOUT      ; Ожидаем, пока SCL = 0. Выдаем следующий знак.
BBS      B_SDA,iIOUT_loop_a3 ; Ожидаем нового стартового условия.
GOTO     I2CWAIT            ; Прекращаем прием.
USER_READ                                ; Программа пользователя по обработке данных.
GOTO     I2CWAIT

```

```

; *****
; * Получен адрес, выполняется его проверка.
; *****

```

ACK_UA

```

SEB      B_UA                ; Установить флажок получения адреса.
BTFSCL  I2CREG,0             ; Пропустить, если вводятся данные.
SEB      B_RD                ; Флаг чтения.
MOVF     I2CREG,W            ; Получение адреса.
ANDLW    OFEH                ; Маскируем флаг направления перед сравнением.
XORLW    DEVICE_ADDRESS      ; Адрес устройства.
BNZ      I2CWAIT            ; Чужой адрес. Сообщение игнорируем.
BBS      B_RD,ACKloop        ; Чтение - нет вторичного адреса.
SEB      B_SA                ; Следующий байт - вторичный адрес.
GOTO     ACKloop            ; Да! Подтверждаем прием адреса и продолжаем.

```

```

; *****
; * Получен второй адрес.
; * SA = 0 преобразуется в 128 для упрощения считывания ID.
; *****

```

ACK_SA

```

CLB      B_SA                ; Флаг приема вторичного адреса.
CLB      B_ID
MOVFW    I2CREG              ; Получаем субадрес.
SKPNZ    ; Пропускаем, если не 0.
SEB      B_ID                ; Флажок - идентификационное поле выбрано.
MOVWF    I2CSUBA             ; Установить подадреса.
GOTO     ACKloop
END

```

ЧЕТЫРЕХКАНАЛЬНЫЙ ВОЛЬТМЕТР СО СВЕТОДИОДНОЙ ИНДИКАЦИЕЙ

Данное устройство использует рассмотренные нами в главе 3 принципы организации динамической индикации многоразрядных индикаторов с одновременным управлением матричной клавиатурой. Кроме того, оно обеспечивает измерение аналоговых сигналов на четырех входах, которые совмещены с линиями управления разрядами индикатора.

В устройстве используется PIC-микроконтроллер типа 16C71, имеющий встроенный аналого-цифровой преобразователь с четырьмя

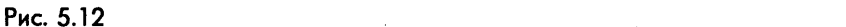


Схема четырехканального вольтметра

Устранить влияние цифровых выходов на измеряемые аналоговые сигналы можно двумя способами. Одно из решений представлено

на рис. 5.13: здесь используется внешний операционный усилитель, включенный в качестве повторителя напряжения. Это решение позволяет значительно сократить интерференционные шумы в широкой полосе частот входных аналоговых сигналов. Второе решение представлено на рис. 5.14 и заключается в использовании фильтрующей RC-цепочки. Такое решение приемлемо, если сопротивление источника аналогового сигнала остается относительно небольшим. В противном случае надо обязательно использовать решение, проиллюстрированное рис. 5.13.

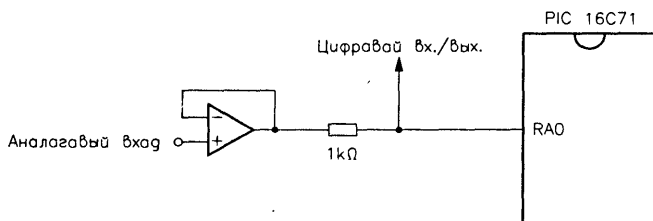


Рис. 5.13

Первый способ устранения интерференции между источником аналоговых сигналов и цифровыми входами/выходами

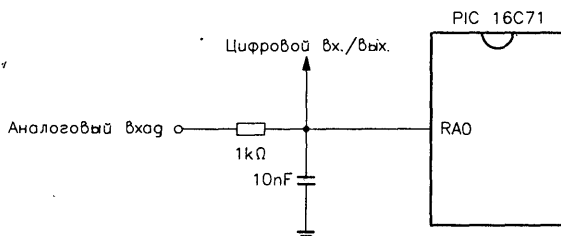


Рис. 5.14

Второй способ устранения интерференции между источником аналоговых сигналов и цифровыми входами/выходами

Программа, которая обеспечивает управление микроконтроллером, достаточно проста (листинг 5.3). Аналоговые входы опрашиваются каждые 20 мс. Для этого сначала выключаются индикаторы за счет подачи низких логических уровней на все линии порта В.

Затем линии порта А переводятся в режим аналоговых входов. Для стабилизации постоянного напряжения на аналоговых входах устанавливается цикл ожидания, соотношенный с инерционностью используемых в нашем примере RC-цепочек.

Предлагаемая программа содержит различные подпрограммы, осуществляющие управление индикаторами, клавиатурой и аналого-цифровым преобразователем. В целом она реализует функции четырехканального вольтметра.

При включении устройство автоматически переходит в режим измерения напряжения по входу RA0, результат представляется на индикаторе. Выбор измеряемого канала производится нажатием соответствующей клавиши (0–3). Нажатия других клавиш игнорируются, и могут быть использованы в вашем приложении для реализации различных функций.

Листинг 5.3

```

;*****
; Эта программа иллюстрирует возможность одновременного управления
; четырехразрядным индикатором, клавиатурой и измерения напряжений
; четырех аналоговых входов посредством микроконтроллера PIC 16C71.
; При включении питания устройство индицирует значение напряжения на входе 0.
; При нажатии одной из клавиш (0–3) происходит переключение входов.
; Обновление индикации осуществляется каждые 20 мс,
; сканирование клавиатуры производится с той же периодичностью.
; Аналоговые входы опрашиваются по очереди, что обеспечивает обновление каждые 80 мс.
; Таймер (RTCC) использован для генерации прерываний каждые 5 мс.
;
; Программа соответствует инструкции по применению AN557 фирмы Microchip.
;*****
LIST      P=16C71,      F=INHX8M
;
include "picreg.equ"
;
TempC     equ    0x0c    ; Временные регистры общего назначения.
TempD     equ    0x0d
TempE     equ    0x0e
PABuf     equ    0x20
PBBuf     equ    0x21
Count     equ    0x0f    ; Счетчик.
MsdTime   equ    0x10    ; Старший байт таймера.
LsdTime   equ    0x11    ; Младший байт таймера.
Flag      equ    0x12    ; Регистр флагов общего назначения.
keyhit    equ    0       ; бит 0 -> флаг нажатия клавиши,

```

```

DebnceOn equ 1 ; бит 1 -> антидребезг включен,
noentry equ 2 ; данные не введены = 0,
ServKey equ 3 ; бит 3 -> служебный ключ,
ADOver equ 4 ; бит 4 -> флаг завершения преобразования АЦП.
Debnce equ 0x13 ; Счетчик антидребезгового автомата.
NewKey equ 0x14
DisplayCh equ 0x15 ; Номер отображаемого канала.
ADTABLE equ 0x16 ; Резервные четыре ячейки с адреса 0x16 до 0x19.
Wbuffer equ 0x2f
StatBuffer equ 0x2e
OptionReg equ 1
PCL equ 2

```

; Макрокоманда сохранения контекстных регистров;

```

push macro
    movwf WBuffer ; Переслать содержимое регистра W в буфер.
    swapf WBuffer ; Поменять местами тетрады.
    swapf STATUS,w ; STATUS помещается в регистр W.
    movwf StatBuffer ; Сохранить.
endm

```

; Макрокоманда восстановления контекстных регистров.

```

pop macro
    swapf StatBuffer,w ; Восстановить STATUS.
    movwf STATUS ; /
    swapf WBuffer,w ; Восстановить регистр W.
endm
org 0
goto Start ; Переход к началу программы.
org 4

```

; Первичная обработка прерывания.

; Рекомендуется сохранить регистр W и регистр состояния на время прерывания.

```

;
    push
    call ServiceInterrupts
    pop
    retfie

Start
    call InitPorts
    call InitAd
    call InitTimers

Loop
    btfsc ServKey ; Проверяем флаг ServKey.
    call ServiceKey ; Если установлен, вызываем подпрограмму обработки.
    btfsc ADOver ; Проверка флага ADOver.
    call ServiceAD ; Если установлен, то переходим к обработке.
    goto loop

```

; Обработка нажатия клавиш.

ServiceKey

```
bcf    ServKey      ; Обнуляем флаг.
movf   NewKey,w     ;
sublw  3            ; NewKey > 3?
btfss  STATUS,C     ; Если нет, пропускаем команду,
return ; иначе игнорируем нажатие.
movf   NewKey,w     ;
movwf  DisplayCh    ; Переключаем канал.
```

LoadAD

```
movlw  ADTABLE      ; Получаем адрес вершины таблицы АЦП.
addwf  DisplayCh,w  ; Прибавляем смещение.
movwf  FSR          ; Загружаем косвенный адрес (FSR).
movf   0,w          ; Считываем код АЦП
movwf  L_byte       ; и помещаем в L_byte.
clrf   H_byte
call   B2_BCD
movf   R2,W         ; Получаем младшие цифры.
movwf  LsdTime      ; Сохраняем в LsdTime.
movf   R1,W         ; Получаем младшие цифры.
movwf  MsdTime      ; Сохраняем в Msd.
return
```

```
;
; Эта подпрограмма загружает результат преобразования (ADRES)
; в ячейку, определенную номером желаемого входа.
; Если он нулевой, ADRES поступает в ADTABLE, а если равен 1,
; ADRES поступает в ADTABLE+1 и т.д.
```

ServiceAD

```
movf   ADCON0,w     ;
movwf  TempC        ; Сохраняем во временном регистре.
movlw  B'00001000'  ; Выбираем следующий канал.
addwf  ADCON0,w     ; /
btfsc  ADCON0,5     ; Если <= ch3, то пропускаем.
movlw  B'11000001'  ; Выбираем ch0.
movwf  ADCON0       ; Записать адрес в таблицу.
movlw  ADTABLE
movwf  FSR          ; Загрузить в регистр косвенного адреса адрес вершины.
rrf    TempC
rrf    TempC
rrf    TempC,w
andlw  3            ; Маскируем все биты, кроме двух младших.
addwf  FSR          ; Прибавляем смещение.
movf   ADRES,w      ; Считываем результат АЦП
movwf  0            ; и помещаем в таблицу по косвенному адресу.
```



```

bcf    ADOVer        ; Обнуляем флаг АЦП.
call   LoadAD        ; Загружаем результат в регистр дисплея.
return

```

InitPorts

```

bsf     STATUS,RPO    ; Выбираем банк 1 ОЗУ.
movlw   3              ; Задаем RA0 - RA3 как цифровые порты.
movwf   ADCON1         ; /
clrf    TRISA          ; Определяем RA0 - RA4 как выходы.
clrf    TRISB          ; Определяем RB0 - RB7 как выходы.
bcf     STATUS,RPO    ; Выбираем банк 0.
clrf    PORT_A         ; Обнуляем все выходы.
clrf    PORT_B         ; /
bsf     PORT_A,3       ; Разрешить отображение старшей цифры.
return

```

```

; При тактировании частотой 4,096 МГц внутренняя частота микроконтроллера
; составляет 1,024 МГц, что при делении на 32 обеспечивает инкрементирование
; таймера (RTCC) каждые 31,25 мкс.
; Переполнение таймера (при значении 96) будет происходить каждые 5 мс,
; прерывания будут генерироваться с той же периодичностью.

```

InitTimers

```

clrf    MsdTime        ; Сброс регистров таймера.
clrf    LsdTime        ; /.
clrf    DisplayCh      ; Отобразить результат измерения канала 0.
clrf    Flag           ; Обнулить все флажки.
bsf     STATUS,RPO    ; Банк 1.
movlw   B'10000100'    ; Определяем коэффициент делителя.
movwf   OptionReg      ; Коэффициент равен 32.
bcf     STATUS,RPO    ; Банк 0.
movlw   B'00100000'    ; Разрешаем прерывание по переполнению таймера.
movwf   INTCON         ;
movlw   .96            ; Загружаем начальное значение таймера (rtcc).
movwf   RTCC           ; и запускаем его.
retfie

```

ServiceInterrupts

```

btfs    INTCON,RTIF    ; Проверяем флаг прерывания по таймеру.
goto    ServiceRTCC    ; Если флаг равен 1, то обслуживаем,
clrf     INTCON         ; иначе запрещаем все прерывания.
bsf     INTCON,RTIE    ; Разрешаем прерывание по таймеру.
return

```

ServiceRTCC

```

movlw   .96            ; Устанавливаем исходное состояние таймера.
movwf   RTCC
bcf     INTCON,RTIF    ; Сбрасываем флаг прерывания таймера.
btfs    PORT_A,0       ;
call    ScanKeys       ; Сканируем клавиши каждые 20 мс.
btfs    PORT_A,3       ;

```

```

call    SampleAd      ; "Сканируем" АЦП каждые 20 мс.
call    UpdateDisplay ; Обновляем дисплей.
return

```

```

; Сканируем клавиатуру 4x4 и, если нажата клавиша, передаем ее номер в NewKey.
; Если нет, указатель keyhit обнуляем. Подпрограмма осуществляет также
; защиту от "дребезга контактов". Клавиатура сканируется каждые 20 мс.

```

ScanKeys

```

    btfss Debnce0n      ; Бит Debnce0n равен 1?
    goto  Scan1         ; Если нет, сканируем клавиатуру,
    decfsz Debnce       ; в противном случае продолжаем счет.
    return              ;
    bcf    Debnce0n      ; По окончании сбрасываем флажок.
    return              ; Возвращаемся.

```

Scan1

```

    call    SavePorts   ; Сохраняем значения портов.
    movlw   B'11101111' ; Иницилируем TempD.
    movwf   TempD

```

ScanNext

```

    movf    PORT_B,w     ; Считываем порт для инициализации.
    bcf     INTCON,RBIF   ; Сбрасываем флаг прерывания по изменению порта В.
    rrf     TempD         ; Сдвигаем вправо.
    btfss   STATUS,C      ; Перенос равен 1?
    goto    NoKey         ; Если нет, заканчиваем,
    movf    TempD,w       ; иначе выдаем на порт В.
    movwf   PORT_B       ;
    nop
    btfss   INTCON,RBIF   ; Проверяем флаг прерывания RBIF
    goto    ScanNext      ; Если не установлен, продолжаем.
    btfsc   keyhit        ; Клавиша нажата?
    goto    SKreturn      ; Если нет, выходим.
    bsf     keyhit        ; Устанавливаем флаг нажатия клавиши.
    swapf   PORT_B,w      ; Считываем порт В.
    movwf   TempE         ; Сохраняем в TempE.
    call    GetKeyValue   ; Получить значение клавиши (0 - F).
    movwf   NewKey        ; Сохранить в NewKey.
    bsf     ServKey       ; Установить флаг обслуживания.
    bsf     Debnce0n      ; Установить "антидребезговой" флаг.
    movlw   4             ;
    movwf   Debnce       ; Задаем время ожидания установления колебаний.

```

SKreturn

```

    call    RestorePorts ; Восстанавливаем порты.
    return

```

NoKey

```

    bcf     keyhit        ; Сбрасываем флаг.
    goto    SKreturn

```

; Нумерация клавиш.

	СТОЛБЦЫ			
	1	2	3	4
	(RB3)	(RB2)	(RB1)	(RB0)
СТРОКИ				
1(RB4)	0	1	2	3
2(RB5)	4	5	6	7
3(RB6)	8	9	A	B
4(RB7)	C	D	E	F

GetKeyValue

```

    clrf    TempC
    btfss   TempD,3      ; Первый столбец.
    goto    RowValEnd
    incf    TempC
    btfss   TempD,2      ; Второй столбец.
    goto    RowValEnd
    incf    TempC
    btfss   TempD,1      ; Третий столбец.
    goto    RowValEnd
    incf    TempC        ; Последний столбец.

```

RowValEnd

```

    btfss   TempE,0      ; Верхняя строка?
    goto    GetValCom    ; Если да, получаем 0,1,2 и 3.
    btfss   TempE,1      ; Вторая строка?
    goto    Get4567      ; Если да, получаем 4,5,6 и 7.
    btfss   TempE,2      ; Третий ряд?
    goto    Get89ab      ; Если да, получаем 8,9,a и b.

```

Getcdef

```

    bsf     TempC,2      ; Устанавливаем младшие (msb) биты.

```

Get89ab

```

    bsf     TempC,3      ; /
    goto    GetValCom    ; Выполняем общую часть.

```

Get4567

```

    bsf     TempC,2

```

GetValCom

```

    movf    TempC,w
    addwf   PCL
    retlw   0
    retlw   1
    retlw   2
    retlw   3
    retlw   4
    retlw   5
    retlw   6

```

```

retlw 7
retlw 8
retlw 9
retlw 0a
retlw 0b
retlw 0c
retlw 0d
retlw 0e
retlw 0f

```

```

; Сохранение состояния портов A и B во время сканирования клавиатуры.

```

SavePorts

```

movf PORT_A,w ; Сохраняем состояние порта A
movwf PABuf ; в буфере PABuf.
clrf PORT_A ; Обнуляем порт A.
movf PORT_B,w ; Сохраняем состояние порта B
movwf PBBuf ; в буфере PBBuf.
movlw 0xff ; Посылаем FF в порт B (все единицы).
movwf PORT_B ; /
bsf STATUS,RPO ; Банк 1.
bcf OptionReg,7 ; Разрешаем подключение внутренних
; "подтягивающих" резисторов порта B.

movlw B'11110000' ; Определяем старшие четыре линии порта B как входы,
movwf TRISB ; младшие - как выходы.
bcf STATUS,RPO ; Банк 0.
return

```

```

; Восстановление состояния портов A и B.

```

RestorePorts

```

movf PBBuf,w ; Восстанавливаем состояние порта B.
movwf PORT_B
movf PABuf,w ; Восстанавливаем состояние порта A.
movwf PORT_A
bsf STATUS,RPO ; Банк 1.
bsf OptionReg,7 ; Отключаем "подтягивающие" резисторы.
clrf TRISA ; Делаем все линии портов A и B выходами.
clrf TRISB ; /
bcf STATUS,RPO ; Банк 0.
return

```

UpdateDisplay

```

movf PORT_A,w ; Считываем состояние порта A в регистр w.
clrf PORT_A ; Обнуляем.
andlw 0x0f
movwf TempC ; Помещаем код состояния младших разрядов порта A в tempC.
bsf TempC,4 ; Устанавливаем четвертый разряд в 1 (Lsd).
rrf TempC ; Сдвигаем вправо.

```

```

    btfss STATUS,CARRY ; С равен 1?
    bcf TempC,3 ; Если нет, обнуляем бит 3 (Lsd).
    btfsc TempC,0 ; Проверяем бит, соответствующий старшей цифре (Msd).
    goto UpdateMsd ; Если бит равен 1, обновляем старшую цифру.
    btfsc TempC,1 ; Тоже самое для третьей цифры.
    goto UpdateSrdLsd ; Обновляем.
    btfsc TempC,2 ; Тоже для второй цифры.
    goto Update2ndLsd ; Обновляем.
UpdateLsd
    movf LsdTime,w ; Пересылаем Lsd в регистр w.
    andlw 0x0f ; /
    goto DisplayOut
Update2ndLsd
    swapf LsdTime,w ; Пересылаем вторую цифру в регистр w.
    andlw 0x0f ; Маскируем старшую тетраду.
    goto DisplayOut ; Включить экран.
Update3rdLsd
    movf MsdTime,w ; Пересылаем 3 цифру в регистр w.
    andlw 0x0f ; Маскируем старшую тетраду.
    goto DisplayOut ; Включаем экран.
UpdateMsd
    swapf MsdTime,w ; Переслать старшую цифру (Msd) в регистр w.
    andlw 0x0f ; Маскируем старшую тетраду.
DisplayOut
    call LedTable ; Получаем семисегментный код.
    movwf PORT_B ; Подаем код на светодиоды.
    movf TempC,w ; Включаем цифру.
    movwf PORT_A
    return
LedTable
    addwf PCL ; Модифицируем программный счетчик
    retlw B'00111111' ; Семисегментный код цифры 0.
    retlw B'0000110' ; Семисегментный код цифры 1.
    retlw B'01011011' ; Семисегментный код цифры 2.
    retlw B'01001111' ; Семисегментный код цифры 3.
    retlw B'01100110' ; Семисегментный код цифры 4.
    retlw B'01101101' ; Семисегментный код цифры 5.
    retlw B'01111101' ; Семисегментный код цифры 6.
    retlw B'00000111' ; Семисегментный код цифры 7.
    retlw B'01111111' ; Семисегментный код цифры 8.
    retlw B'01100111' ; Семисегментный код цифры 9.
    retlw B'01110111' ; Семисегментный код цифры A.
    retlw B'01111100' ; Семисегментный код цифры B.
    retlw B'00111001' ; Семисегментный код цифры C.
    retlw B'01011110' ; Семисегментный код цифры D.
    retlw B'01111001' ; Семисегментный код цифры E.
    retlw B'01110001' ; Семисегментный код цифры F.
InitAd

```

```

    movlw B'11000000' ; Тактирование АЦП от внутреннего
    movwf ADCON0       ; RC-генератора.
    return

SampleAd
    call SavePorts
    call DoAd           ; Выполняем цикл преобразования.

AdDone
    btfsc ADCON0,GO     ; Преобразование закончено?
    goto AdDone         ; Если нет, ожидаем.
    bsf   AD0ver        ; Устанавливаем флаг завершения преобразования.
    call  RestorePorts  ; Восстанавливаем порты.
    return

DoAd
    clrf PORT_B         ; Выключаем светодиоды.
    bsf   STATUS,RPO    ; Банк 1.
    movlw 0x0f          ; Переводим выходы порта В в третье состояние
                          ; (высокоимпедансное).
    movwf TRISA         ; /
    bcf   STATUS,RPO    ; Банк 0.
    bsf   ADCON0,ADON   ; Включаем АЦП.
    movlw .125
    call Wait           ;
    bsf   ADCON0,GO     ; Начинаем преобразование.
    return

Wait
    movwf TempC         ; Пересылаем в регистр temp.

Next
    decfsz TempC
    goto Next
    return
;
count equ 26
temp  equ 27
H_byte equ 20
L_byte equ 21
R0 equ 22 ; Назначаем ячейки ОЗУ (RAM).
R1 equ 23
R2 equ 24
B2_BCD bcf STATUS,0 ; Обнуляем бит переноса.
    movlw .16
    movwf count
    clrf R0
    clrf R1
    clrf R2
loop16 rlf L_byte
    rlf H_byte
    rlf R2
    rlf R1

```

```

        rlf R0
        decfsz count
        goto adjDEC
        RETLW 0
adjDEC  movlw R2
        movwf FSR
        call adjBCD
        movlw R1
        movwf FSR
        call adjBCD
        movlw R0
        movwf FSR
        call adjBCD
        goto loop16
adjBCD  movlw 3
        addwf 0,W
        movwf temp
        btfsc temp,3      ; Проверяем, будет ли результат больше семи.
        movlw 0
        movlw 30
        addwf 0,W
        movwf temp
        btfsc temp,7      ; Проверяем, будет ли результат больше семи.
        movwf 0           ; Сохраняем как MSD.
        retlw 0
        end

```

МИКРОКОМПЬЮТЕР, ПРОГРАММИРУЕМЫЙ НА BASIC

Последний из рассматриваемых примеров является не чем иным, как реализацией с помощью PIC-микроконтроллера 16C5X настоящего микрокомпьютера, программируемого на Basic. Конечно, речь идет ни о QBasic, ни о Visual Basic, но используемый в микрокомпьютере Stamp язык имеет хорошую производительность и позволяет достичь высокой скорости работы различных устройств. Учитывая широкие возможности микрокомпьютера Stamp и все возрастающий интерес к нему, мы решили посвятить этому устройству отдельную главу.

Г Л А В А 6

МИКРОКОНТРОЛЛЕР STAMP

В этой главе:

Общие положения

Система разработки

Язык программирования PBasic

Примеры применений

Stamp – это PIC-микроконтроллер 16C56, в программном обеспечении которого имеется интерпретатор языка Basic. Данный микроконтроллер поставляется в виде отдельных микросхем или в составе платы, являющейся по сути средством разработки и отладки приложений. В любом случае это решение представляет интерес для тех, кто хотел бы создавать собственные приложения с PIC-микроконтроллерами.

Хотя Stamp не обеспечивает столь высокой скорости выполнения программ, как обычные PIC-микроконтроллеры, программы которых пишутся на ассемблере, но во многих устройствах, где быстрое действие не является первостепенным критерием, он отлично себя зарекомендовал.

Низкая стоимость Stamp и средств его разработки, а также крайняя простота применения – вот вполне достаточные аргументы в пользу основательного знакомства с ним.

ОБЩИЕ ПОЛОЖЕНИЯ

Микроконтроллер Stamp предложен американской компанией Parallax. Он выпускается в двух вариантах: микросхема, на основе которой можно создавать различные устройства, или маленькая печатная плата с соответствующим программным обеспечением для связи с IBM совместимым ПК.

Опыт подсказывает, что лучше приобретать второй (полный) вариант Stamp, чтобы использовать его в качестве устройства для разработки и тестирования, а затем уже покупать единичные микросхемы для конкретных приложений.

Мини-система разработки содержит следующие элементы:

- ◆ собственно плату Stamp с PIC-микроконтроллером 16C56 и минимальной периферией, необходимой для функционирования вашего устройства и поддержки интерфейса с IBM совместимым ПК;
- ◆ кабель подключения Stamp к параллельному порту ПК;
- ◆ дискету с программным обеспечением, содержащую специальный редактор и программу для связи со Stamp;
- ◆ учебник на французском языке и описание нескольких примеров на английском языке.

Источник питания в комплект не входит, поскольку для этой цели годится обычная батарейка на 9 В. Если вы планируете интенсивно

использовать Stamp, то лучше подключить сетевой адаптер, обеспечивающий 9 В при токе в несколько десятков мА.

Микроконтроллер 16C56 PBasic – это не что иное, как PIC 16C56 XT, поставляемый с запрограммированным интерпретатором PBasic. Он полностью соответствует микроконтроллеру, установленному на плате Stamp, что обеспечивает успешное функционирование разработанных вами устройств с 16C56 PBasic, отлаживаемых посредством платы Stamp.

СИСТЕМА РАЗРАБОТКИ

Плата Stamp может быть использована вами либо для отладки приложений на базе микроконтроллера 16C56 PBasic, либо для непосредственной реализации на ее базе необходимых приложений, тем более что габариты платы невелики и она может быть легко адаптирована для выполнения различных функций.

Схема платы Stamp

Схема платы Stamp крайне проста (рис. 6.1). Помимо PIC 16C56 она содержит минимум компонентов, которые потребуются для любого разрабатываемого вами устройства. Тактирование работы микроконтроллера осуществляется от встроенного тактового генератора с кварцевым резонатором.

В схеме Stamp не предусмотрено никакого специального устройства, которое обеспечило бы выключение устройства в случае неожиданного понижения напряжения питания. Вывод MCLR просто соединен с положительным выводом питания через резистор. В результате, если плата Stamp будет выключена, повторное включение можно произвести не раньше чем через 0,25 секунд, чтобы конденсатор в цепи стабилизатора успел разрядиться. Поэтому советуем использовать в разрабатываемых устройствах схему, изображенную на рис. 6.2, поскольку она обеспечивает устойчивый «аварийный» сброс.

В сопроводительной документации к микроконтроллеру ничего не сказано о ручном сбросе, но вы можете самостоятельно добавить эту функцию в свое приложение (см. схему на рис. 1.2). Ручной сброс необходим на тот случай, если программа в ходе отладки «зависнет» от неизвестной ошибки.

Питание микроконтроллера должно осуществляться напряжением 3–5 В. Для этого на плате установлен стабилизатор типа LM2936, имеющий малое падение напряжения.

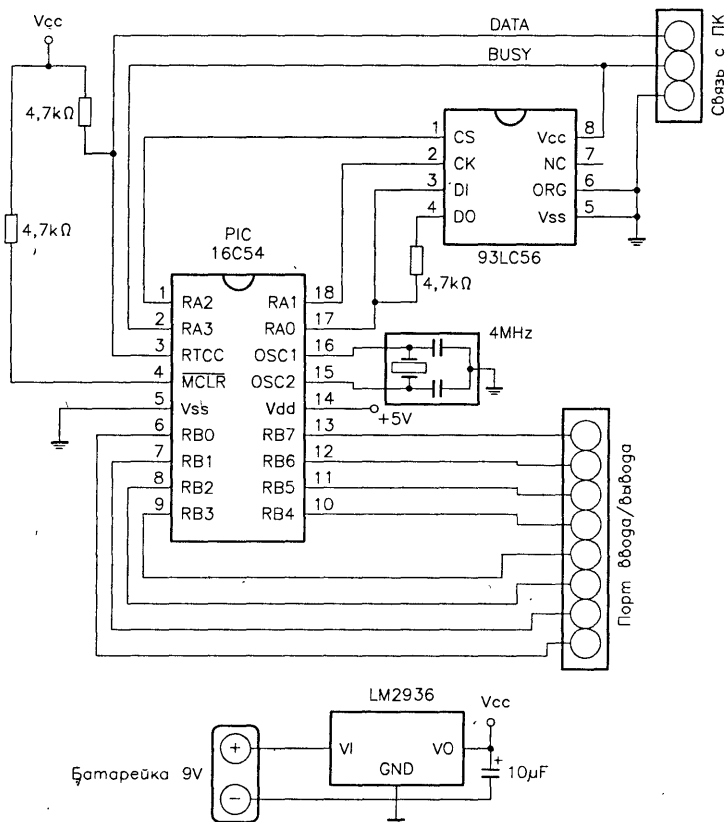
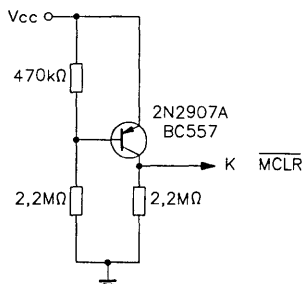


Рис. 6.1

Схема платы Stamp

Рис. 6.2

Рекомендуемая схема сброса при падении напряжения питания



К микроконтроллеру подключено энергонезависимое запоминающее устройство Serial EEPROM с последовательным доступом. Его достоинство – минимальное количество требуемых линий связи, недостаток – усложнение интерпретатора PBasic, обеспечивающего взаимодействие с ним. Именно это запоминающее устройство будет хранить ваши программы на PBasic. Так как его емкость ограничена 256 байтами, программы должны быть компактными, но, как вы увидите на примерах, этого объема вполне достаточно.

Наконец, две линии порта А служат для связи с IBM совместимым ПК. Взаимодействие осуществляется через параллельный порт принтера ПК, уровни линий которого соответствуют сигналам TTL.

Внешний вид печатной платы Stamp показан на рис. 6.3. Это двухсторонняя печатная плата с металлизированными отверстиями, которая помимо компонентов схемы, изображенной на рис. 6.1, имеет свободную контактную зону, где вы сможете смонтировать в случае необходимости свои узлы.

На плате установлены два типовых контакта для подключения батарейки 9 В типа 6F22 и трехконтактный разъем (штыри с шагом 2,54 мм) для соединения с ПК через кабель.

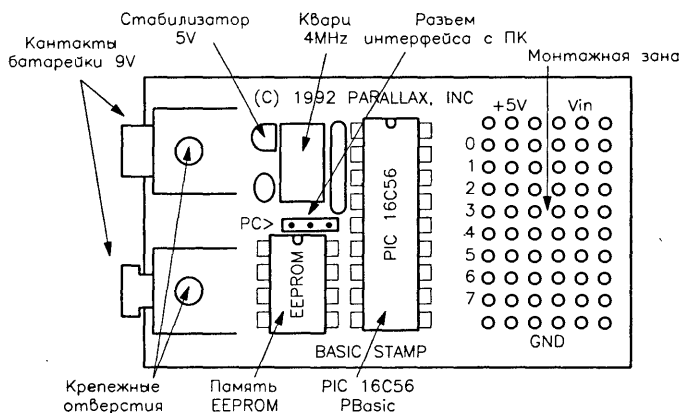


Рис. 6.3

Печатная плата микроконтроллера Stamp с выводами

Программная среда разработки

Программная среда разработки для ПК включает полноэкранный текстовый редактор и программу связи, позволяющую загружать в память микроконтроллера Stamp написанные вами программы.

Редактор имеет все обычные для таких программ функции, но его интерфейс предельно упрощен. Нет управления мышью и развертывающихся меню, зато активно используются функциональные клавиши. Работа с редактором требует небольшого навыка, особенно вначале, но с помощью учебника вы быстро его приобретете. Редактор позволяет сохранять программы на диске и загружать их для использования или отладки.

Программу можно использовать практически с любым ПК. Необходимо только, чтобы он имел ОЗУ не менее 128 Кб, флоппи-диск-вод и параллельный порт. Что касается монитора, то супер-VGA не требуется, так как редактор работает исключительно в текстовом режиме.

ЯЗЫК ПРОГРАММИРОВАНИЯ PBasic

Для программирования функций микроконтроллера Stamp используется язык PBasic. Подробную информацию о структуре и возможностях этого языка вы найдете в инструкции по его применению, здесь же описаны только наиболее часто используемые операторы и их особенности. Хотя программа рассматриваемого интерпретатора языка Basic занимает не более одного килобайта двенадцатирядных слов памяти, он обладает достаточно широкими возможностями.

Во-первых, константы могут быть выражены в десятичном, шестнадцатеричном, двоичном и ASCII кодах. Используемый синтаксис похож на синтаксис ассемблера.

Во-вторых, PBasic поддерживает символические имена (идентификаторы) как для адресов, так и для данных. Таким образом, допускается вместо невыразительного GOTO 100 записать, например, GOTO Circle.

Эта возможность, позволяющая создавать вполне читаемые программы, особенно необходима потому, что PBasic не использует нумерацию строк.

Комментарии могут быть включены в программу с обычным REM; несколько инструкций допускается записывать на одной линии, разделяя двоеточием (:).

Арифметические и логические операции представлены в этом языке достаточно полно (табл. 6.1). Но все же есть область, в которой возможности PBasic ограничены. Микроконтроллер Stamp поддерживает только целую арифметику, что вполне логично, если вспомнить, сколько строк программы требуется для обработки чисел с плавающей запятой (см. листинги в главе 4).

Таблица 6.1

Арифметические и логические операторы языка PBasic

Оператор	Функция
+	Сложение
-	Вычитание
*	Умножение
/	Деление
//	Деление (остаток)
MIN	Переменная \geq значения
MAX	Переменная \leq значения
&	Логическое И
	Логическое ИЛИ
^	Исключающее ИЛИ
&/	Логическое И-НЕ
/	Логическое ИЛИ-НЕ
^/	Исключающее ИЛИ-НЕ

В PBasic предусмотрено несколько предопределенных имен, например, таких как PinX, с помощью которого указываются линии X какого-нибудь порта, и PortX – для указания порта X. PBasic поддерживает также имя DirX для определения назначения линий портов. Все это позволяет значительно сократить программу.

Некоторые из приведенных операторов являются стандартными операторами языка Basic:

- ◆ IF-THEN – условный оператор;
- ◆ GOTO – безусловный переход;
- ◆ GOSUB – обращение к подпрограмме (в одну программу можно помещать до шестнадцати таких обращений);
- ◆ RETURN – возврат из подпрограммы;
- ◆ FOR-NEXT – оператор цикла (вариант с назначаемым шагом STEP также поддерживается);
- ◆ LET – факультативная инструкция, так как можно записать LET A = 2 или A = 2;
- ◆ RANDOM – генератор псевдослучайных чисел (шестнадцатирядных).

Все это стандартные операторы языка Basic и потому не требуют комментариев. В дополнение к ним имеется ряд специфичных, предназначенных специально для работы с микроконтроллерами.

Операторы управления портами ввода/вывода

К этой группе специальных операторов относятся:

- ◆ OUTPUT – определяет выходной порт;
- ◆ INPUT – определяет входной порт;
- ◆ LOW – устанавливает на выходе низкий уровень;
- ◆ HIGH – устанавливает на выходе высокий уровень;
- ◆ TOGGLE – изменяет состояние выхода (с высокого на низкий или с низкого на высокий);
- ◆ PULSOUT – генерирует импульс с программируемой длительностью (импульс может иметь обе полярности);
- ◆ PULSIN – измеряет длительность импульса, поступившего на вход;
- ◆ REVERSE – изменяет назначение вывода (выход оказывается входом и наоборот);
- ◆ BUTTON – очень мощная команда, обеспечивает считывание клавиши, подключенной к входу (устраняя при этом влияние «дребезга контактов»), автоматическое повторение операции, если клавиша удерживается нажатой, и переход к определенному адресу, когда состояние клавиши устойчивое.

Операторы для управления последовательным вводом/выводом

Вторая группа включает всего два оператора:

- ♦ **SERIN** – позволяет принимать последовательные данные через порт. Допускается получение данных со скоростями 300, 600, 1200 и 2400 бод в формате классической асинхронной последовательности;
- ♦ **SEROUT** – последовательный вывод данных (аналогичен **SERIN**, но используется для передачи).

Управление аналоговыми величинами

Для управления аналоговыми величинами предназначены следующие операторы:

- ♦ **PWM** – позволяет генерировать модулируемые по ширине импульсы, в том числе посредством подключения внешней RC-цепочки генерировать напряжение переменной величины на заданном выходе;
- ♦ **POT** – считывает значение переменного резистора или прибора с изменяемым сопротивлением (фоторезистора, терморезистора и т.п.) в пределах между 5 и 50 кОм. Различные параметры позволяют определять пределы и масштабные коэффициенты для этой команды;
- ♦ **SOUND** – генерирует на выбранном выходе звуковые сигналы, программируемые по частоте и длительности.

Прочие операторы

Помимо вышеперечисленных в языке PBASIC есть и другие операторы:

- ♦ **EEPROM** – позволяет загружать в EEPROM переменные величины, используемые программой до ее загрузки с ПК;
- ♦ **READ** – считывает значения переменных величин;
- ♦ **WRITE** – записывает данные в EEPROM;
- ♦ **PAUSE** – позволяет задать паузу (задержку) от 0 до 65535 мс в ходе выполнения программы;
- ♦ **NAP** – переводит микроконтроллер в sleep-режим на указанное короткое (не более 2,3 с) время. В этом режиме потребление Stamp потребляет ток всего в 20 мкА;

- ◆ SLEEP – аналогичен NAP, но предназначен для больших промежутков времени (до 65535 с);
- ◆ END – позволяет входить в sleep-режим на неопределенный срок; микросхема «просыпается», когда поступает запрос от ПК;
- ◆ DEBUG – позволяет посылать в ПК содержимое переменных величин программы в процессе отладки.

Как видно из приведенного перечня, те операторы Basic, которые используются в программах для ПК, обладающих большой оперативной памятью, в PBasic отсутствуют. Зато есть многочисленные операторы, характерные именно для микроконтроллеров (например, BUTTON, SOUND, POT и др.), что позволяет сократить объем кода вашего приложения.

На рис. 6.4 показан пример, который хорошо иллюстрирует действие этих операторов. Схема считывает значение потенциометра (или любого эквивалентного переменного резистора) и генерирует тем более высокий звук, чем ниже его сопротивление. Для этого нужно только четыре оператора:

- ◆ `boucle: pot 0, 100, b2` – считывает значение сопротивления потенциометра, подключенного к выводу 0, и сохраняет полученное значение в переменной величине `b2`;

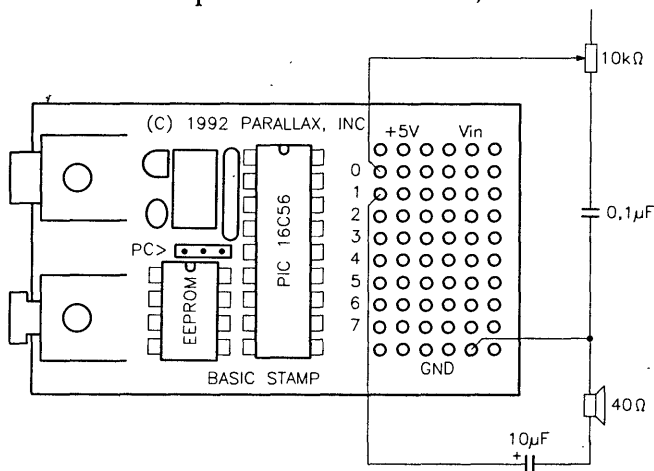


Рис. 6.4

Схема генерации звукового сигнала

- ♦ $b2 = b2 / 2$ – делит $b2$ на 2 для того, чтобы это значение не вышло за пределы от 0 до 128, допустимые для следующего оператора;
- ♦ `sound I, (b2, 10)` – генерирует на выводе I звуковой сигнал, частота которого определяется содержимым $b2$, длительность – значением 10;
- ♦ `goto boucle` – возвращает программу к началу.

Конечно, этот пример не имеет практической пользы, но он хорошо показывает, как операторы PBasic обеспечивают взаимодействие платы Stamp к подключаемым внешним устройствам.

ПРИМЕРЫ ПРИМЕНЕНИЙ

Теперь рассмотрим несколько примеров применения Stamp. Как вы увидите, несмотря на малый размер доступной памяти для хранения программ этот микроконтроллер позволяет реализовывать достаточно сложные алгоритмы.

Три приведенных ниже примера послужат вам основой для создания собственных приложений.

Аналого-цифровое преобразование

Первый пример демонстрирует преобразование аналоговых величин в цифровую форму (восьмиразрядные числа) и их передачу через последовательный интерфейс RS232 на какое-либо устройство.

Учитывая ограниченное количество линий ввода/вывода PIC 16C56 (и платы Stamp), рекомендуется применять аналого-цифровой преобразователь с последовательным интерфейсом. К счастью, такие преобразователи широко распространены, причем тип последовательного интерфейса может быть как стандартным (Microwire, I²C), так и специфичным. Некоторые преобразователи могут практически напрямую подключаться к плате Stamp.

На рис. 6.5 представлена схема, где используется микросхема АЦП типа ADC0831 фирмы National Semiconductor, являющаяся восьмиразрядным аналого-цифровым преобразователем с последовательным интерфейсом типа Microwire. Это тот же преобразователь, на примере которого в главе 3 демонстрировался принцип реализации последовательного интерфейса.



Схема аналого-цифрового преобразователя

Как вы помните, в случае с ADC0831 этот интерфейс использует три линии:

- ◆ линию передачи тактовых импульсов CLK, которая задает ритм передачи данных;
- ◆ линию DO для DATA OUT, которая служит для передачи данных в последовательной форме;
- ◆ линию CS для сигнала Chip Select, которая осуществляет выбор микросхемы и разрешает соединение по шине через линии CLK и DO нескольких микросхем того же типа.

На рис. 6.6 представлены временные диаграммы сигналов этих трех линий. Выбор микросхемы осуществляется подачей низкого уровня на линию CS, вследствие чего отменяется высокоимпедансное состояние на выходе данных DO и начинается процесс преобразования. Формируемые в результате этого данные выдаются на линию DO, начиная со старших разрядов.

Данные достоверны во время положительного фронта тактовых импульсов и изменяются по отрицательному фронту.

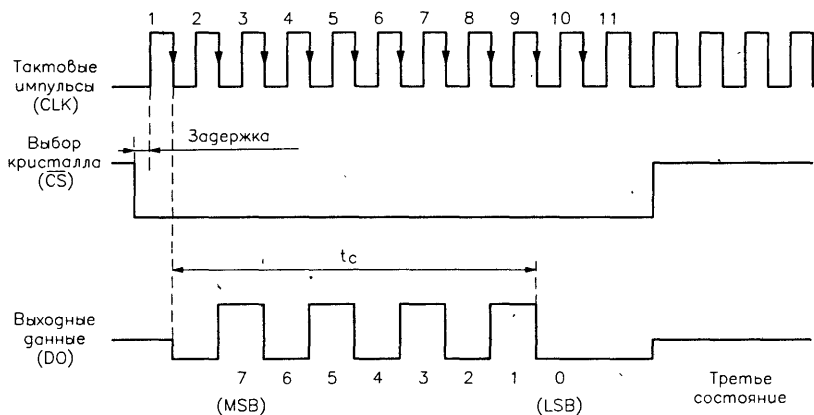


Рис. 6.6

Временные диаграммы сигналов шины Microwire ИС ADC0831

Чтобы упростить схему, линия V_{ref} преобразователя соединена с потенциалом +5 В, а вход $V_{in} (-)$ подключен к нулевому потенциалу. При этом выходной код ADC0831 будет равен 255, когда напряжение, приложенное ко входу $V_{ref} (+)$, равно 5 В. Изменять масштаб преобразования можно, либо изменяя величину опорного напряжения V , либо умножая или деля данные, читаемые программой.

Исходный текст программы представлен в листинге 6.1. Она считывает данные, приходящие из ADC0831, каждые две секунды и посылает их через последовательный асинхронный интерфейс (например, RS232) со скоростью 2400 бит/с на любое внешнее устройство. Главное, чтобы интерфейс был совместим с внешним устройством. Для согласования уровней можно использовать микросхему приемопередатчика типа MAX232 или MAX233.

Листинг 6.1

```

* Программа AD_CONV.BAS.
* Соответствует инструкции по применению Stamp фирмы Parallax.
* Stamp использует ADC0831 для получения кодов аналоговых данных,
* которые посылает затем через интерфейс RS 232.

```

```

Symbol CS      =      0
Symbol AD      =      pin1
Symbol CLK     =      2
Symbol S_out   =      3
Symbol data    =      b0

```

Symbol i = b2

setup: let pins = 255 * На выходах высокий уровень (ADC отключен).
 let dirs = %11111101 * S_out, CLK, CS - выходы; AD - вход.

loop: gosub conv * Преобразование.
 serout S_out, N2400, (#b0, 13, 10) * Передача данных.
 pause 2000 * Ожидание в течение двух секунд.
 goto loop * Бесконечный цикл.

conv: low CLK * Установка 0 на CLK.
 low CS * Выбор ADC.
 pulsout CLK, 1 * Импульс на CLK длительностью в 10 мкс.
 let data = 0 * Сброс данных.
 for i = 1 to 8 * Цикл приема восьми битов данных.
 let data = data * 2 * Сдвиг влево.
 pulsout CLK, 1 * Импульс 10 мкс.
 let data = data + AD * Считывание младшего бита данных.
 next * Следующий бит.
 high CS * Выключение ADC.
 return

Первые строки программы определяют адреса памяти для переменных величин. Идентификаторы pinX используются PBasic для обозначения входов/выходов.

Подпрограмма setup инициализирует микросхему, посылая высокий уровень на все выходы, в том числе и на линию \overline{CS} , что отключает ADC0831. Следующая строка определяет назначение линий портов. Фактически она довольствуется тем, что задает порт 1 как вход, поскольку он соответствует выходу данных DO преобразователя.

Подпрограмма loop – это цикл, который осуществляет передачу данных, полученных подпрограммой conv в результате преобразования. Эти данные (вместе с символами возврата каретки и перевода строки – коды 13 и 10, включенные в инструкцию SEROUT) передаются на выход со скоростью до 2400 бит/с. Эта команда выдает b0 и затем посылает два символа (здесь 13 и 10) с заданной скоростью. Обратите внимание: буква N перед числом 2400 указывает инверсный вывод данных. Поскольку Stamp выдает информацию на линию RS232 непосредственно, без использования специальных передатчиков, осуществляющих инверсию сигналов, то данные должны быть представлены в отрицательной логике. Если вы используете ИС MAX232, которая формирует стандартные уровни интерфейса RS232, необходимо заменить N на T, чтобы выдавать данные без инверсии.

Подпрограмма `conv` наиболее интересна. С ее помощью можно управлять почти любой микросхемой с последовательным интерфейсом, подобным `Microwire`. Сигналы линий `CLK` и \overline{CS} переводятся сначала на низкий уровень оператором `low`, а затем оператор `pulsout` генерирует импульс длительностью 10 мкс на `CLK`. Следующий этап – обнуляется ячейка памяти `data`, в которую будут записываться принимаемые данные, и организуется цикл `for-next`, считывающий восемь бит данных кода `ADC0831`. Прием данных сопровождается выдачей импульсов на линию `CLK`, осуществляемой оператором `pulsout`.

Сдвиг данных влево выполняется простым умножением на два.

Отметим, что программа заняла не более четверти программной памяти платы `Stamp`. Оставшиеся три четверти могут быть использованы для других программ.

Управление шаговыми двигателями

Ввиду чрезвычайной легкости, с которой можно задавать позицию шагового двигателя, они получают все большее распространение. Их применяют в принтерах для подачи бумаги, в дисководах для перемещения считывающих головок и в других аппаратах, где необходимо точное позиционирование.

В отличие от классических двигателей, которые начинают вращаться, как только их подключают к источнику тока, шаговые двигатели, чтобы поворачиваться, должны получить последовательность упорядоченных импульсов на свои обмотки. Кроме того, они вращаются не непрерывно, а скачками (шагами) на единицу углового перемещения, размер которого зависит от типа двигателя. Обычно он составляет от $7,5^\circ$ до $1,8^\circ$.

Если во время работы предельная скорость двигателя не была превышена, точность положения гарантируется, достаточно лишь определить, сколько импульсов послать двигателю, чтобы повернуть его на нужный угол.

Из шаговых двигателей наиболее распространены и просты в управлении униполярные модели с четырьмя катушками. Униполярными их называют потому, что при управлении на их обмотки достаточно подать или не подать ток, в то время как в биполярном двигателе надо менять полярность напряжения на выводах.

Чтобы заставить вращаться такой двигатель, надо подавать на него последовательность импульсов, указанную в табл. 6.2. Если эта последовательность прямая (от шага 1 к шагу 4), двигатель вращается

в одном направлении, если обратная (от шага 4 к шагу 1), то в противоположном.

Таблица 6.2

Последовательность управления униполярным шаговым двигателем

	Шаг			
	1	2	3	4
Обмотка 1	+V	+V	0	0
Обмотка 2	0	0	+V	+V
Обмотка 3	+V	0	0	+V
Обмотка 4	0	+V	+V	0

Сформировать такие последовательности управляющих сигналов несложно, но для этого требуется несколько микросхем стандартной логики. Для управления двигателями были выпущены специализированные интегральные схемы. Наиболее известна из них SAA1027, принцип работы которой показан на рис. 6.7. Чтобы управлять

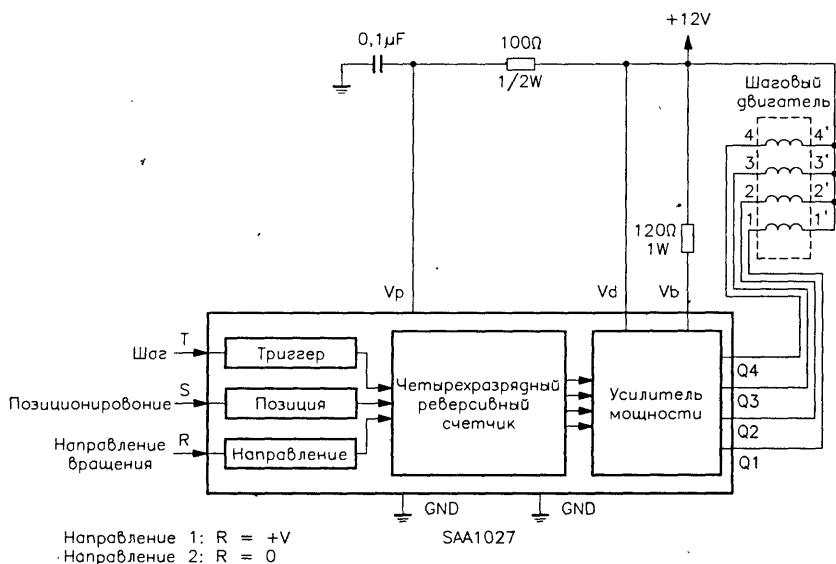


Рис. 6.7

Пример использования SAA1027

C	equ	0h	
DCARRY	equ	1h	
DC	equ	1h	
Z_bit	equ	2h	; Признак 0 - бит 2 регистра 3 (F3).
Z	equ	2h	
PJDOW	equ	3h	
PD	equ	3h	
T_out	equ	4h	
TO	equ	4h	
PA0	equ	5h	; PA0, PA1, PA2 - биты состояния PIC16C5X.
PA1	equ	6h	
PA2	equ	7h	
;			
RP0	equ	5h	; RP0, RP1, IRP - биты состояния PIC16C5X.
RP1	equ	6h	;
IRP	equ	7h	;
GIE	equ	7h	; Биты регистра INTCON PIC16C71.
ADIE	equ	6h	;
RTIE	equ	5h	;
INTE	equ	4h	;
RBIE	equ	3h	;
RTIF	equ	2h	;
INTF	equ	1h	;
RBIF	equ	0	;
ADCS1	equ	7h	; Биты регистра ADCN0 PIC16C71.
ADCS0	equ	6h	;
CHS1	equ	4h	;
CHS0	equ	3h	;
GO	equ	2h	;
ADIF	equ	1h	;
ADON	equ	0	;
PCFG1	equ	1h	; Биты регистра ADCN1 PIC16C71.
PCFG0	equ	0	;
;			
Same	equ	1h	
LSB	equ	0h	
MSB	equ	7h	
;			
TRUE	equ	1h	
YES	equ	1h	
FALSE	equ	0h	
NO	equ	0h	
;			

Кристиан Тавернье

РІС-микроконтроллеры

Практика применения

Справочник

Главный редактор *Захаров И. М.*
editor-in-chief@dmkpress.ru

Перевод *Марченко В. А.*

Научный редактор *Корзинкин В. С.*

Выпускающий редактор *Виноградова Н. В.*

Технический редактор *Прока С. В.*

Дизайн обложки *Панкусова Е. Н.*

Верстка *Захарова Е. П.*

Графика *Шаклунов А. К.*

ИД № 01903 от 30.05.2000

Подписано в печать 21.01.2002. Формат 60×88¹/₁₆.
Гарнитура «Баскервиль». Печать офсетная.
Усл. печ. л. 16,66. Тираж 3000 экз. Зак. № 86

Издательство «ДМК Пресс»,
105023, Москва, пл. Журавлева, д. 2/8.
Электронные адреса: www.dmkpress.ru, info@dmk.ru

Отпечатано в типографии №9,
Волочаевская, 40.