Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра электронной техники и технологии

ПРОГРАММНО-УПРАВЛЯЕМЫЕ МИКРОКОНТРОЛЛЕРНЫЕ УСТРОЙСТВА. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники для специальностей 1-36 04 01 «Программно-управляемые электронно-оптические системы», 1-39 02 02 «Проектирование и производство программно-управляемых электронных средств», 1-39 02 03 «Медицинская электроника» в качестве пособия

Минск БГУИР 2017

УДК 004.31-022.53(076.5)
ББК 32.971.32-04я73
П78

А в т о р ы:
М. В. Давыдов, С. В. Кракасевич, Н. С. Давыдова, М. М. Меженная

Р е ц е н з е н т ы:
кафедра технологий программирования Белорусского государственного университета (протокол №10 от 07.04.2016);

доцент кафедры конструирования и производства приборов Белорусского национального технического университета, кандидат технических наук, доцент Е. Г. Зайцева

Программно-управляемые микроконтроллерные устройства. Лабо-
П78 раторный практикум : пособие / М. В. Давыдов [и др.]. – Минск : БГУИР, 2017. – 68 с. : ил.
ISBN 978-985-543-277-8.

Рассматриваются основы разработки программного обеспечения для микроконтроллеров на языке программирования С. Представлено шесть лабораторных работ по изучению внутренней структуры микропроцессоров и особенностей программирования на языке С, генерации аналоговых сигналов посредством цифроаналогового преобразователя, реализации широтно-импульсной модуляции с помощью программируемого массива счетчиков. Также изучается работа с таймерами-счетчиками микроконтроллера и с универсальным асинхронным приемопередатчиком UART.

Предназначено для закрепления и углубления теоретических знаний, приобретения практических навыков работы в области микропроцессорного управления и создания программ для микропроцессорной техники.

УДК 004.31-022.53(076.5)
ББК 32.971.32-04я73

ISBN 978-985-543-277-8

СОДЕРЖАНИЕ

Лабораторная работа №1

Изучение внутренней структуры микропроцессора и особенностей программирования на языке С

1.1 Цель работы

Изучить структурную организацию микроконтроллера (МК) Cygnal C8051F320, разработать программу для микроконтроллера Cygnal C8051F320 на языке программирования С для генерации прямоугольных сигналов с заданной скважностью и частотой.

1.2 Теоретические сведения

Семейство микроконтроллеров C8051Fxxx оптимально подходит для построения устройств, требующих высокой производительности, точности измерений, большой степени интеграции и малого потребления. Они программно совместимы со стандартом 8051, но одновременно имеют высокую производительность – **до 100 MIPS**.

Энергонезависимая FLASH-память программ может программироваться «в системе», т. е. на плате. В сектора по 512 байт FLASH-памяти (размер FLASH до 128 Кбайт) могут записываться как программы, так и данные, которые становятся таким образом энергонезависимыми.

Микроконтроллеры имеют конструктивно встроенные интерфейсы: CAN-контроллер (серия F040 и F060), USB (серия F320), SMBus/I2C, UART, SPI. Имеется порт с повышенной нагрузочной способностью, 8-, 10-, 12-, 16-, 24-битные аналого-цифровые преобразователи (АЦП) и 12-битные цифроаналоговые преобразователи (ЦАП). В моделях серии C8051F35X имеются сигма-дельта АЦП на 24 и 16 бит. Встроенная автономная отладочная система (JTAG) – полный внутрисхемный эмулятор «in-circuit» не задействует ресурсы кристалла и позволяет проверять и модифицировать память и регистры, устанавливать контрольные и временные точки, выполнять пошаговое выполнение и остановку программы.

1.2.1 Краткий обзор микроконтроллера Cygnal семейства C8051F320

МК C8051F320 имеет встроенную схему сброса по включению питания, схему слежения за напряжением питания, регулятор напряжения, сторожевой таймер, тактовый генератор и представляет собой, таким образом, функционально-законченную систему на кристалле. Имеется возможность внутрисхемного программирования FLASH-памяти, что обеспечивает долговременное (энергонезависимое) хранение данных, а также позволяет осуществлять обновление программного обеспечения в готовых изделиях. Программа пользователя может полностью управлять всеми периферийными модулями, а также может индивидуально отключить любой из них с целью уменьшения энергопотребле-

ния. Встроенный двухпроводной Silicon Labs Development Interface (интерфейс C2) позволяет производить «неразрушающую» (не используются внутренние ресурсы) внутрисхемную отладку в режиме реального времени, используя МК, установленный в конечное изделие. Средства отладки обеспечивают проверку и модификацию памяти и регистров, расстановку точек остановок, пошаговое выполнение программы, а также поддерживают команды запуска и остановки. В процессе отладки с использованием интерфейса C2 все аналоговые и цифровые периферийные модули полностью сохраняют свою работоспособность. Два вывода интерфейса C2 могут использоваться для других пользовательских функций, что позволяет осуществлять внутрисистемную отладку, не занимая для этого отдельные выводы корпуса. Структурная схема МК C8051F320 изображена на рисунке 1.1.

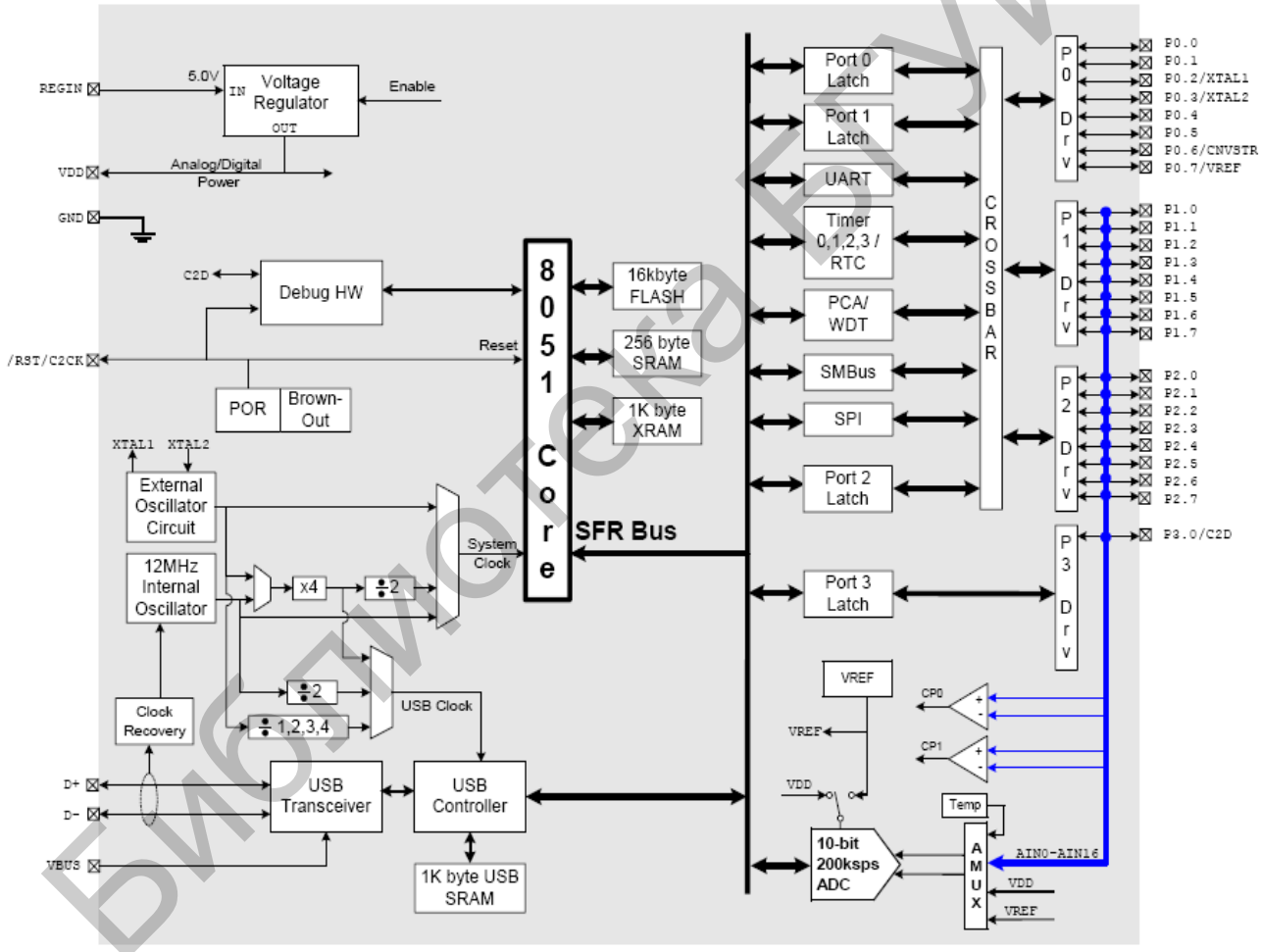

Рисунок 1.1 – Структурная схема C8051F320

Каждый МК предназначен для работы в промышленном температурном диапазоне (от минус 40 до +85 ºС) при напряжении питания 2,7…3,6 В (для взаимодействия по шине USB требуется напряжение 3,0…3,6 В). На порты ввода/вывода и вывод/RST могут быть поданы входные сигналы напряжением

до 5 В. МК C8051F320/1 выпускаются в 32-выводных корпусах типа LQFP и 28-выводных корпусах типа MLP.

Процессорное ядро CIP-51 по системе команд полностью совместимо с ядром MCS-51. Для разработки программного обеспечения могут использоваться стандартные 803x/805x ассемблеры и компиляторы. Процессорное ядро CIP-51 использует конвейерную архитектуру, что существенно повышает скорость выполнения команд по сравнению со стандартной архитектурой 8051. В МК с архитектурой 8051 все команды, кроме MUL и DIV, исполняются за 12 или 24 системных тактовых цикла при максимальной тактовой частоте 12…24 МГц. При работе на тактовой частоте 25 МГц производительность ядра CIP-51 может достигать 25 MIPS.

Ядро имеет стандартную (8051) структуру адресного пространства памяти программ и данных. В состав памяти входит ОЗУ объемом 256 байт, старшие 128 байт которого имеют двойную конфигурацию. В режиме косвенной адресации осуществляется доступ к старшим 128 байтам ОЗУ общего назначения, а в режиме прямой адресации – доступ к 128 байтам адресного пространства регистров специального назначения (SFR). Младшие 128 байт ОЗУ доступны как для прямой, так и для косвенной адресации. Из них первые 32 байта адресуются как четыре банка регистров общего назначения, а следующие 16 байт – побайтно или побитно. Память программ МК состоит из 16 Кбайт FLASH-памяти. Эта память может перепрограммироваться внутрисистемно секторами по 512 байт, не требуя при этом специального внешнего напряжения программирования. На рисунке 1.2 приведена карта распределения памяти МК.

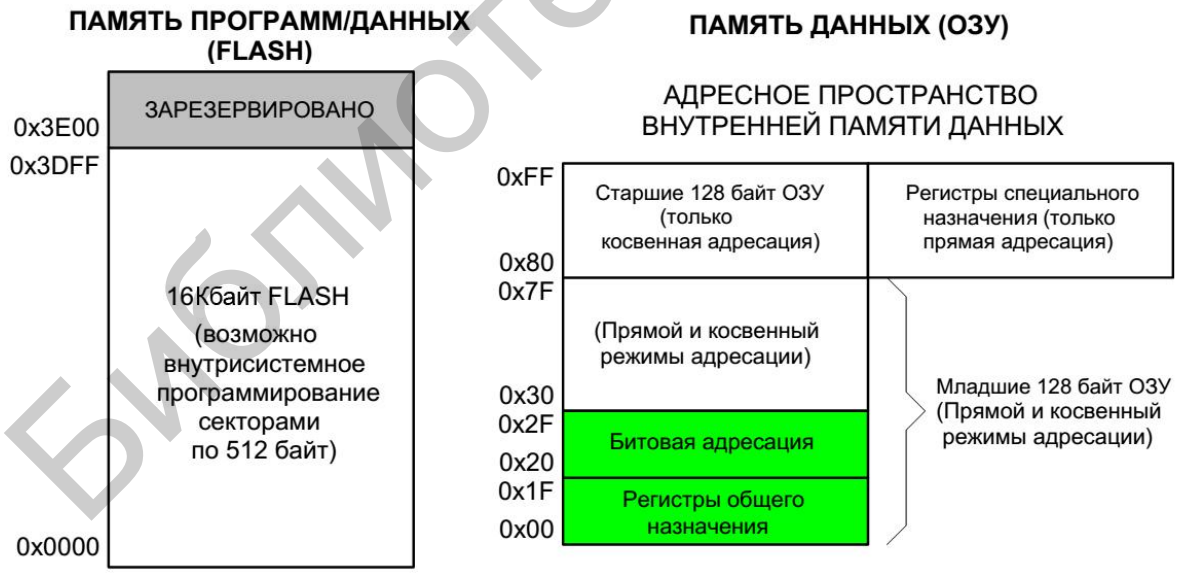

Рисунок 1.2 – Карта распределения памяти МК

Программируемые цифровые порты ввода/вывода и матрица соединений

МК C8051F320 имеет 25 выводов, которые составляют три 8-разрядных порта и один 1-разрядный порт. Порты функционируют в соответствии со стандартом 8051 с некоторыми дополнительными возможностями. Каждый вывод порта можно настроить как аналоговый вход или как цифровой вход/выход. Выводы, настроенные как цифровые входы-выходы, можно, кроме этого, настроить как 2-тактные цифровые выходы или выходы с открытым стоком. Также допускается глобальное отключение слаботоковых подтягивающих резисторов, что позволяет еще более снизить энергопотребление.

Цифровая матрица позволяет необходимым образом соединять внутренние цифровые системные ресурсы с выводами портов ввода/вывода. При помощи регистров управления матрицей на выводы портов могут быть выведены сигналы от внутренних таймеров-счетчиков и последовательных интерфейсов, аппаратные прерывания, выходы компараторов и другие цифровые сигналы. Это позволяет выбрать точную комбинацию связей между портами ввода/вывода общего назначения и цифровыми ресурсами, необходимую для каждого конкретного приложения.

1.2.2 Обработка прерываний в микроконтроллерах Cygnal семейства C8051F320

Процессорное ядро CIP-51 имеет развитую систему прерываний, поддерживающую в общей сложности 16 источников прерываний с двумя уровнями приоритета. Распределение источников прерываний между встроенными периферийными модулями и внешними входными выводами зависит от конкретного типа МК. Каждый источник прерываний имеет один или несколько связанных с ним флагов прерываний, размещенных в SFR. Когда периферийный модуль или внешний источник прерываний регистрирует событие, удовлетворяющее условию прерывания, соответствующий флаг прерывания устанавливается в 1.

Если прерывание от источника прерываний разрешено, то при установке флага прерывания генерируется запрос прерывания. Как только выполнение текущей команды завершится, будет сгенерирована команда LCALL перехода по предопределенному адресу, откуда начнется исполнение процедуры обслуживания прерывания (Interrupt Service Routine – ISR). Каждая ISR должна заканчиваться командой RETI, которая возвращает управление прерванной программе и приводит к выполнению той команды, которая исполнилась бы, если бы запроса прерывания не было. Если прерывания не разрешены, флаг прерывания игнорируется и выполнение программы продолжается в нормальном режиме (флаг прерывания устанавливается в 1, независимо от того, разрешены прерывания или запрещены).

Прерывание от каждого источника прерываний может быть разрешено или запрещено с помощью соответствующих битов разрешения прерываний в регистрах SFR (IE-EIE2). Однако сначала прерывания необходимо разрешить гло-

бально установкой в 1 бита EA (IE.7), только после этого состояние индивидуальных флагов разрешения прерываний будет иметь силу. Сброс в 0 бита EA запрещает прерывания от всех источников прерываний, независимо от состояния индивидуальных флагов разрешения прерываний.

Некоторые флаги прерываний сбрасываются автоматически аппаратными средствами при переходе к процедуре ISR. Однако большинство флагов прерываний не сбрасываются аппаратно и должны быть сброшены программно до возвращения из процедуры ISR. Если флаг прерывания остается установленным после завершения выполнения команды возврата из прерывания (RETI), то сразу же будет сгенерирован новый запрос прерывания и после завершения выполнения следующей команды произойдет повторный переход к процедуре ISR.

Данное семейство МК поддерживает 16 ***источников прерываний***. Программа может симулировать прерывание установкой в 1 любого флага прерывания. Если прерывание для этого флага разрешено, будет сгенерирован запрос прерывания и произойдет переход по адресу процедуры ISR, связанной с этим флагом прерывания. Источники прерываний МК, соответствующие им адреса прерываний, уровень приоритета и биты управления перечислены в таблице 1.1.

Задержка обработки прерывания

Время реакции на прерывание зависит от состояния процессорного ядра в момент возникновения прерывания. Опрос флага прерывания и декодирование приоритета осуществляется каждый системный тактовый цикл. Поэтому наименее возможное время реакции на прерывание составляет пять тактовых циклов: один – для определения прерывания и четыре – для выполнения команды LCALL перехода к процедуре ISR. Если в момент выполнения команды RETI появляется прерывание, то до выполнения команды LCALL перехода на процедуру обслуживания этого прерывания будет исполнена одна команда основной программы. Поэтому максимальное время реакции на прерывание (если в настоящий момент не обслуживается другое прерывание или если новое прерывание имеет более высокий приоритет) будет тогда, когда выполняется команда RETI, а следом за ней должна выполняться команда DIV. В этом случае время реакции составляет 18 тактовых циклов: один – для определения прерывания, пять – для выполнения команды RETI, восемь – для выполнения команды DIV и четыре – для выполнения команды LCALL перехода на процедуру ISR. Если выполняется процедура ISR для прерывания с равным или более высоким приоритетом, новое прерывание не будет обслужено до тех пор, пока не завершится текущая процедура ISR, включая команду RETI и следующую команду.

Таблица 1.1 – Источники прерываний

Источник прерывания	**Вектор прерывания**	**Приоритет**	**Флаг прерывания**	**Битовая адресация**	**Аппаратный сброс**	**Бит разрешения**	**Управление приоритетом**
Сброс	0x0000	max	Нет	N/A	N/A	Разрешен всегда	Всегда наивысший
Внешнее прерывание 0 (/INT0)	0x0003	0	IE0 (TCON.1)	Y	Y	EX0 (IE.0)	PX0 (IP.0)
Переполнение Таймера 0	0x000B	1	TF0 (TCON.5)	Y	Y	ET0 (IE.1)	PT0 (IP.1)
Внешнее прерывание 1 (/INT1)	0x0013	2	IE1 (TCON.3)	Y	Y	EX1 (IE.2)	PX1 (IP.2)
Переполнение Таймера 1	0x001B	3	TF1 (TCON.7)	Y	Y	ET1 (IE.3)	PT1 (IP.3)
Последовательный порт УАПП0	0x0023	4	RI0 (SCON0.0)	Y	N	ES0 (IE.4)	PS0 (IP.4)
Последовательный порт УАПП0	0x0023	4	TI0 (SCON0.1)	Y	N	ES0 (IE.4)	PS0 (IP.4)
Переполнение Таймера 2	0x002B	5	TF2H (TMR2CN.7) TF2L (TMR2CN.6)	Y	N	ET2 (IE.5)	PT2 (IP.5)
Модуль SPI0	0x0033	6	SPIF (SPI0CN.7) WCOL (SPI0CN.6) MODF (SPI0CN.5) RXOVRN (SPI0CN.4)	Y	N	ESPI0 (IE.6)	PSPI0 (IP.6)
Модуль SMBus0	0x003B	7	SI (SMB0CN.0)	Y	N	ESMB0 (EIE1.0)	PSMB0 (EIP1.0)
Детектор диапазона АЦП0	0x004B	9	ADWINT (ADC0CN.3)	Y	N	EWADC0 (EIE1.2)	PWADC0 (EIP1.2)
Завершение преобразования АЦП0	0x0053	10	ADC0INT (ADC0CN.5)	Y	N	EADC0 (EIE1.3)	PADC0 (EIP1.3)
Переполнение Таймера 3	0x0073	14	TF3H (TMR3CN.7) TF3L (TMR3CN.6)	N	N	ET3 (EIE1.7)	PT3 (EIP1.7)
Уровень VBUS	0x007B	15	N/A	N/A	N/A	EVBUS (EIE2.0)	PVBUS (EIP2.0)

1.2.3 Таймеры-счетчики в микроконтроллерах Cygnal семейства C8051F320

МК содержит четыре таймера-счетчика (ТС): два из них представляют собой 16-разрядные ТС, совместимые с ТС стандартной архитектуры 8051, а другие два являются 16-разрядными ТС с режимом автоперезагрузки и предназначены для использования совместно с АЦП, SMBus, USB (для измерения фреймов), а также в качестве ТС общего назначения. Эти ТС можно использовать для измерения временных интервалов, подсчета внешних событий, а также для генерации периодических запросов прерываний. Таймер 0 и Таймер 1 почти идентичны и имеют четыре основных режима работы. Таймер 2 и Таймер 3 могут работать (каждый) как один 16-разрядный таймер или как два 8-разрядных таймера, причем во всех случаях поддерживается режим автоперезагрузки. Рассмотрим подробно Таймер 0 и Таймер 1.

Режимы Таймера 0 и Таймера 1:

- 13-разрядный ТС;
- 16-разрядный ТС;
- 8-разрядный ТС с автоперезагрузкой;
- два 8-разрядных ТС (только Таймер 0).

Таймеры 0 и 1 могут тактироваться от одного из пяти источников, выбор которых осуществляется с помощью битов выбора режима таймера (T1M–T0M) и битов выбора коэффициента деления тактовой частоты (SCA1–SCA0). Биты выбора коэффициента деления тактовой частоты настраивают предварительный делитель тактовой частоты, сигнал с выхода которого может использоваться для тактирования Таймера 0 и/или Таймера 1.

В качестве сигнала тактирования Таймеров 0 и 1 можно выбрать либо сигнал с выхода предварительного делителя тактовой частоты, либо системный тактовый сигнал. Таймер 0 и Таймер 1 могут также функционировать как счетчики. В этом случае регистр таймера-счетчика инкрементируется под воздействием каждого перехода внешнего сигнала на выбранном входном выводе (T0 или T1) из состояния лог. 1 в состояние лог. 0. Могут подсчитываться импульсы с частотой до 1/4 системной тактовой частоты. Входной сигнал не обязательно должен быть периодическим, однако он должен удерживаться на заданном уровне как минимум в течение двух полных системных тактовых циклов, чтобы гарантировать его корректную выборку.

Каждый таймер реализован в виде 16-разрядного регистра, доступного как два отдельных байта: младший (TL0 или TL1) и старший (TH0 или TH1). Регистр управления ТС (TCON) используется для включения Таймера 0 и Таймера 1, а также для индикации их состояния. Прерывания от Таймера 0 можно включить установкой в 1 бита ET0 в регистре IE. Прерывания от Таймера 1 можно включить установкой в 1 бита ET1 в регистре IE. Оба таймера-счетчика работают в одном из четырех основных режимов, задаваемых битами выбора режима T1M1–T0M0 в регистре режима ТС (TMOD). Каждый ТС может быть настроен независимо от другого.

Рассмотрим подробно Режимы 0, 1, 2 (13-разрядный ТС, 16-разрядный ТС, 8-разрядный ТС с автоперезагрузкой), которые будут использованы при выполнении лабораторной работы.

В *режиме 0* Таймеры 0 и 1 работают как 13-разрядный таймер-счетчик. Далее приводится описание настройки и функционирования Таймера 0 (рисунок 1.3). Однако оба таймера идентичны и Таймер 1 настраивается точно так же, как и Таймер 0.

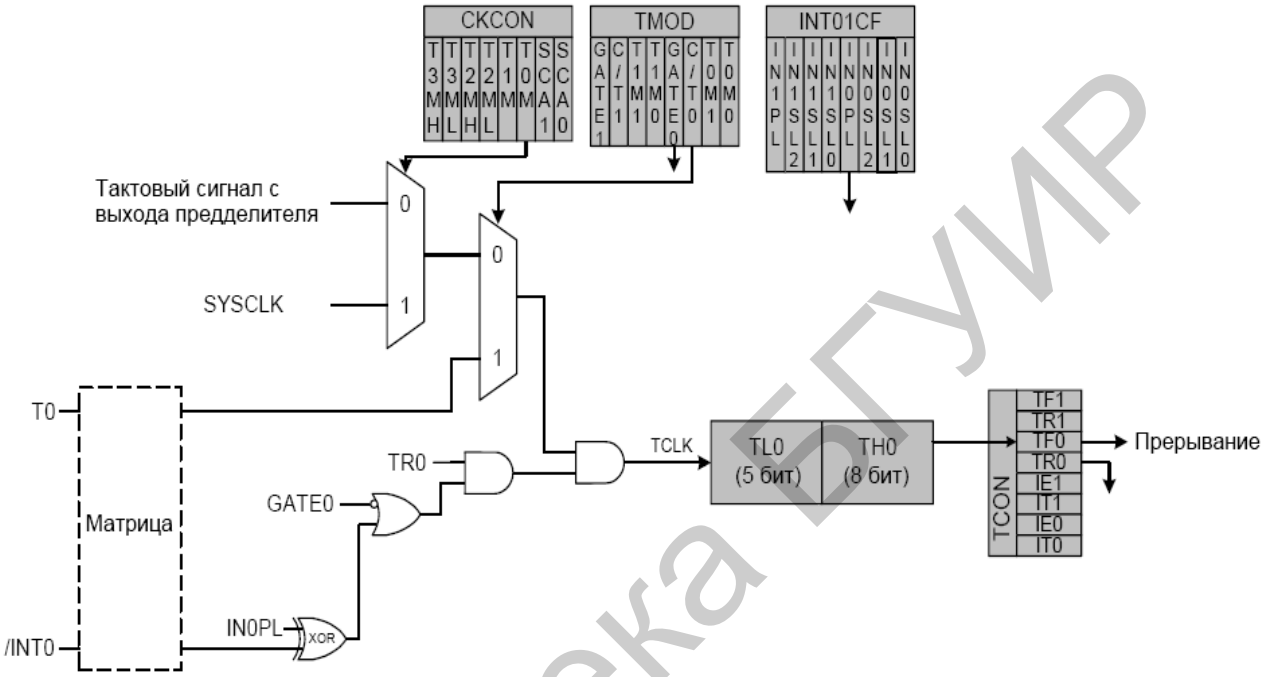

Рисунок 1.3 – Структурная схема Таймера 0 в режиме 0

Регистр TH0 содержит восемь старших битов 13-разрядного значения регистра ТС. Регистр TL0 содержит в разрядах TL0.4–TL0.0 пять младших битов 13-разрядного значения регистра ТС. Три старших бита регистра TL0 (TL0.7–TL0.5) не определены и должны маскироваться или игнорироваться при чтении регистра TL0. При инкрементировании 13-разрядного таймера и переполнении его из состояния 0x1FFF (все единицы) в состояние 0x0000 устанавливается в 1 флаг переполнения таймера TF0 (TCON.5) и будет сгенерировано прерывание, если оно разрешено. Бит C/T0 (TMOD.2) выбирает источник сигнала тактирования TC0. Если бит C/T0 установлен в 1, то инкрементирование регистра таймера осуществляется под воздействием перехода внешнего сигнала на выбранном входном выводе (T0) из состояния лог. 1 в состояние лог. 0. Если бит C/T0 сброшен в 0, то в качестве источника тактирования TC0 будет использоваться сигнал, определяемый битом T0M (CKCON.3). Если бит T0M установлен в 1, то Таймер 0 тактируется системным тактовым сигналом. Если бит T0M сброшен в 0, то в качестве источника тактирования TC0 будет использоваться сигнал, определяемый битами настройки предварительного делителя в регистре CKCON .

Установка в 1 бита TR0 (TCON.4) включит таймер, если либо бит GATE0 (TMOD.3) равен нулю, либо на внешнем выводе /INT0 присутствует сигнал с активным логическим уровнем, который определяется битом IN0PL в регистре INT01CF. После установки в 1 бита GATE0 управление таймером передается внешнему сигналу /INT0, что позволяет легко осуществлять измерение ширины импульсов (таблица 1.2).

Таблица 1.2 – Управление включением и выключением таймера-счетчика

TR0	**GATE0**	**/INT0**	**Таймер-счетчик**
0	X	X	Отключен
1	0	X	Включен
1	1	0	Отключен
1	1	1	Включен

Примечание – X обозначает «не имеет значения».

Установка TR0 не сбрасывает таймер. Регистры таймера следует инициализировать необходимыми значениями до включения таймера.

TL1 и TH1 образуют 13-разрядный регистр Таймера 1 точно так же, как описано ренее для регистров TL0 и TH0. Для настройки Таймера 1 и управления им используются соответствующие биты регистров TCON и TMOD таким же образом, как и для Таймера 0. Входной сигнал /INT1 используется совместно с Таймером 1; полярность /INT1 определяется битом IN1PL в регистре INT01CF.

Режим 1 аналогичен режиму 0 с тем лишь исключением, что регистры ТС используют все 16 бит. Таймеры-счетчики включаются и настраиваются в режиме 1 точно так же, как в режиме 0.

В режиме 2 Таймеры 0 и 1 настраиваются для работы в качестве 8-разрядных таймеров-счетчиков с автоматической перезагрузкой начального значения (рисунок 1.4). Регистр TL0 содержит значение счетчика, а регистр TH0 – перезагружаемое значение. Когда счетчик в регистре TL0 переполняется (переходит из состояния 0xFF в состояние 0x00), флаг переполнения таймера TF0 (TCON.5) устанавливается в 1 и значение регистра TH0 загружается в регистр TL0. При установке флага TF0 будет сгенерировано прерывание, если оно разрешено. Перезагружаемое значение в регистре TH0 не изменяется. Чтобы первый отсчет был корректным, необходимо проинициализировать регистр TL0 требуемым значением до включения таймера. Таймер 1 в режиме 2 работает точно так же, как Таймер 0.

В режиме 2 оба ТС включаются и настраиваются точно так же, как и в режиме 0. Установка в 1 бита TR0 (TCON.4) включит таймер, если либо бит GATE0 (TMOD.3) равен нулю, либо на внешнем выводе /INT0 присутствует сигнал с активным логическим уровнем, который определяется битом IN0PL в регистре INT01CF.

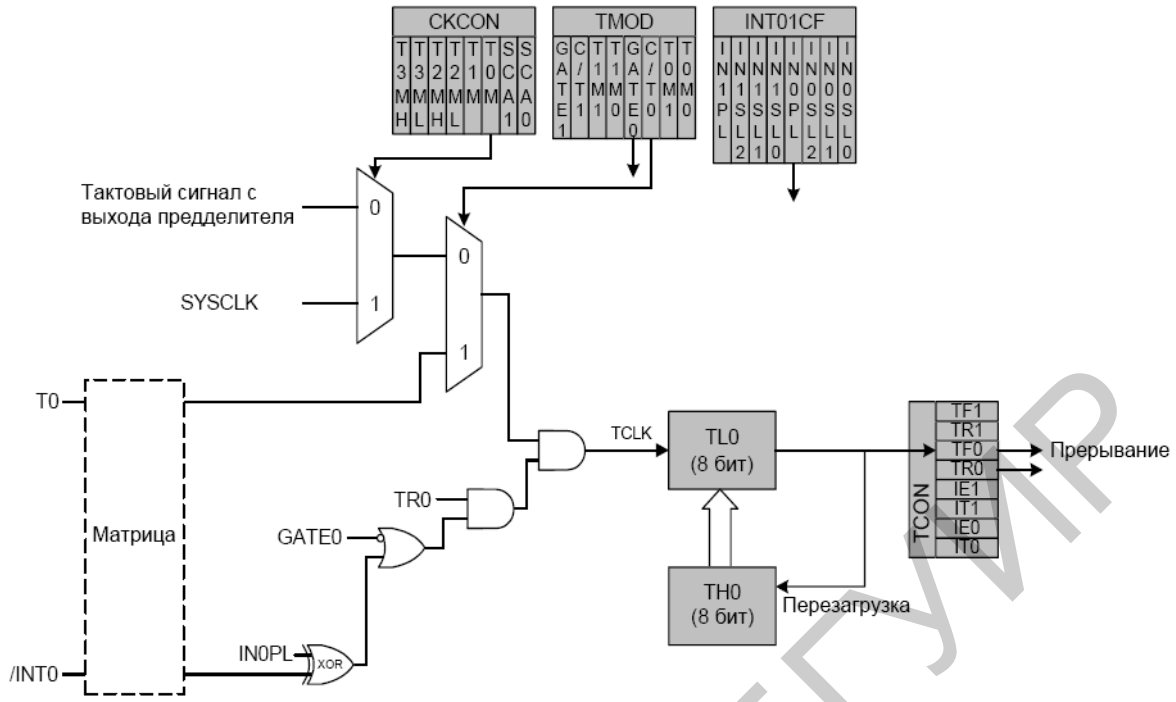

Рисунок 1.4 – Структурная схема Таймера 0 в режиме 2

1.2.4 Особенности создания программного обеспечения на языке С для микроконтроллеров семейства С8051

Язык С часто называют языком программирования среднего уровня. Но это не значит, что С менее мощный, менее развитой или более трудный в использовании, чем языки высокого уровня, такие как C++, C#, Java, JavaScript, Python, PHP и т. д. Это также не значит, что С такой же громоздкий и неудобный, как ассемблер. Языком среднего уровня его называют скорее потому, что он объединяет в себе лучшие черты языков высокого уровня с возможностями ассемблера.

Как язык среднего уровня С позволяет манипулировать битами, байтами и адресами, т. е. теми базовыми элементами данных, с которыми работают микроконтроллеры. Несмотря на это, программа, написанная на языке С, обладает высокой переносимостью. Переносимость – это свойство программного обеспечения, созданного для одного типа контроллеров (например, для изучаемого семейства с ядром 8051), позволяющее легко переделать его для другого типа контроллеров (например, для MSP 430, Cortex и т. д.).

Язык С является структурированным языком. Отличительная особенность структурированного языка состоит в возможности отдельного размещения различных частей кода программы и данных. Таким способом программист может «скрыть» часть информации, используемой для выполнения специфической задачи, от тех участков программы, где эта информация не нужна. Один из способов достижения этого – использование подпрограмм с локальными переменными. В этом случае любые действия внутри программы не вызовут побочных эффектов в других ее частях. Это позволяет программам, написанным на

языке С, совместно использовать готовые части кода. Для использования функции, хранящейся отдельно, необходимо только знать, что эта функция делает, при этом совсем не обязательно знать, как именно она это делает. Но следует помнить, что чрезмерное использование глобальных переменных (т. е. переменных, видимых во всей программе) приводит к ошибкам и побочным эффектам, которые очень трудно устранить.

Структурированные языки предоставляют программисту много различных возможностей благодаря тому, что содержат несколько типов операторов цикла, таких как while, do-while и for. Однако в структурированных языках запрещено или не рекомендовано использование оператора goto.

Главная конструкция структурного программирования на языке С – функция, являющаяся здесь единственным видом подпрограммы. Функция С – это «строительный кирпичик», в котором осуществляются все действия программы. Функции позволяют определить и отдельно закодировать различные задачи, решаемые программой, благодаря чему эта программа становится модульной. Написав правильно функцию, можно быть уверенным в ее надежной работе в различных ситуациях без побочных эффектов в других частях программы. При работе над большим проектом, когда особенно важно, чтобы одна часть кода ни в коем случае не могла непредвиденно подействовать на другую часть, умение создать отдельную функцию приобретает для программиста исключительное значение.

Структура программы на языке С

Ниже перечислены 32 ключевых слова, определенные стандартом С89. Они же являются ключевыми словами языка С как подмножества С++. Набор ключевых слов вместе с формальным синтаксисом С составляет язык программирования С.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizof	volatile
do	if	static	while

Кроме стандартных ключевых слов многие компиляторы для лучшего функционирования в среде программирования разрешают дополнительно использовать некоторые нестандартные ключевые слова.

В языке С различаются верхний и нижний регистры символов: else – ключевое слово, а ELSE – нет. В программе ключевое слово может быть использовано только как ключевое слово, т. е. никогда не допускается его использование в качестве переменной или имени функции.

Любая программа на языке С состоит из одной или нескольких функций. Обязательно должна быть определена единственная главная функция main(), именно с нее всегда начинается выполнение программы. В хорошем исходном тексте программы главная функция всегда содержит операторы, отражающие сущность решаемой задачи, чаще всего это вызовы функций. Хотя main() и не является ключевым словом, относиться к нему следует как к ключевому. Например, не следует использовать main как имя переменной, так как это может нарушить работу транслятора.

Базовые типы данных

Стандарт С89 определяет пять фундаментальных типов данных: char – символьные данные, int – целые, float – с плавающей точкой, double – двойной точности, void – без значения. На основе этих типов формируются другие типы данных. Размер (объем занимаемой памяти) и диапазон значений этих типов данных для разных процессоров и компиляторов могут быть разными (таблица 1.3). Однако объект типа char всегда занимает 1 байт. Размер объекта int обычно совпадает с размером слова в конкретной среде программирования.

Таблица 1.3 – Все типы данных, определенные стандартом С

Тип	Типичный размер в битах	Минимально допустимый диапазон значений
char	8	от –127 до 127
unsigned char	8	от 0 до 255
int	16	от –32 767 до 32 767
unsigned int	16	от 0 до 65 535
long int	32	от –2 147 483 647 до 2 147 483 647
long long int	64	от $-(2^{63}-1)$ до $(2^{63}-1)$, добавлен стандартом С99
float	32	от 1E–37 до 1E+37, с точностью не менее 6 значащих десятичных цифр
double	64	от 1E–37 до 1E+37, с точностью не менее 10 значащих десятичных цифр
long double	80	от 1E–37 до 1E+37, с точностью не менее 10 значащих десятичных цифр

Тип void служит для объявления функции, не возвращающей значения, или для создания универсального указателя.

Имена переменных

В языке С имена переменных, функций, меток и других параметров называются идентификаторами. Длина идентификатора (количество символов, из которых состоит идентификатор) является натуральным числом, обычно идентификатор представляет собой последовательность из одного или нескольких символов: первый должен быть буквой или символом подчеркивания, последующие – буквами, цифрами или символами подчеркивания.

Верхние и нижние регистры символов рассматриваются как различные. Следовательно, count, Count и COUNT – это три разных идентификатора.

Переменные

Переменная – это именованный участок памяти, где хранится значение, которое может быть изменено программой. Все переменные перед их использованием должны быть объявлены. Общая форма объявления имеет такой вид:

тип список_переменных;

Здесь *тип* означает один из базовых или объявленных программистом типов (если необходимо – с одним или несколькими спецификаторами), а *список_переменных* состоит из одного или более идентификаторов, разделенных запятыми. Ниже приведены примеры объявлений:

```
int     i,j,l;
short     int si;
unsigned     int ui;
double      balance, profit, loss;
```

Примечание – Необходимо помнить, что в языке С имя переменной никогда не определяет ее тип.

Объявление переменных может быть расположено в трех местах: внутри функции, в определении параметров функции и вне всех функций. Это места объявлений соответственно локальных переменных, формальных параметров функций и глобальных переменных.

Локальные переменные

Переменные, объявленные внутри функций, называются локальными переменными. Локальную переменную можно использовать только внутри блока, в котором она объявлена. Иными словами, локальная переменная невидима за пределами своего блока.

Примечание – Блок программы – это описания и инструкции, объединенные в одну конструкцию путем заключения их в фигурные скобки { }.

Формальные параметры функции

Если функция имеет аргументы, значит должны быть объявлены переменные, которые примут их значения. Эти переменные называются формальными параметрами функции. Внутри функции они фигурируют как обычные локальные переменные.

Глобальные переменные

В отличие от локальных, глобальные переменные видимы и могут использоваться в любом месте программы. Они сохраняют свое значение на протяжении всей работы программы. Чтобы создать глобальную переменную, ее необходимо объявить за пределами функции. Глобальная переменная может быть использована в любом выражении, независимо от того, в каком блоке это выражение используется.

При объявлении переменной она может быть инициализирована. Для этого нужно после ее объявления поставить знак равенства и константу, т. е. общая форма инициализации имеет следующий вид:

тип имя_переменной = константа;

Приведем несколько примеров инициализации переменных:

```
char ch = 'a';
int first = 0;
double balance = 123.23;
```

Глобальные и статические локальные переменные инициализируются только один раз в начале работы программы, а локальные переменные (исключая статические локальные) – каждый раз при входе в блок, в котором они объявлены. Неинициализированные локальные переменные до первого присвоения имеют произвольное значение. Неинициализированные глобальные и статические локальные переменные в начале работы программы автоматически обнуляются.

Операции

Язык С содержит большое количество встроенных операций. Их роль в языке С значительнее, чем в других языках программирования. Существует четыре основных класса операций: *арифметические*, *логические*, *поразрядные* и *операции сравнения*. Кроме них есть также некоторые специальные операторы, например оператор присваивания.

Оператор присваивания

Оператор присваивания может присутствовать в любом выражении языка С. Этим данный язык отличается от большинства других языков программирования (Pascal, BASIC и FORTRAN), в которых присваивание возможно только в отдельном операторе. Общая форма оператора присваивания:

имя_переменной=выражение;

Выражение может быть просто константой или сколь угодно сложным выражением. В языке С оператором присваивания служит единственный знак присваивания "=". *Адресатом* (*получателем*), т. е. левой частью оператора присваивания должен быть объект, способный получить значение, например переменная.

Преобразование типов при присваиваниях

Если в операции встречаются переменные разных типов, происходит *преобразование типов*. В операторе присваивания действует простое правило: значение выражения в правой части преобразуется к типу объекта в левой части.

Преобразование целых в символы и длинных целых в целые удаляет соответствующее количество старших двоичных разрядов. В 16-разрядной среде теряются 8 бит при преобразовании целого в символ и 16 бит при преобразовании длинного целого в целое. В 32-разрядной среде теряются 24 бита при преобразовании целого в символ и 16 бит при преобразовании целого в короткое целое.

В таблице 1.4 приведены варианты потери информации при некоторых преобразованиях.

Таблица 1.4 – Результат некоторых преобразований типов

Тип адресата	**Тип выражения**	**Потеря информации**
signed char	char	Если значение >127, то результат отрицательный
char	short int	Старшие 6 бит
char	int (16-разрядный)	Старшие 8 бит
char	int (32-разрядный)	Старшие 24 бита
char	long int	Старшие 24 бита
short int	int (16-разрядный)	Нет
short int	int (32-разрядный)	Старшие 16 бит
int (16-разрядный)	long int	Старшие 16 бит
int (32-разрядный)	long int	Нет
long int (32-разрядный)	long long int (64-разрядный)	Старшие 32 бита (это относится только к С99)
int	float	Дробная часть
float	double	Результат округляется
double	long double	Результат округляется

Необходимо помнить, что преобразование int во float или float в double не повышает точность вычислений. При таком преобразовании только изменяется форма представления числа. Некоторые компиляторы при преобразовании char в int считают переменную char положительной независимо от ее значения. Другие компиляторы считают переменную char отрицательной, если она больше 127. Поэтому для обеспечения переносимости программы необходимо использовать переменные типа char для хранения символов, а переменные типа signed char и int (целый) – для хранения чисел.

Если какое-либо преобразование не приведено в таблице 1.4, то чтобы определить, что именно теряется в результате этого преобразования, нужно представить его в виде композиции (суперпозиции, произведения) указанных в таблице преобразований и затем провести последовательные преобразования. Например, преобразование double в int эквивалентно последовательному выполнению двух преобразований: сначала double в float, а затем float в int.

Множественные присваивания

В одном операторе присваивания можно присвоить одно и то же значение многим переменным. Для этого используется оператор *множественного присваивания*, например:

```
x = y = z = 0;
```

Следует отметить, что в практике программирования этот прием используется очень часто.

Составное присваивание

Составное присваивание – это разновидность оператора присваивания, в которой запись сокращается и становится более удобной в написании.

Например, оператор

```
x = x+10;
```

можно записать как

```
x += 10;
```

Оператор «+=» сообщает компилятору, что к переменной x нужно прибавить 10.

Составные операторы присваивания существуют для всех бинарных операций (т. е. операций, имеющих два операнда). Любой оператор вида

переменная = переменная оператор выражение;

можно записать как

переменная оператор = выражение;

Еще один пример:

```
x = x–100;
```

означает то же самое, что и

```
x – = 100;
```

Составное присваивание значительно компактнее, чем соответствующее простое присваивание, поэтому его иногда называют *стенографическим* (*shorthand*) *присваиванием*. В программах на языке С этот оператор широко используется, поэтому необходимо хорошо его усвоить.

Арифметические операции

В таблице 1.5 приведены арифметические операции языка С. Операции «+, –, * и /» работают так же, как и в большинстве других языков программирования. Их можно применять почти ко всем встроенным типам данных. Если операция «/» применяется к целому или символьному типам, то остаток от деления отбрасывается. Например, результатом операции 5/2 является 2.

Таблица 1.5 – Арифметические операции

Оператор	Операция
–	Вычитание, также унарный минус
+	Сложение
*	Умножение
/	Деление
%	Остаток от деления
– –	Декремент, или уменьшение
++	Инкремент, или увеличение

Оператор деления по модулю % в языке С работает так же, как и в других языках, его результатом является остаток от целочисленного деления. Этот оператор, однако, нельзя применять к типам данных с плавающей точкой.

Операции сравнения и логические операции

Операции *сравнения* – это операции, в которых значения двух переменных сравниваются друг с другом. *Логические* же операции реализуют средствами языка С операции формальной логики. Между логическими операциями и операциями сравнения существует тесная связь: результаты операций *сравнения* часто являются операндами *логических* операций.

В операциях сравнения и логических операциях в качестве операндов и результатов операций используются значения ИСТИНА (true) и ЛОЖЬ (false). В языке С значение ИСТИНА представляется любым числом, отличным от 0. Значение ЛОЖЬ представляется 0. Результатом операции сравнения или логической операции являются ИСТИНА (true, 1) или ЛОЖЬ (false, 0).

В таблице 1.6 приведен полный список операций сравнения и логических операций.

Таблица истинности логических операций имеет следующий вид.

p	***q***	***p && q***	***p || q***	***!p***
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Как операции сравнения, так и логические операции имеют низший приоритет по сравнению с арифметическими. То есть выражение 10>1+12 интерпретируется как 10>(1+12). Результат, конечно, равен ЛОЖЬ.

В одном выражении можно использовать несколько операций:

10>5 && !(10<9) || 3<4

В этом случае результатом будет ИСТИНА.

Как и в арифметических выражениях, для изменения порядка выполнения операций сравнения и логических операций можно использовать круглые скобки.

Таблица 1.6 – Операции сравнения и логические операции

Оператор	Операция
Операторы сравнения	
>	Больше чем
>=	Больше или равно
<	Меньше чем
<=	Меньше или равно
==	Равно
!=	Не равно
Логические операции	
&&	И
||	ИЛИ
!	НЕ, отрицание

Необходимо помнить, что результатом любой операции сравнения или логической операции есть 0 или 1.

Поразрядные операции

В отличие от многих других языков программирования, в языке С определен полный набор *поразрядных операций*. Это обусловлено тем, что данный язык был задуман как язык, призванный во многих приложениях заменить ассемблер, который способен оперировать битами данных. *Поразрядные операции* – это тестирование (проверка), сдвиг или присвоение значений отдельным битам данных. Эти операции осуществляются над ячейками памяти, содержащими данные типа char или int. Данные типа float, double, long double, void или другие более сложные не могут участвовать в поразрядных операциях. В таблице 1.7 приведен полный список знаков поразрядных операций, выполняемых над отдельными разрядами (битами) операндов.

Таблица 1.7 – Поразрядные операции

Оператор	Операция
&	И
|	ИЛИ
^	Исключающее ИЛИ
~	НЕ (отрицание, дополнение к 1)
>>	Сдвиг вправо
<<	Сдвиг влево

Таблицы истинности логических операций и поразрядных операций И, ИЛИ, НЕ совпадают. Отличие лишь в том, что поразрядные операции выполняются над отдельными разрядами (битами) операндов. Операция «исключающее ИЛИ» имеет следующую таблицу истинности.

p	***q***	***p ^ q***
0	0	0
1	0	1
1	1	0
0	1	1

Как показано в таблице, результат операции «исключающее ИЛИ» равен ИСТИНА, только если один из операндов равен 1, иначе результат будет равен ЛОЖЬ.

Наиболее часто поразрядные операции используются для маскирования определенных битов.

Операция И может быть использована для *очищения бита*. Иными словами, для гашения бита используется следующее свойство операции И: если бит одного из операндов равен 0, то соответствующий бит результата будет равен 0, независимо от состояния этого бита во втором операнде.

Поразрядная операция ИЛИ, являющаяся двойственной операции И, применяется для установки необходимых битов в 1.

Операция исключающего ИЛИ (XOR) устанавливает бит результата в 1, если соответствующие биты операндов различны.

Поразрядные операторы сдвига >> и << сдвигают все биты переменной вправо или влево. Общая форма оператора сдвига вправо:

переменная >> количество_разрядов

Общая форма оператора сдвига влево:

переменная << количество_разрядов

Во время сдвига битов в один конец числа другой конец заполняется нулями. Но если число типа signed int отрицательно, то при сдвиге вправо левый конец заполняется единицами, так что знак числа сохраняется. Необходимо отметить различие между сдвигом и циклическим сдвигом. При циклическом сдвиге биты, сдвигаемые за пределы операнда, появляются на другом конце операнда. А при сдвиге вышедшие за границу биты теряются.

Поразрядные операции сдвига очень полезны при декодировании выходов внешних устройств, например таких, как цифроаналоговые преобразователи, а также при считывании информации о статусе устройств. Побитовые операторы сдвига могут быстро умножать и делить целые числа. Как показано в табли-

це 1.7, сдвиг на один бит вправо делит число на 2, а на один бит влево – умножает на 2. Поразрядная операция отрицания (дополнения) ~ инвертирует состояние каждого бита операнда. То есть 0 преобразует в 1, а 1 – в 0.

Операция получения адреса (&) и раскрытия ссылки (*)

Указатель – это адрес объекта в памяти. *Переменная типа «указатель» (или просто переменная-указатель)* – это специально объявленная переменная, в которой хранится указатель на переменную определенного типа. В языке С указатели служат мощнейшим средством создания программ и широко используются для самых разных целей. Например, с их помощью можно быстро обратиться к элементам массива или дать функции возможность модифицировать свои аргументы. Указатели широко используются для связи элементов в списках, двоичных деревьях и других динамических структурах данных.

Первый из них – *оператор &* – это унарный оператор, возвращающий адрес операнда в памяти. (Унарной операцией называется операция, имеющая только один операнд.) Например, оператор

```
m = &count;
```

записывает в переменную m адрес переменной count. Этот адрес представляет собой адрес ячейки памяти компьютера, в которой размещена переменная. Адрес и значение переменной – совершенно разные понятия. Выражение «&переменная» означает «адрес переменной». Следовательно, инструкция m = &scount; означает «Переменной m присвоить адрес, по которому расположена переменная count;».

Допустим, переменная count расположена в памяти в ячейке с адресом 2000, а ее значение равно 100. Тогда в предыдущем примере переменной m будет присвоено значение 2000.

Второй рассматриваемый *оператор* * является двойственным (дополняющим) по отношению к &. Оператор * является унарным оператором, он возвращает значение объекта, расположенного по указанному адресу. Операндом для * служит адрес объекта (переменной). Например, если переменная m содержит адрес переменной count, то оператор

```
q = *m;
```

записывает значение переменной count в переменную q. В нашем примере переменная q получит значение 100, потому что по адресу 2000 записано число 100, причем этот адрес записан в переменной m. Выражение «* адрес» означает «по адресу». Наш фрагмент программы можно прочесть как «q получает значение, расположенное по адресу m».

К сожалению, символ операции раскрытия ссылки совпадает с символом операции умножения, а символ операции получения адреса – с символом операции поразрядного И. Необходимо помнить, что эти операторы не имеют никакого отношения друг к другу. Операторы * и & имеют более высокий приоритет, чем любая арифметическая операция, кроме унарного минуса, имеющего такой же приоритет.

Если переменная является указателем, то в объявлении перед ее именем нужно поставить символ *, он сообщит компилятору о том, что это указатель на переменную данного типа. Например, объявление указателя на переменную типа char записывается так:

```
char *ch;
```

Необходимо понимать, что ch – это не переменная типа char, а указатель на переменную данного типа, это совершенно разные вещи. Тип данных, на который указывает указатель (в данном случае это char), называется *базовым типом* указателя. Сам указатель является переменной, содержащей адрес объекта базового типа. Компилятор учтет размер указателя в архитектуре компьютера и выделит для него необходимое количество байтов, чтобы в указатель поместился адрес. Базовый тип указателя определяет тип объекта, хранящегося по этому адресу.

В одном операторе объявления можно одновременно объявить и указатель, и переменную, не являющуюся указателем. Например, оператор

```
int x, *y, count;
```

объявляет x и count как переменные целого типа, а y – как указатель на переменную целого типа.

Оператор доступа к члену структуры (оператор . (точка)) и оператор доступа через указатель -> (оператор стрелка)

В языке С операторы . (точка) и -> (стрелка) обеспечивают доступ к элементам структур и объединений. *Структуры* и *объединения* – это составные типы данных, в которых под одним именем хранятся многие объекты

Оператор точка используется для прямой ссылки на элемент структуры или объединения, т. е. перед точкой стоит имя структуры, а после – имя элемента структуры. Оператор стрелка используется с указателем на структуру или объединение, т. е. перед стрелкой стоит указатель на структуру.

Оператор [] и ()

Круглые скобки являются оператором, повышающим приоритет выполнения операций, которые в них заключены. Квадратные скобки служат для индексации массива. Если в программе определен массив, то выражение в квадратных скобках представляет собой индекс массива.

1.3 Приборы и оборудования, применяемые в работе

В данной лабораторной работе используются:

1. Комплект разработки для микроконтроллера C8051F320.
2. Персональный компьютер с USB-портом.
3. Программное обеспечение.

1.4 Порядок выполнения работы

1. Запустить среду разработки программ для микроконтроллеров SiLabs.

2. Подготовить комплект отладки и ПК. Подключить плату к программатору, подключить программатор к компьютеру по интерфейсу USB. Подать питание на плату с помощью 9В блока питания. Установить связь МК с компьютером.

3. Включить программу F32x_Timer0_16bitTimer.c в проект.

4. Провести инициализацию микроконтроллера:

- инициализация кварцевого резонатора (регистр OSCICN);
- инициализация порта (регистры XBR0, XBR1, PnMDIN, PnMDOUT, PnSKIP);
- инициализация таймера (регистры TCON, TMOD);
- инициализация прерывания (регистры IE, IP).

5. Установить значения в таймеры для генерации сигнала необходимой частоты (регистры TH0, TL0), рассчитанные исходя из заданной частоты мерцания светодиода (частота задана в таблице 1.8).

6. Скомпилировать программу.

7. Выполнить построение проекта.

8. Записать программу во внутреннюю память МК.

9. Запустить работу программы.

Таблица 1.8 – Варианты заданий

Номер варианта	**Частота, Гц**	**Номер варианта**	**Частота, Гц**
1	0,1	16	1,8
2	1,9	17	1,2
3	3,2	18	4,7
4	2,8	19	1,2
5	3,4	20	5
6	1,2	21	4,5
7	3,7	22	2,3
8	4,2	23	2,8
9	2,6	24	0,1
10	0,1	25	2,5
11	3,2	26	1,2
12	1,6	27	4,8
13	3,3	28	0,7
14	3,7	29	3,4
15	3,9	30	0,4

1.5 Содержание отчета

1. Цель работы.
2. Листинг программы генерации сигнала.
3. Выводы.

1.6 Контрольные вопросы

1. Какова структура встроенной памяти микроконтроллера?
2. Перечислите источники прерываний микроконтроллера и опишите регистры прерываний.
3. Какие существуют порты ввода/вывода, какие регистры используются для их управления?
4. Как осуществляется инициализация портов ввода/вывода?
5. Какие существуют режимы работы Таймера 0 и Таймера 1?
6. Перечислите основные регистры Таймера 0 и Таймера 1 их назначение.

Лабораторная работа №2

Работа с таймерами-счетчиками микроконтроллера. Генерация сигналов с произвольной частотой и скважностью

2.1 Цель работы

Изучить организацию работы портов ввода/вывода микроконтроллера Cygnal C8051F320 и разработать программу для подключения кнопок управления.

2.2 Теоретические сведения

Порты ввода/вывода в микроконтроллерах Cygnal C8051F320

Для доступа к аналоговым и цифровым ресурсам МК используются 17 внешних линий ввода/вывода, которые организованы как два 8-разрядных порта и один 1-разрядный порт. Каждый из выводов портов можно определить как порт ввода/вывода общего назначения (GPIO) или аналоговый вход; выводы портов P0.0–P1.7 можно назначить одному из внутренних цифровых модулей, как показано на рисунке 2.1. Разработчик системы сам определяет, какие цифровые ресурсы будут назначены внешним выводам, ограничиваясь только количеством доступных выводов. Такая гибкость при распределении ресурсов достигается благодаря использованию приоритетного декодера матрицы. Следует иметь в виду, что состояние на внешнем выводе порта всегда можно прочитать из соответствующего регистра-защелки порта, независимо от настройки матрицы.

Матрица назначает внешние порты ввода/вывода выбранным внутренним цифровым ресурсам, используя приоритетный декодер. Для выбора внутренних цифровых ресурсов используются регистры XBR0 и XBR1.

Допустимое напряжение на внешних выводах портов составляет 5 В (см. схему ячейки порта на рисунке 2.2). С помощью регистров настройки выходов портов (PnMDOUT, где n = 0,1) можно настроить выходные драйверы портов ввода/вывода либо как обычные цифровые выходы, либо как выходы с открытым стоком.

Приоритетный декодер матрицы

Приоритетный декодер матрицы (рисунок 2.3) назначает приоритет каждой функции ввода/вывода, начиная с выводов UART0. Если какой-либо цифровой ресурс выбран, то этому ресурсу назначается еще неназначенный внешний вывод с наименьшим приоритетом (кроме UART0, которому всегда назначаются выводы P0.4 и P0.5). Если вывод порта назначен, то матрица пропускает этот вывод при назначении выводов следующему выбранному ресурсу. Кроме того, матрица будет пропускать выводы портов, если соответствующие им биты в регистрах PnSKIP установлены в 1. Регистры PnSKIP позволяют программе настроить матрицу таким образом, чтобы она пропускала выводы портов, которые используются в качестве аналоговых входов, портов ввода/вывода общего назначения или для реализации специальных функций.

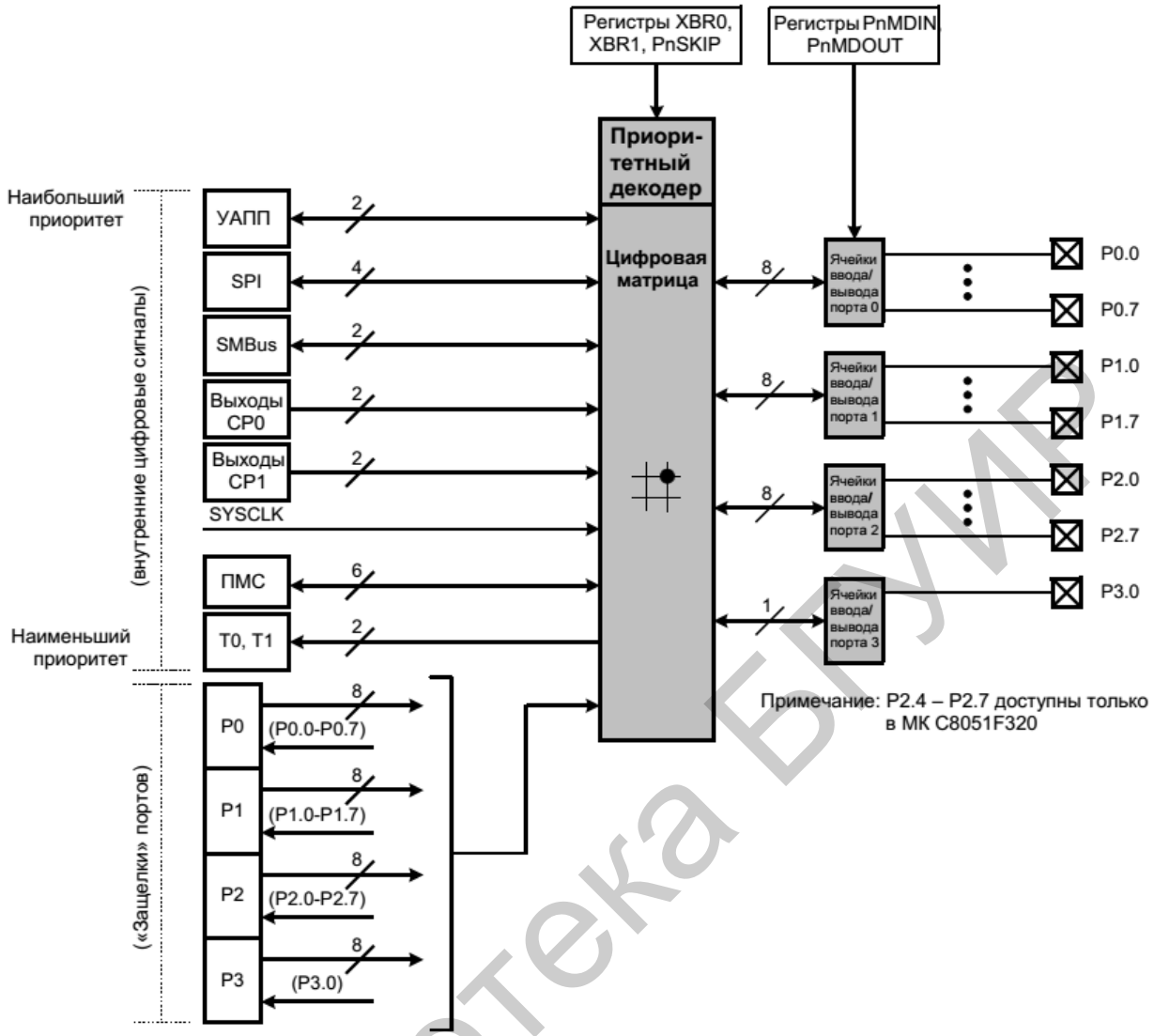

Рисунок 2.1 – Функциональная схема портов ввода/вывода

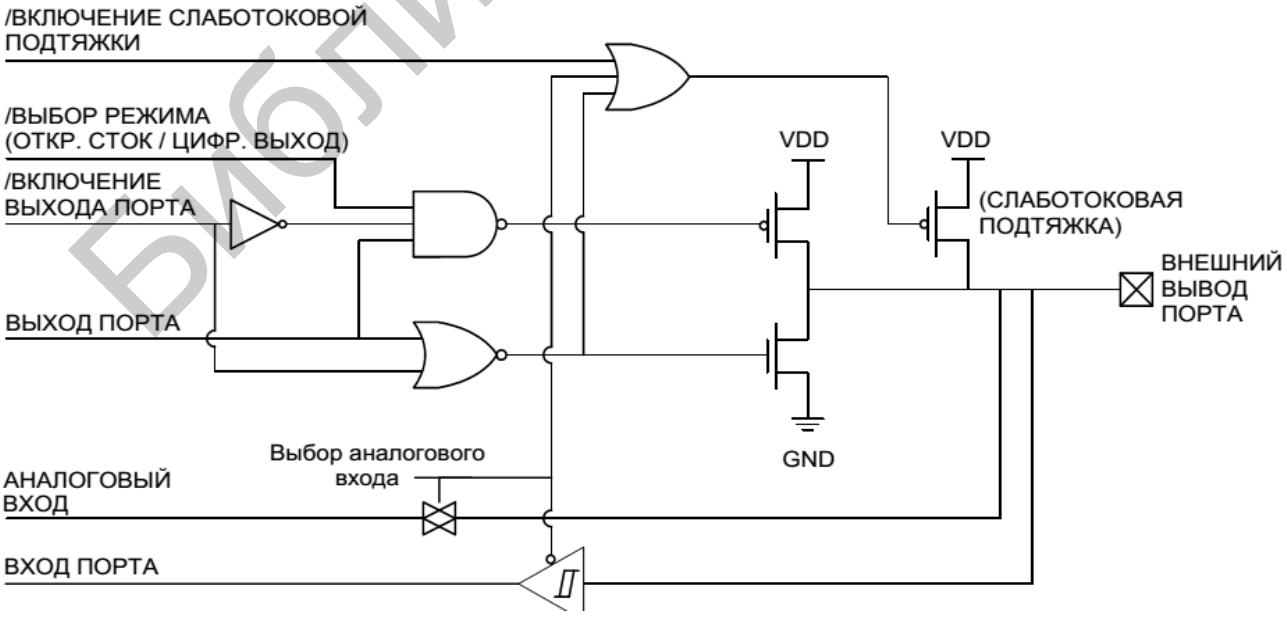

Рисунок 2.2 – Структурная схема ячейки порта ввода/вывода

Замечание. Если вывод порта закреплен за периферийным модулем без использования матрицы, то соответствующий ему бит в регистрах PnSKIP должен быть установлен в 1. Это касается P0.0, если используется VREF, P0.3 и/или P0.2, если включена схема возбуждения внешнего генератора, P0.6, если АЦП или ЦАП настроены на использование внешнего сигнала запуска преобразования (CNVSTR), а также любых выбранных входов АЦП или компаратора. Матрица пропускает выбранные выводы, как если бы они были уже назначены, и переходит к следующему неназначенному выводу. На рисунке 2.3 показаны приоритеты декодера матрицы без пропуска каких-либо выводов портов (P0SKIP, P1SKIP = 0x00).

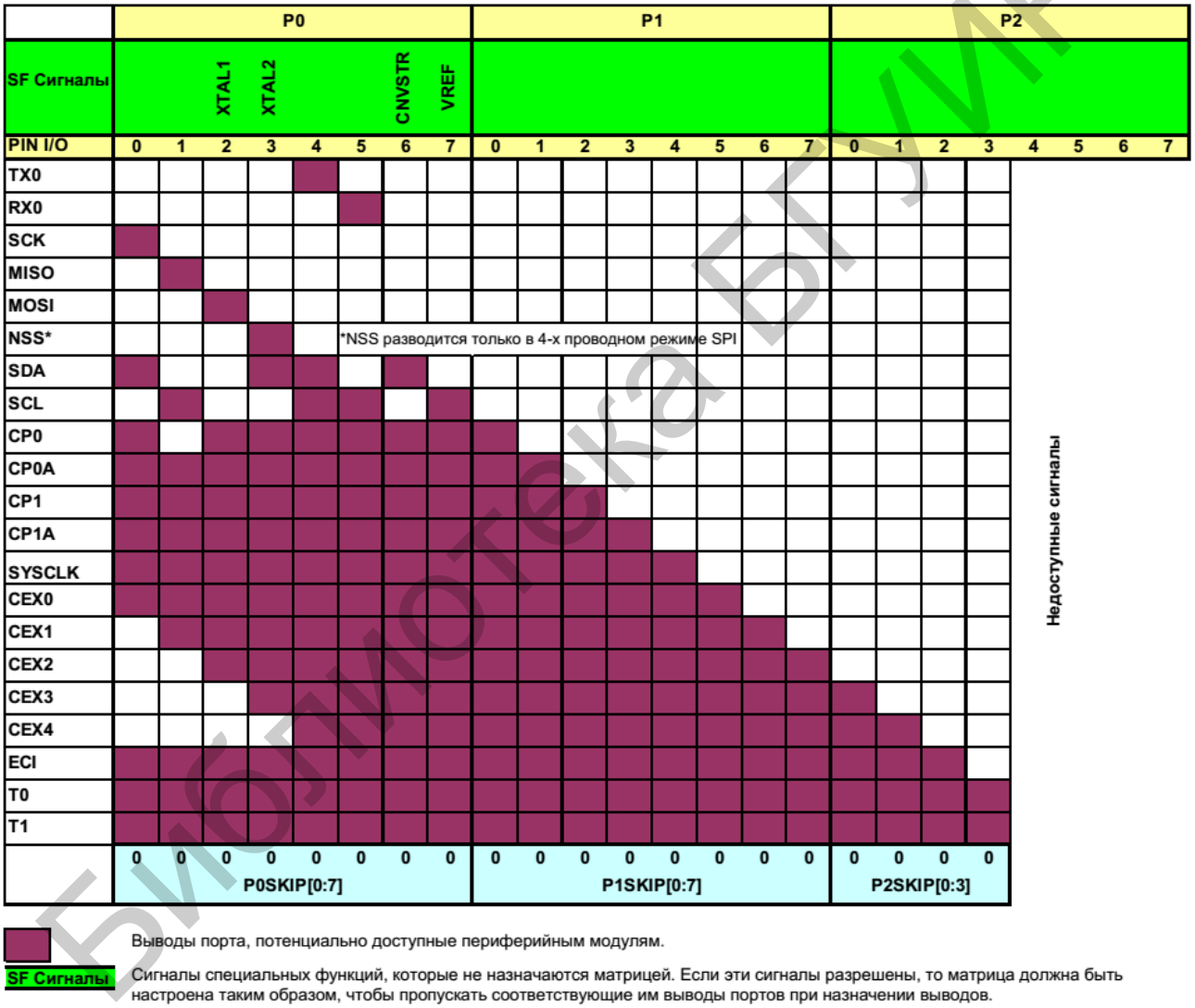

Рисунок 2.3 – Приоритетный декодер матрицы (нет пропускаемых выводов)

Регистры XBR0 и XBR1 используются для назначения цифровых ресурсов внешним портам ввода/вывода. Следует иметь в виду, что если выбран SMBus, то матрица назначает оба вывода, связанные с модулем SMBus (SDA и SCL); если выбран UART0, то матрица назначает оба вывода, связанные с модулем UART0 (TX и RX). Назначение выводов UART0 фиксировано с целью

обеспечения возможности самозагрузки: TX0 всегда назначается выводу P0.4; RX0 всегда назначается выводу P0.5. После назначения приоритетных функций следуют стандартные порты ввода/вывода.

Замечание. Модуль SPI может функционировать в 3- или 4-проводном режимах в зависимости от состояния битов NSSMD1–NSSMD0 в регистре SPI0CN. В соответствии с режимом работы SPI сигнал NSS либо разводится, либо не разводится на внешний порт.

Порт ввода/вывода общего назначения

Выводы портов, которые не назначены матрицей и не используются аналоговыми периферийными модулями, можно использовать в качестве выводов ввода/вывода общего назначения. Порты 3–0 доступны с помощью соответствующих SFR-регистров как в побайтном, так и в побитном режимах адресации. При записи в порт значение, записываемое в SFR-регистр, «защелкивается»; это позволяет удерживать на каждом выводе порта выходное значение. При чтении логические уровни входных выводов портов возвращаются независимо от значений регистров XBRn (т. е. даже если вывод назначен матрицей другому сигналу, регистр порта все равно может прочитать логическое состояние на соответствующем входе). Исключением являются команды типа *чтение-модификация-запись*. При работе с SFR-регистром порта командами типа *чтение-модификация-запись* являются следующие команды: ANL, ORL, XRL, JBC, CPL, INC, DEC, DJNZ, а также MOV, CLR или SETB, если они адресуют отдельный бит в SFR-регистре порта. В случае использования этих команд считывается, модифицируется и записывается обратно значение регистра (а не вывода).

Инициализация порта ввода/вывода осуществляется следующим образом:

1. Выбрать тип входа (аналоговый или цифровой) для всех выводов порта, используя регистр настройки входов порта (PnMDIN).

2. Выбрать тип выхода (с открытым стоком или 2-тактный цифровой) для всех выводов порта, используя регистр настройки выходов порта (PnMDOUT).

3. Выбрать все выводы, которые должны пропускаться матрицей при назначении выводов, используя регистры выбора пропускаемых выводов (PnSKIP).

4. Назначить выводы порта требуемым периферийным модулям (XBR0, XBR1).

5. Включить матрицу (XBARE = ‘1’).

Все выводы порта должны быть настроены как аналоговые или как цифровые входы. Любые выводы, используемые в качестве входов компаратора или АЦП, должны быть настроены как аналоговые входы. Если вывод настроен как аналоговый вход, то его слаботоковая подтяжка, цифровой драйвер и цифровой приемник отключаются. Это позволяет снизить энергопотребление и уменьшить уровень шумов на аналоговом входе.

Выводы, настроенные как цифровые входы, все равно могут использоваться аналоговыми периферийными модулями, однако это не рекомендуется.

Чтобы настроить вывод порта как цифровой вход, следует сбросить в 0 соответствующий бит в регистре PnMDOUT и установить в 1 соответствующий бит регистра-защелки порта (регистр Pn).

Кроме этого, все аналоговые входы необходимо настроить таким образом, чтобы они пропускались матрицей при назначении выводов (это достигается установкой в 1 соответствующих битов в регистрах PnSKIP).

Тип входа устанавливается с помощью соответствующих битов регистра PnMDIN (1 – цифровой вход, 0 – аналоговый вход). При сбросе все выводы настраиваются по умолчанию как цифровые входы. Параметры выходных драйверов выводов порта задаются с помощью регистров настройки выходов порта (PnMDOUT). Выходной драйвер каждого порта можно настроить либо как цифровой 2-тактный выход, либо как выход с открытым стоком. Такая настройка не осуществляется автоматически, ее необходимо выполнить даже для цифровых ресурсов, выбранных в регистрах XBRn. Единственным исключением из этого правила являются выводы SMBus (SDA, SCL), которые настраиваются как выходы с открытым стоком, независимо от значения PnMDOUT. Если бит WEAKPUD в регистре XBR1 сброшен в 0, то слаботоковая подтяжка отключается у всех выводов портов, настроенных как выходы с открытым стоком. Бит WEAKPUD не влияет на выводы, настроенные как цифровые 2-тактные выходы. Более того, слаботоковая подтяжка отключается у выхода, на который выведен лог. 0, чтобы предотвратить нежелательное увеличение энергопотребления.

Для выбора цифровых ресурсов, требуемых для конкретного проекта, необходимо загрузить регистры XBR0 и XBR1 соответствующими значениями. Установка в 1 бита XBARE в регистре XBR1 включает матрицу. До включения матрицы внешние выводы остаются стандартными портами ввода/вывода (настроенными на вход), независимо от значений регистров XBRn. Зная значение регистров XBRn, можно определить разводку выводов, используя таблицу декодирования приоритетов.

Замечание. Чтобы использовать порты P0, P1 и P2.0–P2.3 как стандартные порты ввода/вывода в режиме выходов, необходимо включить матрицу. Выходные драйверы этих портов отключаются при выключении матрицы. P2.4–P2.7 и P3.0 всегда функционируют как стандартные порты ввода/вывода общего назначения.

2.3 Приборы и оборудования, применяемые в работе

В данной лабораторной работе используются:

1. Комплект разработки для микроконтроллера C8051F320.
2. Персональный компьютер с USB-портом.
3. Программное обеспечение.

2.4 Порядок выполнения работы

1. Запустить среду разработки программ для микроконтроллеров SiLabs.

2. Подготовить комплект отладки и ПК. Подключить плату к программатору, подключить программатор к компьютеру по интерфейсу USB. Подать питание на плату с помощью 9В блока питания. Установить связь МК с компьютером.

3. Включить программу main_TimerWork2.c в проект.

4. Провести инициализацию микроконтроллера:

- инициализация внутреннего кварцевого резонатора (регистр OSCICN);
- инициализация порта (регистры XBR0, XBR1, PnMDIN, PnMDOUT, PnSKIP) для подключения светодиода и кнопки;
- инициализация таймеров 0 и 1 (регистры TCON, TMOD) в реж. 1 и 2;
- инициализация прерывания (регистры IE, IP) от таймера-счетчика.

5. Установить значения в таймеры для генерации сигнала необходимой частоты (регистры TH0, TL0).

6. Запустить работу программы (кнопка Run среды).

7. Разработать алгоритм управления кнопкой. При нажатии частота мерцания светодиода изменяется в соответствии с заданием (задание необходимо получить у преподавателя перед началом выполнения лабораторной работы).

8. Написать программу управления кнопкой, включающую подпрограмму гашения дребезга контактов.

9. Скомпилировать программу.

10. Выполнить построение проекта.

11. Записать программу во внутреннюю память МК.

12. Запустить работу программы и выполнить отладку кода.

2.5 Содержание отчета

1. Цель работы.
2. Алгоритм программы управления кнопкой.
3. Листинг программы управления кнопкой.
4. Выводы.

2.6 Контрольные вопросы

1. Какие существуют порты ввода/вывода, какие регистры используются для их управления?
2. Опишите принцип работы приоритетного декодера матрицы.
3. Как осуществляется инициализация портов ввода/вывода?
4. Какие выводы используются в качестве портов ввода/вывода общего назначения, и с помощью каких регистров осуществляется к ним доступ?
5. Для чего применяются алгоритмы гашения дребезга контактов?

Лабораторная работа №3

Работа со встроенным АЦП микроконтроллера

3.1 Цель работы

Изучить АЦП и структуру внутренней памяти данных микроконтроллера Cygnal C8051F320, разработать программу на языке программирования С для аналого-цифрового преобразования сигнала и сохранения этого сигнала в памяти микроконтроллера.

3.2 Теоретические сведения

10-разрядный АЦП (АЦП0)

Модуль АЦП0 МК С8051F320 состоит из двух 17-канальных аналоговых мультиплексоров (обозначаются вместе как AMUX0) и 10-разрядного АЦП последовательного приближения (максимальная производительность – 200 тыс. преобразований в секунду) с интегрированным устройством выборки-хранения (УВХ) и программируемым детектором диапазона. AMUX0, режимы преобразования и детектор диапазона настраиваются программным путем при помощи регистров специального назначения (рисунок 3.1). АЦП0 функционирует как в 1-фазном, так и в дифференциальном режимах, и может быть настроен на измерение напряжения на выводах P1.0–P3.0, выходного напряжения датчика температуры или напряжения VDD относительно P1.0–P3.0, VREF или GND. Модуль АЦП0 включен только тогда, когда бит AD0EN регистра управления АЦП0 (ADC0CN) установлен в 1. Сброс этого бита в 0 переводит АЦП0 в режим отключения с пониженным энергопотреблением.

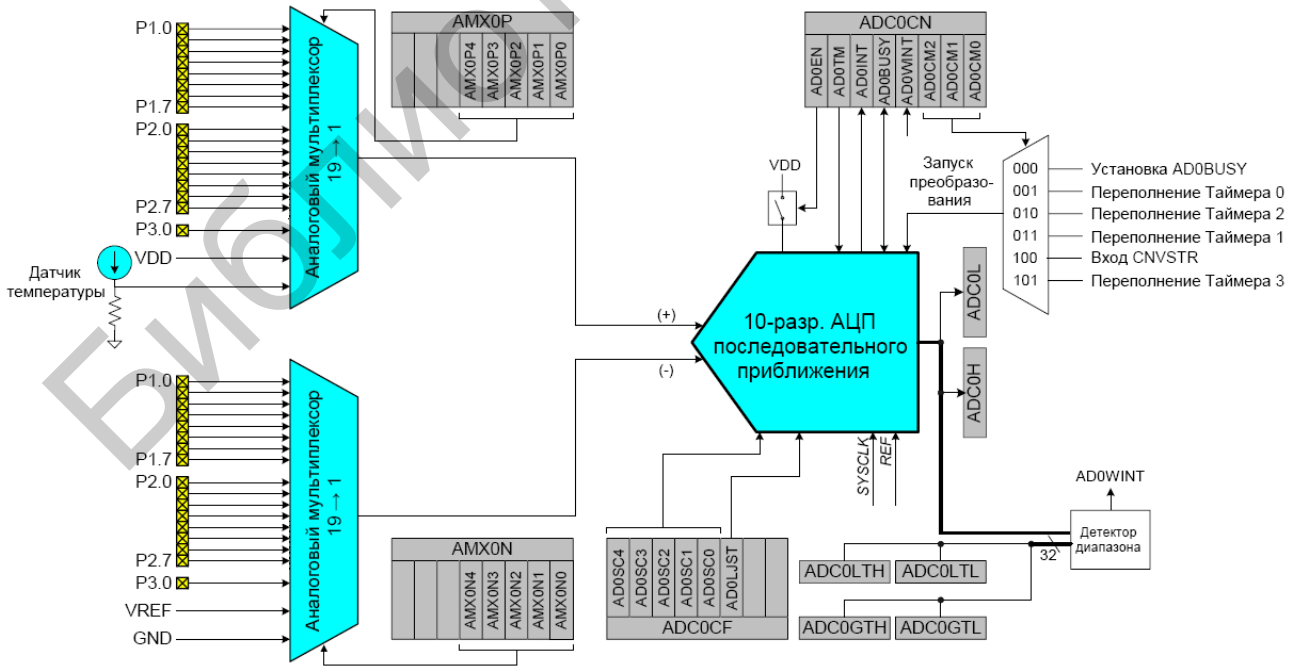

Рисунок 3.1 – Функциональная схема АЦП0

Аналоговый мультиплексор

AMUX0 осуществляет выбор положительного и отрицательного входов АЦП. В качестве положительного входа можно выбрать:

- P1.0–P3.0;
- выходной сигнал встроенного датчика температуры;
- положительное напряжение питания (VDD).

В качестве отрицательного входа можно выбрать:

- P1.0-P3.0;
- VREF;
- общий вывод питания GND.

Если в качестве отрицательного входа выбран GND, то АЦП0 функционирует в 1-фазном режиме; в остальных случаях АЦП0 функционирует в дифференциальном режиме. Входные каналы АЦП0 выбираются в регистрах AMX0P и AMX0N.

Формат получаемого результата преобразования различен для 1-фазного и дифференциального режимов. По окончании каждого преобразования в регистры ADC0H и ADC0L записываются соответственно старший и младший байты результата преобразования. Данные могут быть выровнены вправо или влево в зависимости от значения бита AD0LJST (ADC0CN.0). В 1-фазном режиме результаты преобразований представлены в виде 10-разрядных целых чисел без знака. Диапазон измерения входных напряжений: 0…VREF*1023/1024. В дифференциальном режиме результаты преобразований представлены в виде 10-разрядных чисел в дополнительном коде со знаком. Диапазон измерения входных напряжений: –VREF…VREF*511/512.

Замечание. Выводы порта, выбранные в качестве входов АЦП0, должны быть настроены как аналоговые входы, и должны пропускаться матрицей при назначении выводов. Чтобы настроить вывод порта как аналоговый вход, следует сбросить в 0 соответствующий бит в регистре PnMDIN (для n = 0, 1, 2, 3). Чтобы заставить матрицу пропускать вывод порта при назначении выводов, следует установить в 1 соответствующий бит в регистре PnSKIP (для n = 0, 1, 2). Более подробная информация о настройке порта ввода/вывода приведена в лабораторной работе №1.

Максимальная скорость преобразования АЦП0 – 200 тыс. преобразований в секунду. Частота дискретизации АЦП0 определяется частотой системного тактового сигнала, деленной на значение, задаваемое битами AD0SC регистра ADC0CF, т. е. SYSCLK/(AD0SC + 1) для $0 \le AD0SC \le 31$.

Запуск преобразования может быть осуществлен одним из шести способов в зависимости от состояния битов режима запуска преобразования АЦП0 (AD0CM2-0) в регистре ADC0CN. Преобразование может быть инициировано следующими способами:

1) установкой в 1 бита AD0BUSY в регистре ADC0CN;

2) переполнением Таймера 0 (т. е. непрерывное по времени преобразование);

3) переполнением Таймера 2;

4) переполнением Таймера 1;

5) нарастающим фронтом внешнего входного сигнала CNVSTR (вывод P0.6);

6) переполнением Таймера 3.

Установка в 1 бита AD0BUSY позволяет осуществлять программное управление АЦП0, т. е. выполнять преобразования «по требованию». Бит AD0BUSY устанавливается в 1 во время преобразования и сбрасывается в 0 после окончания преобразования. При сбросе бита AD0BUSY инициируется прерывание (если оно разрешено) и устанавливается флаг прерывания от АЦП0 (AD0INT).

Примечание – При определении окончания преобразования методом опроса следует использовать флаг прерывания от АЦП0 (AD0INT). Преобразованные данные доступны в регистрах старшего и младшего слова данных АЦП0, ADC0H и ADC0L соответственно, когда AD0INT = 1. Следует иметь в виду, что если запуск преобразования осуществляется переполнением Таймера 2 или Таймера 3, то используются переполнения младшего байта, когда Таймер 2/3 работает в 8-разрядном режиме, и переполнения старшего байта, когда Таймер 2/3 работает в 16-разрядном режиме.

Замечание относительно использования CNVSTR. Входной вывод CNVSTR функционирует так же, как вывод порта P0.6. Если вход CNVSTR используется для запуска преобразования АЦП0, то вывод порта P0.6 должен пропускаться матрицей при назначении выводов. Чтобы заставить матрицу пропускать P0.6, следует установить в 1 бит 6 в регистре P0SKIP.

Время установления. Если конфигурация входов АЦП0 изменяется (т. е. изменяются настройки AMUX0), то после этого для обеспечения точности преобразования необходимо выдержать паузу длительностью не менее минимального времени установления сигнала. Время установления определяется сопротивлением AMUX0, емкостью накопительного конденсатора УВХ, сопротивлением внешнего источника сигнала и требуемой точностью преобразования. Следует отметить, что в энергосберегающем режиме выборки-хранения после запуска каждого преобразования выборка длится три периода сигнала дискретизации АЦП. Для большинства приложений эти три периода сигнала дискретизации будут соответствовать требованиям, предъявляемым ко времени установления.

Необходимые регистры АЦП перечислены в таблице 3.1 и подробно описаны в приложении Б (подраздел Б.4).

Таблица 3.1– Регистры АЦП

AMX0P	Регистр выбора положительного канала AMUX0
AMX0N	Регистр выбора отрицательного канала AMUX0
ADC0CF	Регистр конфигурации АЦП0
ADC0H	Регистр старшего байта слова данных АЦП0
ADC0L	Регистр младшего байта слова данных АЦП0
ADC0CN	Регистр управления АЦП0

3.3 Приборы и оборудования, применяемые в работе

В данной лабораторной работе используются:

1. Комплект разработки для микроконтроллера C8051F320.
2. Персональный компьютер с USB-портом.
3. Генератор.
4. Программное обеспечение.

3.4 Порядок выполнения работы

1. Запустить среду разработки программ для микроконтроллеров SiLabs.
2. Подготовить комплект отладки и ПК. Подключить плату к программатору, подключить программатор к компьютеру по интерфейсу USB. Подать питание на плату с помощью 9В блока питания. Установить связь МК с компьютером.
3. Включить файл main_ADC.c в проект.
4. Провести инициализацию микроконтроллера:

- инициализация внутреннего кварцевого резонатора (регистр OSCICN);
- инициализация порта (регистры XBR0, XBR1, PnMDIN, PnMDOUT, PnSKIP) для подключения светодиода и кнопки;
- инициализация таймеров 0 и 1 (регистры TCON, TMOD) в реж. 1 и 2;
- инициализация прерывания (регистры IE, IP) от таймера-счетчика;
- инициализация АЦП (регистры AMX0P, AMX0N, ADC0CN, ADC0CF).

5. Установить значения в таймеры для генерации сигнала необходимой частоты (регистры TH0, TL0).
6. Запустить работу программы (кнопка Run среды).
7. Разработать алгоритм управления кнопкой (в соответствии с заданием).
8. Написать программу управления кнопкой, включающую подпрограмму гашения дребезга контактов.
9. Скомпилировать программу.
10. Выполнить построение проекта.
11. Записать программу во внутреннюю память МК.
12. Запустить работу программы (кнопка Run среды SiLabs).
13. Подать на АЦП сигнал с потенциометра.
14. Выполнить аналого-цифровое преобразование сигнала с частотой дискретизации 5 Гц.
15. Изменять частоту мерцания светодиода в зависимости от значения напряжения на выводе потенциометра (см. таблицу 1.8).

3.5 Содержание отчета

1. Цель работы.
2. Алгоритм программы управления АЦП.
3. Листинг программы управления АЦП.
4. Выводы.

3.6 Контрольные вопросы

1. Перечислите основные составляющие модуля АЦП микроконтроллера C8051F320.
2. В каких режимах функционирует АЦП, в чем их отличия?
3. Как осуществляется настройка входов АЦП и какие регистры при этом используются?
4. Как может быть инициировано преобразование? Назовите основные особенности каждого способа.
5. Какова структура памяти микроконтроллера? Назовите основные регистры, необходимые для работы с памятью данных.

Лабораторная работа №4

Генерация аналоговых сигналов посредством цифроаналогового преобразователя

4.1 Цель работы

Приобрести практические навыки генерации аналоговых сигналов посредством цифроаналогового преобразователя.

4.2 Теоретические сведения

МК C8051F330 содержат 10-разрядный ЦАП с токовым выходом (ЦАП0) (рисунок 4.1). Максимальный выходной ток этого ЦАП можно настроить, выбрав любое из следующих трех значений: 0,5, 1 или 2 мА.

ЦАП0 можно включать или отключать с помощью бита IDA0EN в регистре управления ЦАП0 IDA0CN. Если IDA0EN = 0, то вывод порта IDA0 (P0.1) функционирует как обычный порт ввода/вывода общего назначения (GPIO). Если IDA0EN = 1, то у вывода IDA0 автоматически отключаются драйвер цифрового выхода и слаботоковая подтяжка, а сам вывод подключается к выходу ЦАП0. Внутренний стабилизированный генератор смещения используется для генерации опорного тока ЦАП0, независимо от того, включен ЦАП0 или нет. При использовании ЦАП0 необходимо настроить матрицу таким образом, чтобы она пропускала вывод IDA0 при назначении выводов: для этого следует установить в 1 соответствующий бит в регистре P0SKIP.

Обновление выходного сигнала ЦАП0

ЦАП0 отличает гибкий механизм обновления выходного сигнала, который позволяет плавно («бесшовно») изменять выходной сигнал во всем диапазоне выходных значений и поддерживает обновление выходного сигнала без джиттера (т. е. без фазовых искажений). Возможны три режима обновления выходного сигнала ЦАП:

1) при записи в регистр IDA0H;
2) при переполнении таймера;
3) синхронно с фронтом сигнала на внешнем выводе.

Обновление выходного сигнала «по требованию»

В режиме по умолчанию (IDA0CN.[6:4] = ‘111’) выходной сигнал ЦАП обновляется «по требованию» при записи регистра старшего байта данных ЦАП0 (IDA0H). Необходимо иметь в виду, что при записи регистра младшего байта данных (IDA0L) записываемое значение удерживается, но не влияет на состояние выхода ЦАП0 до тех пор, пока не произойдет запись в регистр IDA0H. Если в регистры данных ЦАП0 необходимо загрузить полное 10-разрядное слово, то оно должно записываться в регистры младшего (IDA0L) и старшего (IDA0H) байтов данных. Данные (оба байта) «защелкиваются» в ЦАП0 только после записи регистра IDA0H, поэтому, если требуется получить полную 10-разрядную точность, последовательность записи должна быть сле-

дующей: сначала IDA0L, затем IDA0H. ЦАП0 может использоваться и в 8-разрядном режиме: для этого необходимо инициализировать регистр IDA0L требуемым значением (обычно 0x00) и записывать данные только в регистр IDA0H.

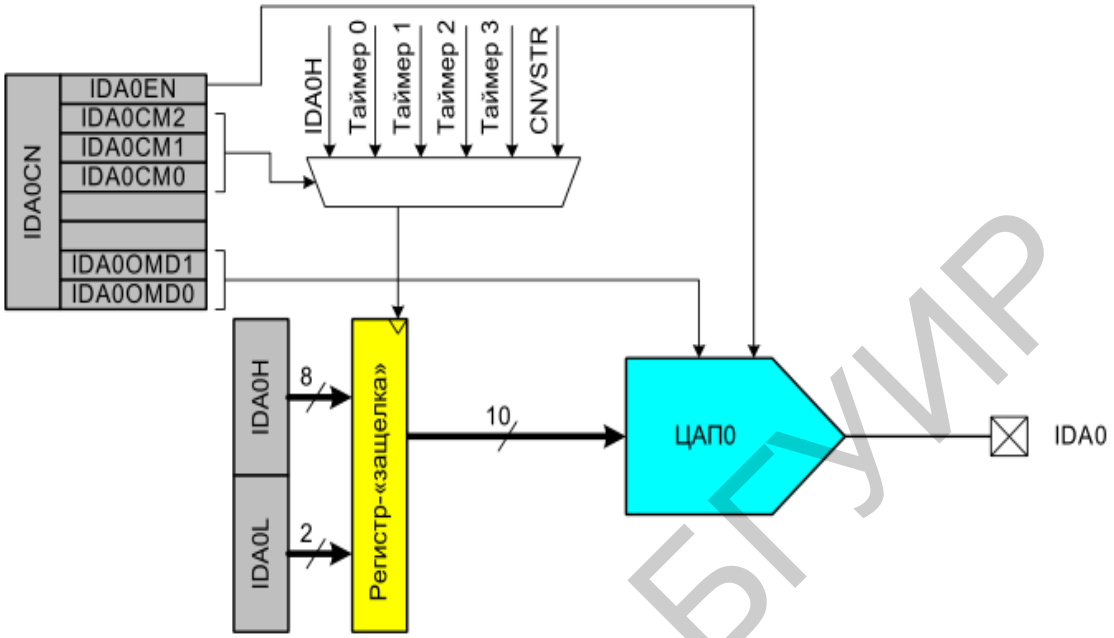

Рисунок 4.1 – Структурная схема ЦАП0

Обновление выходного сигнала при переполнении таймера

Запуск преобразования АЦП0 может осуществляться при переполнении таймера, независимо от состояния процессора. Для обновления выходного сигнала ЦАП также можно использовать переполнение таймера. Эта функция чрезвычайно полезна в системах, в которых ЦАП используется для генерации сигнала с жестко заданной частотой выборки, т. к. задержка реакции на прерывание и время выполнения команд не будут влиять на временные параметры выходного сигнала ЦАП. Если состояние битов IDA0CM (IDA0CN.[6:4]) равно ‘000’, ‘001’, ‘010’ или ‘011’, то при записи обоих регистров данных ЦАП (IDA0L и IDA0H) записываемые значения удерживаются до момента переполнения соответствующего таймера (Таймера 0, Таймера 1, Таймера 2 или Таймера 3 соответственно). В момент переполнения содержимое регистров IDA0H:IDA0L копируется во входные защелки ЦАП, вызывая тем самым обновление его выходного сигнала.

Обновление выходного сигнала по фронту CNVSTR

ЦАП0 можно настроить таким образом, чтобы его выходной сигнал обновлялся по переднему фронту, по заднему фронту или по обоим фронтам сигнала на внешнем выводе CNVSTR. Если состояние битов IDA0CM (IDA0CN.[6:4]) равно ‘100’, ‘101’ или ‘110’, то при записи регистров данных ЦАП (IDA0L и IDA0H) записываемые значения удерживаются до момента обнаружения фронта сигнала на внешнем входе CNVSTR. Конкретные значения битов IDA0CM определяют, по какому фронту CNVSTR (по переднему, по зад-

нему или по обоим) будет происходить обновление выходного сигнала ЦАП. При обнаружении соответствующего фронта содержимое регистров IDA0H:IDA0L копируется во входные защелки ЦАП, вызывая тем самым обновление его выходного сигнала.

Форматирование входных данных ЦАП

Регистры данных ЦАП (IDA0L и IDA0H) выровнены влево. Это означает, что старшие 8 бит слова данных ЦАП отображаются на биты 7–0 регистра IDA0H, а младшие 2 бита слова данных ЦАП отображаются на биты 7 и 6 регистра IDA0L. Отображение слова данных для ЦАП0 показано на рисунке 4.2.

IDA0H								IDA0L							
D9	D8	D7	D6	D5	D4	D3	D2	D1	D0						

Слово данных ЦАП0 (D9 – D0)	Выходной ток в зависимости от значения битов IDA0OMD[1:0]		
	‘1x’ (2 мА)	‘01’ (1 мА)	‘00’ (0.5 мА)
0x000	0 мА	0 мА	0 мА
0x001	1/1024 × 2 мА	1/1024 × 1 мА	1/1024 × 0.5 мА
0x200	512/1024 × 2 мА	512/1024 × 1 мА	512/1024 × 0.5 мА
0x3FF	1023/1024 × 2 мА	1023/1024 × 1 мА	1023/1024 × 0.5 мА

Рисунок 4.2 – Форматирование слова данных ЦАП0

Значение выходного тока полной шкалы ЦАП выбирается с помощью битов IDA0MD (IDA0CN[1:0]).

По умолчанию максимальное значение выходного тока устанавливается равным 2 мА. С помощью битов IDA0MD можно также выбрать значения 0,5 или 1 мА.

4.3 Приборы и оборудования, применяемые в работе

В данной лабораторной работе используются:

1. Комплект разработки для микроконтроллера C8051F320.
2. Персональный компьютер с USB-портом.
3. Программное обеспечение.

4.4 Порядок выполнения работы

1. Запустить среду разработки программ для микроконтроллеров SiLabs.
2. Подготовить комплект отладки и ПК. Подключить плату к программатору, подключить программатор к компьютеру по интерфейсу USB. Подать питание на плату с помощью 9В блока питания. Установить связь МК с компьютером.
3. Включить программу main_DAC.c в проект.
4. Провести инициализацию микроконтроллера:

- инициализация внутреннего кварцевого резонатора (регистр OSCICN);

- инициализация порта (регистры XBR0, XBR1, PnMDIN, PnMDOUT, PnSKIP) для подключения светодиода и кнопки;

- инициализация таймеров 0 и 1 (регистры TCON, TMOD) в реж. 1 и 2;

- инициализация прерывания (регистры IE, IP) от таймера-счетчика;

- инициализация АЦП (регистры AMX0P, AMX0N, ADC0CN, ADC0CF);

- инициализация ЦАП (регистр (IDA0CN).

5. Установить значения в таймеры для генерации сигнала необходимой частоты (регистры TH0, TL0).

6. Разработать алгоритм программы, включающий управление аналого-цифровым преобразованием (в соответствии с заданием), сохранение результатов в памяти микроконтроллера, генерацию синусоидальной волны с частотой, выбираемой на основе значений, полученных в регистрах АЦП.

7. Написать программу, реализующую разработанный алгоритм.

8. Скомпилировать программу.

9. Выполнить построение проекта.

10. Записать программу во внутреннюю память МК.

11. Запустить работу программы (кнопка Run).

12. Подать на АЦП сигнал со встроенного потенциометра.

13. Выполнить аналого-цифровое преобразование сигнала.

14. Настроить генерацию синусоидального сигнала посредством встроенного ЦАП.

4.5 Содержание отчета

1. Цель работы.
2. Листинг программы.
3. Выводы.

4.6 Контрольные вопросы

1. Определите назначение ЦАП.
2. Назовите основные регистры, используемые модулем ЦАП.
3. Дайте описание битов регистра управления ЦАП (IDA0CN).
4. Назовите особенности выбора частоты дискретизации при генерации цифрового сигнала.
5. Каковы особенности выхода ЦАП по напряжению и по току? Какой тип выхода реализован во встроенном в МК ЦАП?
6. Назовите обновления сигнала на выходе ЦАП.

Лабораторная работа №5

Работа с универсальным асинхронным приемопередатчиком UART

5.1 Цель работы

Приобрести практические навыки установки связи между ПК и МК посредством порта UART. Определить скорость передачи данных.

5.2 Теоретические сведения

Порт UART0 представляет собой асинхронный полнодуплексный последовательный порт, способный работать в режимах 1 и 3 стандартного (для архитектуры 8051) UART (рисунок 5.1). Поддержка усовершенствованного режима генерации скорости передачи данных позволяет использовать для генерации стандартных скоростей обмена различные источники тактирования. Буферизация принимаемых данных позволяет UART0 начать прием второго входящего байта данных до того, как программа закончит чтение предыдущего байта данных.

С работой UART0 связаны следующие регистры специального назначения: регистр управления UART0 (SCON0) и буфер данных UART0 (SBUF0). Одна и та же ячейка памяти, адресуемая как SBUF0, обеспечивает доступ и к регистру передатчика, и к регистру приемника. Операции записи в SBUF0 всегда обращаются к регистру передатчика. Операции чтения из SBUF0 всегда обращаются к буферизованному регистру приемника; невозможно прочитать данные из регистра передатчика.

Если прерывания от модуля UART0 разрешены, то запрос прерывания генерируется каждый раз при завершении передачи байта данных (установка в 1 флага TI0 в регистре SCON0) или при получении байта данных (установка в 1 флага RI0 в регистре SCON0). Флаги прерываний от UART0 не сбрасываются аппаратно при переходе к процедуре обслуживания прерывания. Они должны сбрасываться программно. Это позволяет программе определить причину, вызвавшую прерывание от UART0 (завершение передачи или завершение приема).

5.2.1 Усовершенствованный режим генерации скорости передачи данных

Скорость передачи данных UART0 генерируется Таймером 1, работающим в 8-разрядном режиме с автоперезагрузкой. Частота передатчика (Таймер TX) определяется переполнением регистра TL1, частота приемника – переполнением регистра-копии регистра TL1 (обозначенного как «RX-Таймер» на рисунке 5.2), который недоступен из программы пользователя. Скорость передачи данных передатчика и приемника равна деленной на два частоте переполнения регистров TL1 и RX-Таймер соответственно. RX-Таймер работает тогда, когда включен Таймер 1 и использует то же самое значение перезагрузки (TH1). Од-

нако перезагрузка регистра RX-Таймер происходит в тот момент, когда на выводе RX обнаруживается событие START. Это позволяет начать прием данных в любой момент при обнаружении события START, независимо от состояния Таймера TX.

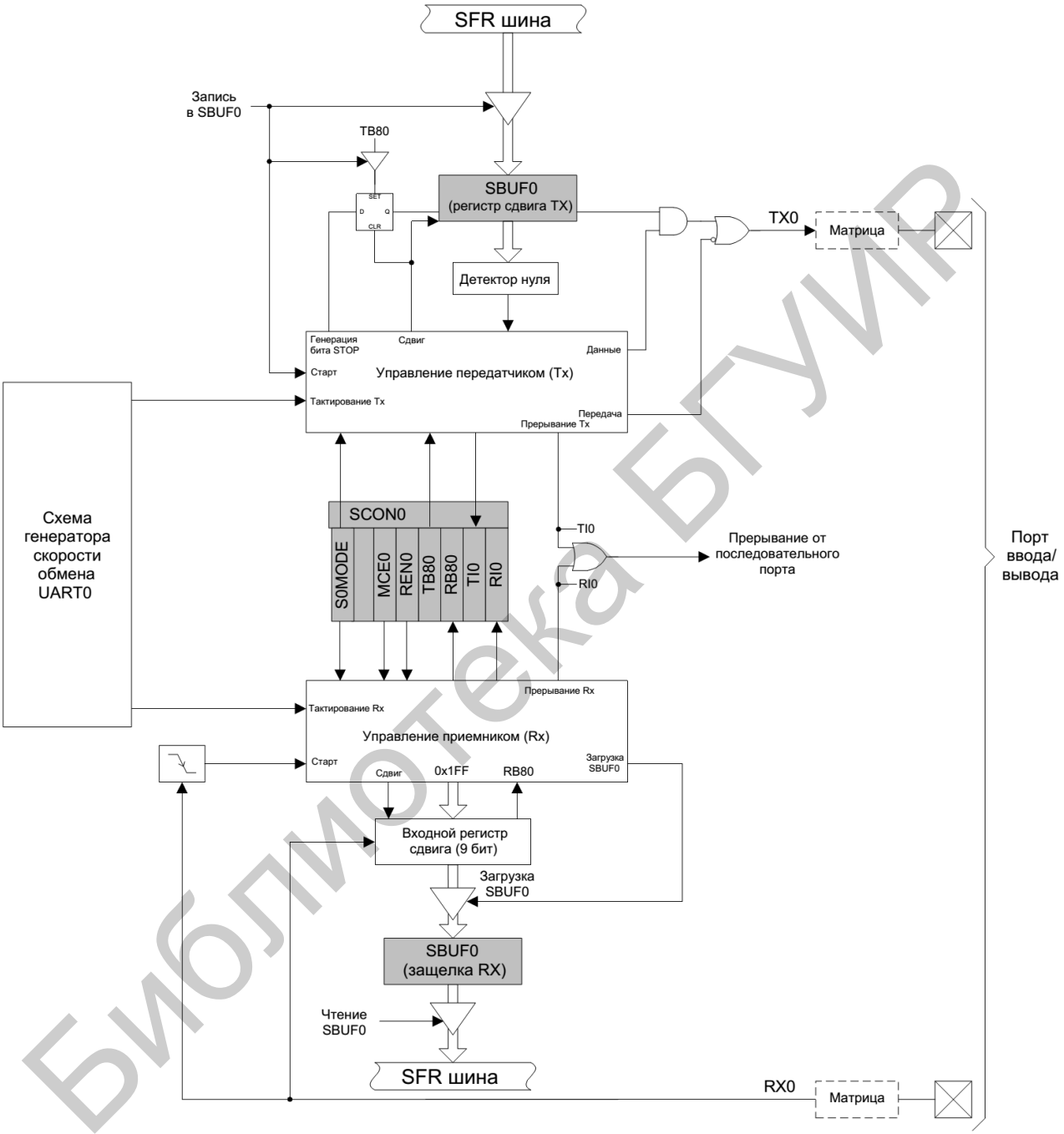

Рисунок 5.1 – Структурная схема UART0

Таймер 1 следует настроить для работы в режиме 2, т. е. как 8-разрядный таймер с автоперезагрузкой. Значение перезагрузки Таймера 1 следует установить таким образом, чтобы частота переполнений таймера была в два раза больше необходимой скорости передачи данных. Частота тактового сигнала Таймера 1 может быть одной из следующих:

1) SYSCLK;
2) SYSCLK/4;
3) SYSCLK/12;
4) SYSCLK/48;
5) частота внешнего генератора/8;
6) частота внешнего сигнала на входе T1.

Скорость передачи данных UART0 определяется из равенства

$$V_{\text{UART0}} = \frac{F_{\text{T1}}}{2}, \tag{5.1}$$

где V_{UART0} – скорость передачи данных UART0;

F_{T1} – частота переполнения Таймера 1.

Расчет частоты переполнения Таймера 1 F_{T1} производим по следующей формуле:

$$F_{\text{T1}} = \frac{\text{CLK}_{\text{T1}}}{256 - \text{H}_{\text{T1}}}, \tag{5.2}$$

где CLK_{T1} – частота тактирования Таймера 1;

H_{T1} – старший байт Таймера 1 (8-разрядное значение перезагрузки).

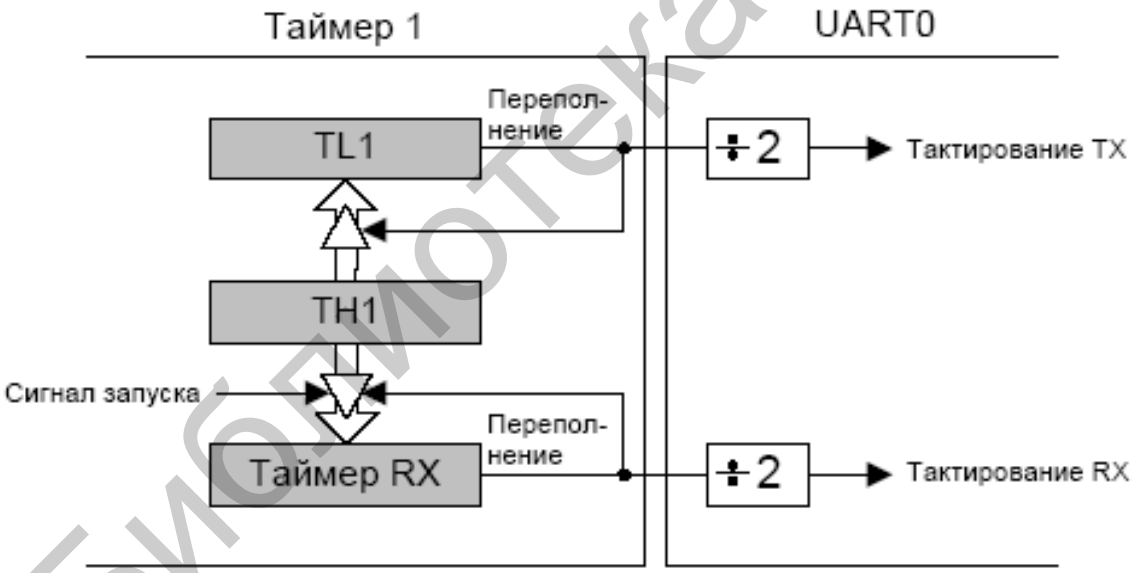

Рисунок 5.2 – Логика генератора скорости передачи данных UART0

Следует отметить, что внутренний генератор может генерировать системный тактовый сигнал, в то время как выходной сигнал внешнего генератора подается на Таймер 1. Примеры настройки частоты работы UART0 при тактирующем сигнале частотой 11,0592 МГц приведены в таблице 5.1.

Таблица 5.1 – Параметры настройки таймера для стандартных скоростей передачи данных при тактировании от внутреннего генератора (11,0592 МГц)

Требуемая скорость передачи данных (бит/с)	Погрешность установки скорости передачи данных	Коэффициент деления генератора	Частота сигнала тактирования	SCA1-SCA0 (выбор коэффициента предварительного деления)*	TIM*	Значение перезагрузки Таймера 1
230400	0,00 %	48	SYSCLK	XX	1	0xE8
115200	0,00 %	96	SYSCLK	XX	1	0xD0
57600	0,00 %	192	SYSCLK	XX	1	0xA0
28800	0,00 %	384	SYSCLK	XX	1	0x40
14400	0,00 %	768	SYSCLK/12	00	0	0xE0
9600	0,00 %	1152	SYSCLK/12	00	0	0xD0
2400	0,00 %	4608	SYSCLK/12	00	0	0x40
1200	0,00 %	9216	SYSCLK/48	10	0	0xA0

Примечание – X обозначает «не имеет значения».

5.2.2 Режимы работы UART0

UART0 обеспечивает стандартный асинхронный полнодуплексный обмен данными. Режим работы UART0 (8- или 9-разрядный) выбирается при помощи бита S0MODE (SCON0.7). Типичные варианты использования UART приведены на рисунке 5.3.

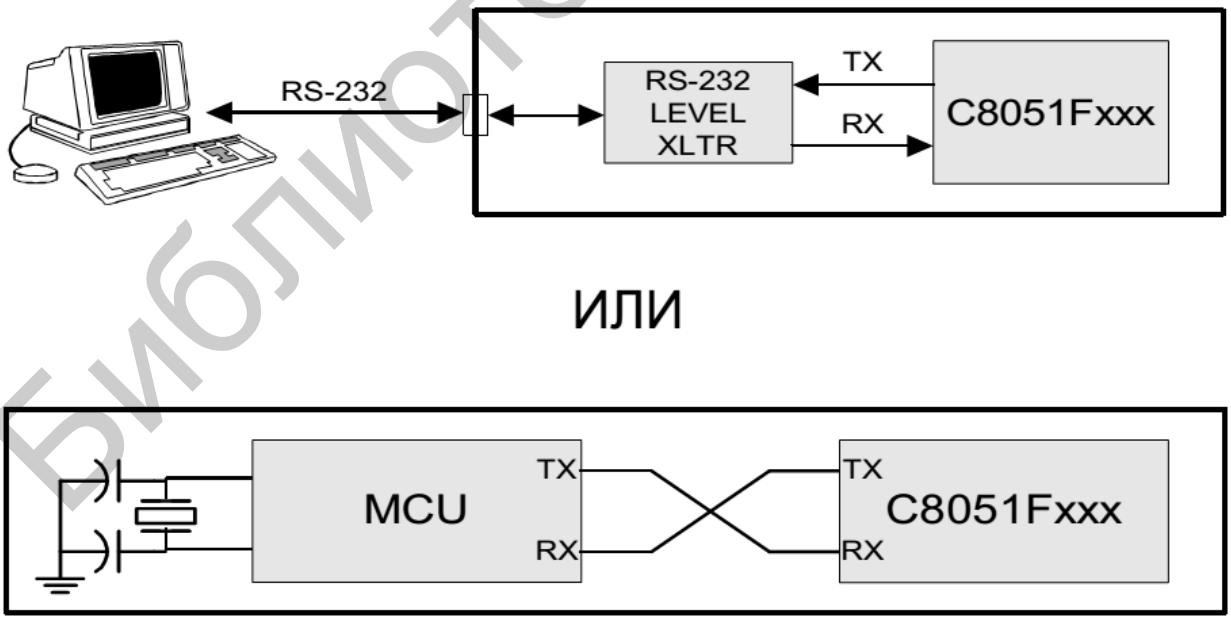

Рисунок 5.3 – Примеры использования UART

8-разрядный UART

В режиме 8-разрядного UART для передачи одного байта данных используются 10 бит: 1 стартовый бит, 8 бит данных (МЗР вперед) и 1 стоповый бит.

Данные передаются МЗР вперед через внешний вывод TX0 и принимаются через внешний вывод RX0 (рисунок 5.4). При приеме в регистре SBUF0 сохраняются 8 бит данных, а бит RB80 (SCON0.2) принимает значение стопового бита.

Передача данных начинается, когда происходит запись байта данных в регистр SBUF0. Флаг прерывания от передатчика TI0 (SCON0.1) устанавливается в 1 в конце передачи (в начале передачи стопового бита). Прием данных может быть начат в любое время после установки в 1 флага включения приемника REN0 (SCON0.4). После приема стопового бита байт данных будет загружен в регистр приемника SBUF0, если соблюдаются следующие условия: RI0 должен быть равен лог. 0, и если MCE0 = 1, то стоповый бит должен быть равен лог. 1. В случае переполнения данных при приеме первые принятые 8 бит защелкиваются в регистре приемника SBUF0, а следующие данные, вызвавшие переполнение, теряются.

Если эти условия соблюдаются, то 8 бит данных сохраняются в регистре SBUF0, стоповый бит сохраняется в бите RB80 и устанавливается в 1 флаг RI0. Если эти условия не соблюдаются, то SBUF0 и RB80 не будут загружаться и флаг RI0 не устанавливается. При установке флагов TI0 или RI0 будет сгенерировано прерывание, если оно разрешено.

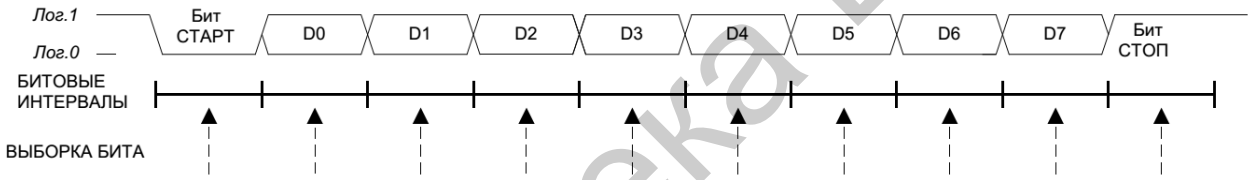

Рисунок 5.4 – Временные диаграммы в режиме 8-разрядного UART

5.3 Приборы и оборудования, применяемые в работе

В данной лабораторной работе используются:

1. Комплект разработки для микроконтроллера C8051F320.
2. Персональный компьютер с USB-портом.
3. Программное обеспечение.

5.4 Порядок выполнения работы

1. Запустить среду разработки программ для микроконтроллеров SiLabs.

2. Подготовить комплект отладки и ПК. Подключить плату к программатору, подключить программатор к компьютеру по интерфейсу USB. Подать питание на плату с помощью 9В блока питания. Установить связь МК с компьютером.

3. Включить программу main_UART.c в проект.

4. Провести инициализацию микроконтроллера:

- инициализация внутреннего кварцевого резонатора (регистр OSCICN);

- инициализация порта (регистры XBR0, XBR1, PnMDIN, PnMDOUT, PnSKIP) для подключения светодиода и кнопки;

- инициализация таймеров 0 и 1 (регистры TCON, TMOD) в реж. 1 и 2;

- инициализация прерывания (регистры IE, IP) от таймер/счетчика;

- инициализация АЦП (регистры AMX0P, AMX0N, ADC0CN, ADC0CF);

- инициализация UART (регистр SCON0).

5. Установить значения в таймеры для генерации сигнала необходимой частоты (регистры TH0, TL0).

6. Разработать алгоритм программы, включающий управление аналого-цифровым преобразованием (в соответствии с заданием), сохранение результатов в памяти микроконтроллера, передачу данных с помощью универсального асинхронного приемопередатчика в ПК.

7. Написать программу, реализующую разработанный алгоритм.

8. Скомпилировать программу.

9. Выполнить построение проекта.

10. Записать программу во внутреннюю память МК.

11. Запустить работу программы (кнопка Run).

12. Подать на АЦП сигнал с частотой, соответствующей варианту задания.

13. Выполнить аналого-цифровое преобразование сигнала с частотой дискретизации, рассчитанной соответственно частоте входного сигнала, и сохранить 25 байт данных в память МК.

14. Передать полученные данные на ПК.

5.5 Содержание отчета

1. Цель работы.
2. Листинг программы
3. Выводы.

5.6 Контрольные вопросы

1. Определите назначение UART.
2. Назовите основные регистры, используемые портом UART.
3. Дайте описание битов регистра управления UART0 (SCON0).
4. Объясните назначение генератора скорости передачи данных в UART.
5. Назовите основные параметры, определяющие скорость передачи данных.
6. Назовите режимы работы UART и их особенности.

Лабораторная работа №6

Реализация широтно-импульсной модуляции с помощью программируемого массива счетчиков

6.1 Цель работы

Приобрести практические навыки работы с широтно-импульсной модуляцией (ШИМ) сигнала, а также с модулем программируемого массива счетчиков, научиться обоснованно подходить к выбору режимов работы модулей захвата/сравнения.

6.2 Теоретические сведения

Программируемый массив счетчиков (ПМС) реализует расширенные таймерные функции, при этом требует меньшего вмешательства со стороны процессорного ядра, чем стандартные таймеры-счетчики архитектуры 8051. ПМС состоит из специального 16-разрядного таймера-счетчика и пяти 16-разрядных модулей захвата/сравнения. Каждый модуль захвата/сравнения имеет свою собственную линию ввода/вывода (CEXn), которая через матрицу соединяется, если разрешено, с портом ввода/вывода.

Таймер-счетчик тактируется программируемым внутренним сигналом, в качестве которого могут использоваться:

- внутренний сигнал с частотой, равной системной тактовой частоте;
- внутренний сигнал с частотой, равной 1/4 системной тактовой частоты;
- внутренний сигнал с частотой, равной 1/12 системной тактовой частоты;
- сигнал от внешнего генератора, деленный по частоте на 8;
- переполнение Таймера 0;
- входной сигнал на внешнем выводе ECI.

Каждый модуль захвата/сравнения можно независимо настроить для работы в одном из шести режимов:

- инициируемый по фронту сигнала захват;
- программный таймер;
- высокоскоростной выход;
- выход заданной частоты;
- 8-разрядный широтно-импульсный модулятор;
- 16-разрядный широтно-импульсный модулятор.

В общем виде структура ПМС представлена на рисунке 6.1.

Таймер-счетчик модуля ПМС

16-разрядный таймер-счетчик модуля ПМС состоит из двух 8-разрядных SFR регистров: PCA0L и PCA0H. Регистр PCA0H является старшим байтом (СЗБ) 16-разрядного таймера-счетчика, а регистр PCA0L образует младший байт (МЗБ). При чтении регистра PCA0L значение регистра PCA0H автоматически фиксируется в регистре-защелке; последующее чтение регистра PCA0H возвратит данные именно из этого регистра-защелки. Таким образом, для обес-

печения точности считывания полного 16-разрядного значения таймера-счетчика ПМС необходимо сначала прочитать регистр PCA0L, а затем – регистр PCA0H.

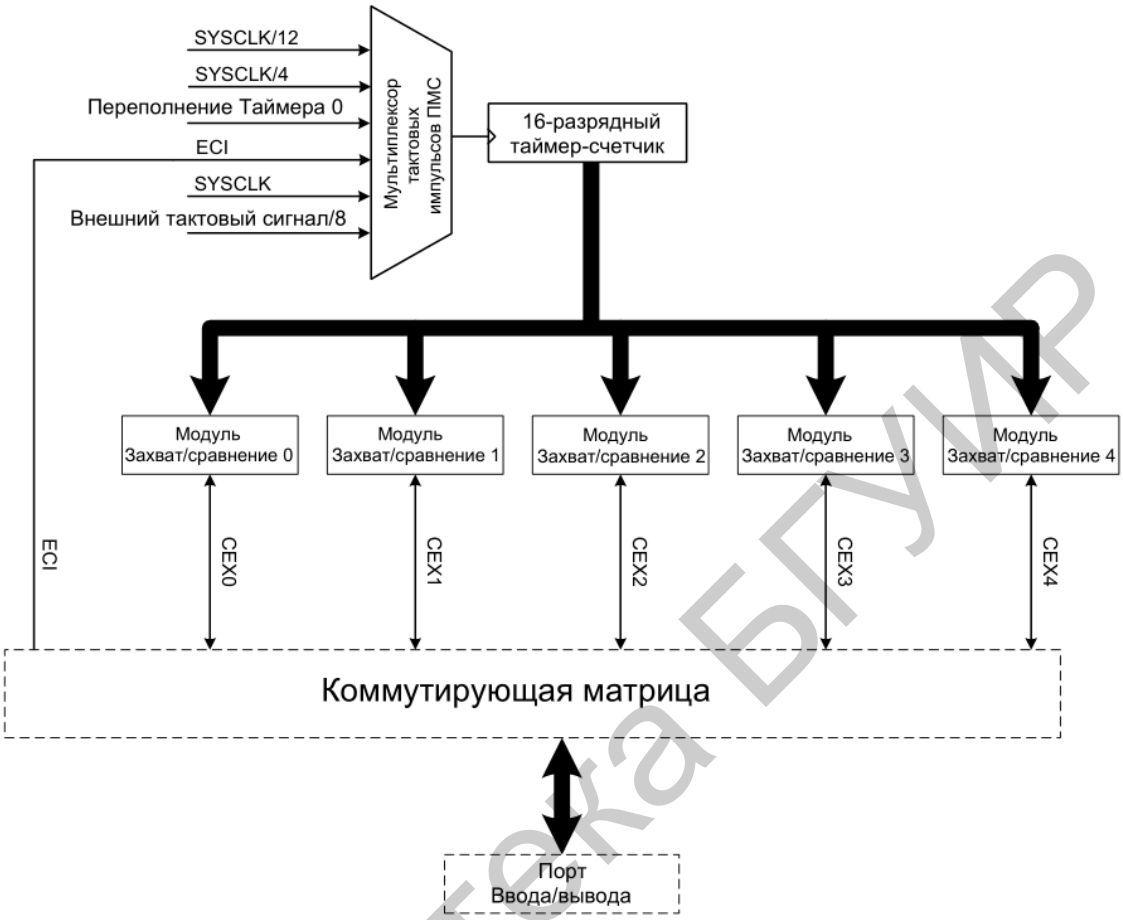

Рисунок 6.1 – Структура программируемого массива счетчиков

Модули захвата/сравнения

Каждый модуль можно настроить независимо для работы в одном из шести режимов: инициируемый по фронту сигнала захват, программный таймер, высокоскоростной выход, выход заданной частоты, 8-разрядный широтно-импульсный модулятор и 16-разрядный широтно-импульсный модулятор. Каждый модуль имеет связанные с ним регистры специального назначения, которые используются для обмена данными с модулем и для настройки режимов работы.

В таблице 6.1 приведены комбинации битов в регистрах PCA0CPMn, используемые для перевода модулей захвата/сравнения в различные режимы работы. Установка в 1 бита ECCFn в регистрах PCA0CPMn разрешает генерацию прерываний при установке в 1 флагов CCFn регистра PCA0CN. Следует иметь в виду, что индивидуальные CCFn прерывания распознаются только в том случае, если прерывания от модуля ПМС разрешены глобально. Прерывания от ПМС разрешаются глобально установкой в 1 битов EA (IE.7) и EPCA0 (EIE1.3). Схема формирования прерываний от модуля ПМС приведена на рисунке 6.2.

Таблица 6.1 – Комбинации битов в регистрах PCA0CPMn

PWM16	ECOM	CAPP	CAPN	MAT	TOG	PWM	ECCF	Operation Mode
x	x	1	0	0	0	0	x	Захват инициируется положительным фронтом сигнала на линии CEXn
x	x	0	1	0	0	0	x	Захват инициируется отрицательным фронтом сигнала на линии CEXn
x	x	1	1	0	0	0	x	Захват инициируется изменением сигнала на линии CEXn
x	1	0	0	1	0	0	x	Программный таймер
x	1	0	0	1	1	0	x	Высокоскоростной выход
x	1	0	0	x	1	1	x	Выход заданной частоты
0	1	0	0	x	0	1	x	8-разр. широтно-импульсный модулятор
1	1	0	0	x	0	1	x	16-разр. широтно-импульсный модулятор

Режим 8-разрядного широтно-импульсного модулятора

Каждый модуль захвата/сравнения можно использовать независимо от других для генерации на соответствующем ему выводе CEXn выходного сигнала с широтно-импульсной модуляцией. Частота этого выходного сигнала зависит от частоты сигнала тактирования таймера-счетчика ПМС. Для изменения коэффициента заполнения (скважности) выходного широтно-импульсного сигнала используется регистр захвата/сравнения PCA0CPLn соответствующего модуля. Когда значение младшего байта таймера-счетчика ПМС (PCA0L) становится равным значению регистра PCA0CPLn, на внешнем выводе CEXn устанавливается сигнал высокого уровня. Когда регистр PCA0L переполнится, на выводе CEXn установится сигнал низкого уровня.

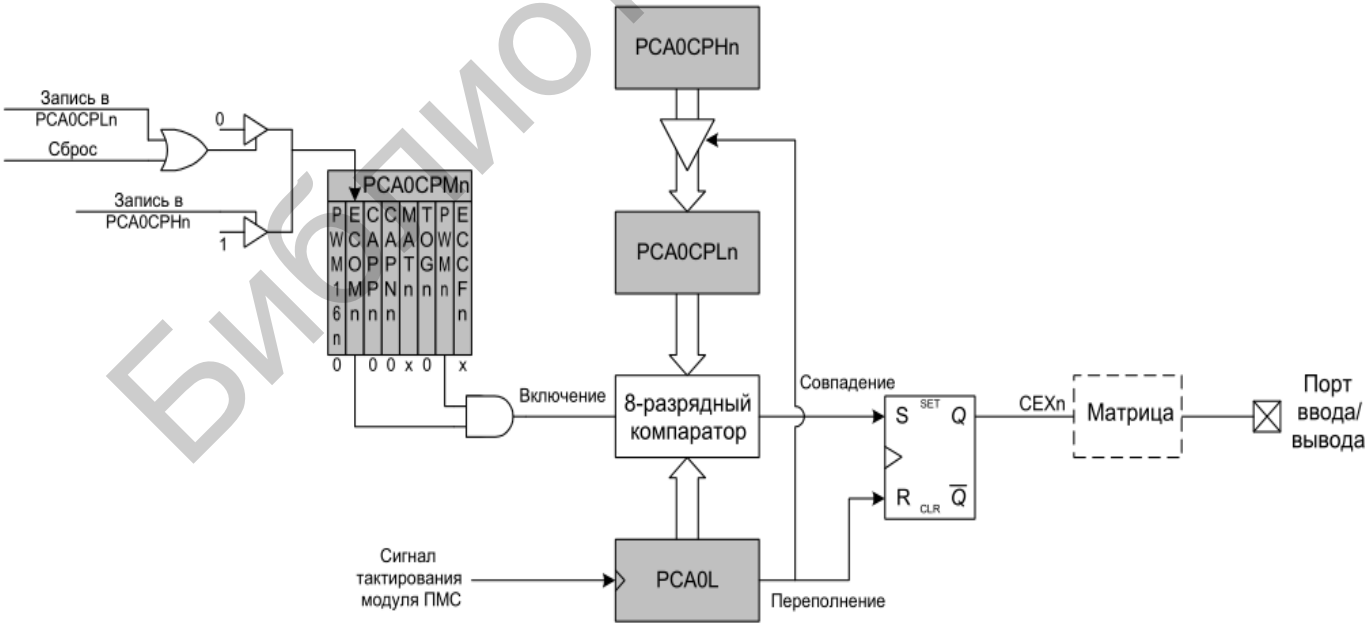

Рисунок 6.2 – Схема формирования прерываний от модуля ПМС

Кроме этого, при переполнении младшего байта таймера-счетчика (PCA0L) из состояния 0xFF в состояние 0x00 регистр PCA0CPLn автоматически перезагружается значением, хранящимся в регистре PCA0CPHn, без вмешательства со стороны программы. Во избежание сбоев в работе цифрового компаратора рекомендуется осуществлять запись в регистр PCA0CPHn, а не в регистр PCA0CPLn. Режим 8-разрядного широтно-импульсного модулятора включается установкой в 1 битов ECOMn и PWMn регистра PCA0CPMn. Скважность выходного сигнала в режиме 8-разрядного ШИМ определяется уравнением (6.1).

Замечание. При записи 16-разрядного значения в регистры захвата/сравнения модуля ПМС младший байт всегда необходимо записывать первым. Запись в регистр PCA0CPLn сбрасывает в 0 бит ECOMn; запись в регистр PCA0CPHn устанавливает в 1 бит ECOMn.

$$DutyCycle = (256 - PCA0CPHn) / 256. \quad (6.1)$$

6.3 Приборы и оборудования, применяемые в работе

В данной лабораторной работе используются:

1. Комплект разработки для микроконтроллера C8051F320.
2. Персональный компьютер с USB-портом.
3. Программное обеспечение.

6.4 Порядок выполнения работы

1. Запустить среду разработки программ для микроконтроллеров SiLabs.

2. Подготовить комплект отладки и ПК. Подключить плату к программатору, подключить программатор к компьютеру по интерфейсу USB. Подать питание на плату с помощью 9В блока питания. Установить связь МК с компьютером.

3. Включить программу main_PCA.c в проект.

4. Провести инициализацию микроконтроллера:

- инициализация внутреннего кварцевого резонатора (регистр OSCICN);
- инициализация порта (регистры XBR0, XBR1, PnMDIN, PnMDOUT, PnSKIP) для подключения светодиода и кнопки;
- инициализация прерывания (регистры IE, IP) от таймера-счетчика;
- инициализация АЦП (регистры AMX0P, AMX0N, ADC0CN, ADC0CF);
- инициализация PCA (регистры PCA0CN, PCA0MD, PCA0CPMn).

5. Установить значения в модуль захвата сравнения для генерации сигнала ШИМ необходимой частоты (регистры PCA0H, PCA0L).

6. Разработать алгоритм программы, включающий управление аналого-цифровым преобразованием (в соответствии с заданием), сохранение результатов в памяти микроконтроллера, генерацию сигнала ШИМ в соответствии со значениями, полученными с помощью АЦП.

7. Написать программу, реализующую разработанный алгоритм.
8. Скомпилировать программу.
9. Выполнить построение проекта.
10. Записать программу во внутреннюю память МК.
11. Запустить работу программы (кнопка Run среды SiLabs).

6.5 Содержание отчета

1. Цель работы.
2. Листинг программы
3. Выводы.

6.6 Контрольные вопросы

1. Определите назначение программируемого массива счетчиков (PCA).
2. Назовите основные регистры, используемые модулем PCA.
3. Дайте описание режимов работы PCA.
4. Назовите особенности работы таймера модуля PCA.
5. Назовите особенности работы модулей захвата/сравнения модуля PCA.
6. Какие основные параметры определяют работу ШИМ?

ПРИЛОЖЕНИЕ А

(*обязательное*)

Пример листинга программы для микроконтроллера C8051F320

```
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------

#include <ioC8051f330.h>              // SFR declarations

//-----------------------------------------------------------------------------
// Global Constants
//-----------------------------------------------------------------------------
#define SYSCLK          24500000/8  // SYSCLK in Hz (24.5 MHz internal
                                    // oscillator / 8) the internal oscillator has a
                                    // tolerance of +/- 2%

#define TIMER_PRESCALER         48  // Based on Timer CKCON settings

// There are SYSCLK/TIMER_PRESCALER timer ticks per second
#define TIMER_TICKS_PER_SEC  SYSCLK/TIMER_PRESCALER

#define TIMER0_FREQ             100 // Timer interrupt per second (Hz)

// Note: TIMER_TICKS_PER_SEC/TIMER0_Freq should not exceed 65535
// (0xFFFF) for the 16-bit timer

#define AUX_T0_1    TIMER_TICKS_PER_SEC/TIMER0_FREQ
#define AUX_T0_2    -AUX_T0_1
#define AUX_T0_3    AUX_T0_2&0x00FF
#define AUX_T0_4    ((AUX_T0_2&0xFF00)>>8)

#define TIMER0_RELOAD_HIGH      AUX_T0_4  // Timer0 high byte
#define TIMER0_RELOAD_LOW       AUX_T0_3  // Timer0 low byte

#define TIMER1_FREQ             10 // Timer interrupt per second (Hz)

#define AUX_T1_1    TIMER_TICKS_PER_SEC/TIMER1_FREQ
#define AUX_T1_2    -AUX_T0_1
#define AUX_T1_3    AUX_T0_2&0x00FF
#define AUX_T1_4    ((AUX_T0_2&0xFF00)>>8)
```

```
#define TIMER1_RELOAD_HIGH      AUX_T1_4  // Timer0 high byte
#define TIMER1_RELOAD_LOW       AUX_T1_3  // Timer0 low byte
#define LED_TOGGLE_FREQ         6  // LED toggle frequency
// Number of T0 interrupt LED toggle frequency
#define LED_T0_CNT      TIMER0_FREQ/LED_TOGGLE_FREQ

//-----------------------------------------------------------------------
// Bit Definition
//-----------------------------------------------------------------------

#define pLED P1_bit.P13                 // LED='1' means ON
#define pButton P0_bit.P07              // pButton='0' means switch pressed

//-----------------------------------------------------------------------
// Global Variables Init
//-----------------------------------------------------------------------

volatile unsigned char T0_LED_Cnt = LED_T0_CNT;
volatile unsigned char Toggle1 = 0;

//-----------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------

void Port_Init (void);          // Port initialization routine
void Timer0_Init (void);        // Timer0 initialization routine

//-----------------------------------------------------------------------
// main() Routine
//-----------------------------------------------------------------------

void main (void)
{
  PCA0MD &= ~0x40;              // Clear watchdog timer enable

  Timer0_Init ();               // Initialize the Timer0
  Port_Init ();                 // Init Ports
  IE_bit.EA = 1;                // Enable global interrupts

  while (1);                    // Loop forever
}

//-----------------------------------------------------------------------
// Initialization Subroutines
```

```
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Port_Init
//-----------------------------------------------------------------------------
// Pinout:
//    P2.2 -> LED
//    all other port pins unused
//-----------------------------------------------------------------------------
void Port_Init (void)
{
  XBR1 = 0x40;                    // Enable crossbar
  P1MDOUT = 0x08;                 // LED as push-pull
}

//-----------------------------------------------------------------------------
// Timer0_Init
//-----------------------------------------------------------------------------

void Timer0_Init(void)
{
  TH0 = TIMER0_RELOAD_HIGH;       // Init Timer0 High register
  TL0 = TIMER0_RELOAD_LOW ;       // Init Timer0 Low register
  TH1 = TIMER1_RELOAD_HIGH;       // Init Timer1 High register
  TL1 = TIMER1_RELOAD_LOW ;       // Init Timer1 Low register
  TMOD = 0x11;                    // Timer 0, 1 in 16-bit mode
  CKCON = 0x02;                   // Timer0 uses a 1:48 prescaler
  IE_bit.ET0 = 1;                 // Timer0 interrupt enabled
  IE_bit.ET1 = 1;                 // Timer1 interrupt enabled
  TCON_bit.TR0 = 1;               // Timer0 ON
  TCON_bit.TR1 = 1;               // Timer1 ON
}

//-----------------------------------------------------------------------------
// Interrupt Service Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Timer0_ISR
//-----------------------------------------------------------------------------

#pragma vector=TF0_int //interrupt 2
__interrupt void Timer0_ISR (void)
{
```

```
  TH0 = TIMER0_RELOAD_HIGH;          // Reinit Timer0 High register
  TL0 = TIMER0_RELOAD_LOW;           // Reinit Timer0 Low register

  T0_LED_Cnt = T0_LED_Cnt - 1;
  if (T0_LED_Cnt == 0)
  {
    pLED = ~pLED;                          // Toggle the LED
    T0_LED_Cnt = LED_T0_CNT;
  }
}

//-------------------------------------------------------------------------
// Timer1_ISR
//-------------------------------------------------------------------------

#pragma vector=TF1_int //interrupt 2
__interrupt void Timer1_ISR (void)
{
  TH1 = TIMER1_RELOAD_HIGH;          // Reinit Timer0 High register
  TL1 = TIMER1_RELOAD_LOW;           // Reinit Timer0 Low register

  if (pButton == 0)
  {
    if (Toggle1 == 1)
    {

      TCON_bit.TR0 = ~TCON_bit.TR0;
      Toggle1 = 0;
      pLED = 0;

    }
  }
  if (pButton == 1)
  {
    Toggle1 = 1;
  }
}

//-------------------------------------------------------------------------
// End Of File
//-------------------------------------------------------------------------
```

ПРИЛОЖЕНИЕ Б

(*справочное*)

Описание регистров

Б.1 Описание регистров прерываний

IE: регистр разрешения прерываний:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 00000000 SFR Адрес: 0xA8
EA	ESPI0	ET2	ES0	ET1	EX1	ET0	EX0	
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	

Бит 7: **EA**: бит разрешения всех прерываний.

Этот бит глобально разрешает/запрещает все прерывания. Будучи сброшенным в 0, он перекрывает индивидуальные маски прерываний.

0: все источники прерываний запрещены.

1: каждое прерывание разрешено/запрещено в соответствии с его индивидуальной маской.

Бит 6: **ESPI0**: бит разрешения прерываний от модуля SPI0.

Этот бит устанавливает маскирование прерывания от модуля SPI0.

0: все прерывания от модуля SPI0 запрещены.

1: разрешены запросы прерываний, генерируемые при установке флага SPI0.

Бит 5: **ET2:** бит разрешения прерывания от Таймера 2.

Этот бит устанавливает маскирование прерывания от Таймера 2.

0: все прерывания от Таймера 2 запрещены.

1: разрешены запросы прерываний, генерируемые при установке флагов TF2L или TF2H.

Бит 4: **ES0**: бит разрешения прерываний от последовательного порта УАПП0.

0: прерывания от УАПП0 запрещены.

1: прерывания от УАПП0 разрешены.

Бит 3: **ET1**: бит разрешения прерывания от Таймера 1.

Этот бит устанавливает маскирование прерывания от Таймера 1.

0: все прерывания от Таймера 1 запрещены.

1: разрешены запросы прерываний, генерируемые при установке флага TF1.

Бит 2: **EX1:** бит разрешения внешнего прерывания 1.

Этот бит устанавливает маскирование внешнего прерывания 1.

0: внешнее прерывание 1 запрещено.

1: разрешены запросы прерываний, генерируемые сигналом на входе /INT1.

Бит 1: **ET0**: бит разрешения прерывания от Таймера 0.

Этот бит устанавливает маскирование прерывания от Таймера 0.

0: все прерывания от Таймера 0 запрещены.

1: разрешены запросы прерываний, генерируемые при установке флага TF0.

Бит 0: **EX0**: бит разрешения внешнего прерывания 0.

Этот бит устанавливает маскирование внешнего прерывания 0.

0: внешнее прерывание 0 запрещено.

1: разрешены запросы прерываний, генерируемые сигналом на входе /INT0.

IP: регистр приоритетов прерываний:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 10000000 SFR Адрес: 0xB8
-	PSPI0	PT2	PS0	PT1	PX1	PT0	PX0	
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	

Бит 7: не используется, читается как 1b, запись не оказывает никакого влияния.

Бит 6: **PSPI0**: управление приоритетом прерывания от модуля SPI0.

Этот бит устанавливает приоритет прерывания от модуля SPI0.

0: прерыванию от модуля SPI0 назначается низкий уровень приоритета.

1: прерыванию от модуля SPI0 назначается высокий уровень приоритета.

Бит 5: **PT2**: управление приоритетом прерывания от Таймера 2.

Этот бит устанавливает приоритет прерываний от Таймера 2.

0: прерыванию от Таймера 2 назначается низкий уровень приоритета.

1: прерываниям от Таймера 2 назначается высокий уровень приоритета.

Бит 4: **PS0**: управление приоритетом прерывания от последовательного порта УАПП0.

0: прерываниям от УАПП0 назначается низкий уровень приоритета.

1: прерываниям от УАПП0 назначается высокий уровень приоритета.

Бит 3: **PT1**: управление приоритетом прерывания от Таймера 1.

Этот бит устанавливает приоритет прерываний от Таймера 1.

0: прерываниям от Таймера 1 назначается низкий уровень приоритета.

1: прерываниям от Таймера 1 назначается высокий уровень приоритета.

Бит 2: **PX1:** управление приоритетом внешнего прерывания 1.

Этот бит устанавливает приоритет внешнего прерывания 1.

0: внешнему прерыванию 1 назначается низкий уровень приоритета.

1: внешнему прерыванию 1 назначается высокий уровень приоритета.

Бит 1: **PT0**: управление приоритетом прерывания от Таймера 0.

Этот бит устанавливает приоритет прерываний от Таймера 0.

0: прерываниям от Таймера 0 назначается низкий уровень приоритета.

1: прерываниям от Таймера 0 назначается высокий уровень приоритета.

Бит 0: **PX0**: управление приоритетом внешнего прерывания 0.

Этот бит устанавливает приоритет внешнего прерывания 0.

0: внешнему прерыванию 0 назначается низкий уровень приоритета.

1: внешнему прерыванию 0 назначается высокий уровень приоритета.

Б.2 Описание регистров портов ввода/вывода

XBR0: Регистр 0 матрицы портов ввода/вывода:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 00000000
CP1AE	CP1E	CP0AE	CP0E	SYSCKE	SMB0E	SPI0E	URT0E	
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	SFR Адрес: 0xE1

Бит 7: **CP1AE:** бит подключения асинхронного выхода Компаратора 1 (CP1).

0: асинхронный выход CP1 не соединен с выводом порта.

1: асинхронный выход CP1 соединен с выводом порта.

Бит 6: **CP1E**: бит подключения выхода Компаратора 1 (CP1).

0: CP1 не соединен с выводом порта.

1: CP1 соединен с выводом порта.

Бит 5: **CP0AE**: бит подключения асинхронного выхода Компаратора 0 (CP0).

0: асинхронный выход CP0 не соединен с выводом порта.

1: асинхронный выход CP0 соединен с выводом порта.

Бит 4: **CP0E**: бит подключения выхода Компаратора 0 (CP0).

0: CP0 не соединен с выводом порта.

1: CP0 соединен с выводом порта.

Бит 3: **SYSCKE**: бит подключения выхода /SYSCLK.

0: выход /SYSCLK не соединен с выводом порта.

1: выход /SYSCLK соединен с выводом порта.

Бит 2: **SMB0E**: бит подключения входов/выходов модуля SMBus0.

0: входы/выходы модуля SMBus0 не соединены с выводами порта.

1: входы/выходы модуля SMBus0 соединены с выводами порта.

Бит 1: **SPI0E**: бит подключения входов/выходов модуля SPI0.

0: входы/выходы модуля SPI0 не соединены с выводами порта.

1: входы/выходы модуля SPI0 соединены с выводами порта.

Бит 0: **URT0E**: бит подключения входов/выходов УАПП0.

0: входы/выходы УАПП0 не соединены с выводами порта.

1: TX0 и RX0 соединены с выводами P0.4 и P0.5 соответственно.

XBR1: Регистр 1 матрицы портов ввода/вывода:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 00000000
WEAKPUD	XBARE	T1E	T0E	ECIE	PCA0ME			
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	SFR Адрес: 0xE2

Бит 7: **WEAKPUD**: бит отключения слаботоковых подтяжек портов ввода/вывода.

0: слаботоковые подтяжки включены (кроме портов, чьи выводы настроены как аналоговые входы или 2-тактные цифровые выходы).

1: слаботоковые подтяжки отключены.

Бит 6: **XBARE**: бит включения матрицы.

0: матрица отключена, все драйверы портов отключены.

1: матрица включена.

Бит 5: **T1E**: бит подключения T1.

0: T1 не соединен с выводом порта.

1: T1 соединен с выводом порта.

Бит 4: **T0E**: бит подключения T0.

0: T0 не соединен с выводом порта.

1: T0 соединен с выводом порта.

Бит 3: **ECIE**: бит подключения внешнего входа счетчика ПМС.

0: ECI не соединен с выводом порта.

1: ECI соединен с выводом порта.

Биты 2-0: **PCA0ME**: биты подключения входов/выходов модуля ПМС.

000: все входы/выходы модуля ПМС не соединены с выводами порта.

001: CEX0 соединен с выводом порта.

010: CEX0, CEX1 соединены с двумя выводами порта.

011: CEX0, CEX1, CEX2 соединены с тремя выводами порта.

100: CEX0, CEX1, CEX2, CEX3 соединены с четырьмя выводами порта.

101: CEX0, CEX1, CEX2, CEX3, CEX4 соединены с пятью выводами порта.

P0MDIN: регистр настройки входов Порта 0:

Биты 7–0: **P0MDIN**.[7:0]: биты выбора режима входов Порта 0.

Если вывод настроен как аналоговый вход, то его слаботоковая подтяжка, цифровой драйвер и цифровой приемник отключаются.

0: вывод P0.n настроен как аналоговый вход.

1: вывод P0.n не настроен как аналоговый вход.

P0MDOUT: регистр настройки выходов Порта 0:

Биты 7–0: **P0MDOUT**.[7:0]: биты настройки выходного драйвера порта 0: игнорируются, если соответствующий бит в регистре P0MDIN сброшен в 0.

0: соответствующий вывод P0.n настроен как выход с открытым стоком.

1: соответствующий вывод P0.n настроен как цифровой 2-тактный выход.

Примечание – Если сигналы SDA и SCL появляются на любом выводе порта, то каждый из этих выводов будет настроен как выход с открытым стоком, независимо от значения регистра P0MDOUT.

P0SKIP: регистр выбора выводов Порта 0, пропускаемых матрицей:

Биты 7–0: P0SKIP.[7:0]: биты выбора выводов порта 0, пропускаемых матрицей при назначении выводов. Эти биты выбирают выводы порта, пропускаемые декодером матрицы. Выводы порта, используемые как аналоговые входы (для АЦП или компаратора) или используемые для специальных целей (вход

VREF, схема внешнего генератора, вход CNVSTR) должны пропускаться матрицей.

0: соответствующий вывод P0.n не пропускается матрицей при назначении выводов.

1: соответствующий вывод P0.n пропускается матрицей при назначении выводов.

Б.3 Описание регистров управления таймерами

TCON: Регистр управления Таймерами 0 и 1:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 00000000 SFR Адрес: 0x88
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	

Бит 7: **TF1**: флаг переполнения Таймера 1.

Устанавливается аппаратно при переполнении Таймера 1. Сбрасывается аппаратно при переходе к процедуре обслуживания прерывания от Таймера 1, но может быть сброшен и программно.

0: переполнение Таймера 1 не обнаружено.

1: Таймер 1 переполнился.

Бит 6: **TR1**: управление запуском Таймера 1.

0: Таймер 1 отключен.

1: Таймер 1 включен.

Бит 5: **TF0**: флаг переполнения Таймера 0.

Устанавливается аппаратно при переполнении Таймера 0. Сбрасывается аппаратно при переходе к процедуре обслуживания прерывания от Таймера 0, но может быть сброшен и программно.

0: переполнение Таймера 0 не обнаружено.

1: Таймер 0 переполнился.

Бит 4: **TR0**: управление запуском Таймера 0.

0: Таймер 0 отключен.

1: Таймер 0 включен.

Бит 3: **IE1**: внешнее прерывание 1.

Этот флаг аппаратно устанавливается в 1 при обнаружении активного фронта/уровня (определяется битом IT1) внешнего сигнала. Может быть сброшен программно, но при переходе к процедуре обслуживания внешнего прерывания 1 сбрасывается аппаратно, если IT1=1. При IT1=0 этот флаг устанавливается в 1, если на внешнем выводе /INT1 присутствует сигнал с активным логическим уровнем, который определяется битом IN1PL в регистре INT01CF.

Бит 2: **IT1**: выбор типа внешнего прерывания 1.

Этот бит определяет, какое событие будет вызывать внешнее прерывание 1: фронт или активный уровень внешнего сигнала /INT1 (активный уровень внешнего сигнала /INT1 определяется битом IN1PL в регистре INT01CF).

0: внешнее прерывание 1 вызывается активным уровнем сигнала /INT1.

1: внешнее прерывание 1 вызывается фронтом сигнала /INT1.

Бит 1: **IE0**: внешнее прерывание 0.

Этот флаг аппаратно устанавливается в 1 при обнаружении активного фронта/уровня (определяется битом IT0) внешнего сигнала. Может быть сброшен программно, но при переходе к процедуре обслуживания внешнего прерывания 0 сбрасывается аппаратно, если IT0=1. При IT0=0 этот флаг устанавливается в 1, если на внешнем выводе /INT0 присутствует сигнал с активным логическим уровнем, который определяется битом IN0PL в регистре INT01CF.

Бит 0: **IT0**: выбор типа внешнего прерывания 0.

Этот бит определяет, какое событие будет вызывать внешнее прерывание 0: фронт или активный уровень внешнего сигнала /INT0 (активный уровень внешнего сигнала /INT0 определяется битом IN0PL в регистре INT01CF).

0: внешнее прерывание 0 вызывается активным уровнем сигнала /INT0.

1: внешнее прерывание 0 вызывается фронтом сигнала /INT0.

TMOD: Регистр режима Таймеров 0 и 1:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 00000000
GATE1	C/T1	T1M1	T1M0	GATE0	C/T0	T0M1	T0M0	
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	SFR Адрес: 0x89

Бит 7: **GATE1**: управление блокировкой Таймера 1.

0: Таймер 1 включен, если TR1 = 1, независимо от логического уровня на входе /INT1.

1: Таймер 1 включен только тогда, когда TR1 = 1 и на входе /INT1 активный логический уровень; определяется битом IN1PL в регистре INT01CF.

Бит 6: **C/T1**: выбор режима таймера или счетчика для TC1.

0: TC1 работает как таймер: Таймер 1 инкрементируется от внутреннего сигнала тактирования, который задается битом T1M (CKCON.4).

1: TC1 работает как счетчик: Таймер 1 инкрементируется под воздействием перехода из 1 в 0 внешнего входного сигнала (T1).

Биты 5 и 4: **T1M1 и T1M0**: выбор режима работы Таймера 1:

00 Режим 0: 13-разрядный таймер-счетчик;

01 Режим 1: 16-разрядный таймер-счетчик;

10 Режим 2: 8-разрядный таймер-счетчик с автоперезагрузкой;

11 Режим 3: Таймер 1 неактивен.

Бит 3: **GATE0**: управление блокировкой Таймера 0.

0: Таймер 0 включен, если TR0 = 1, независимо от логического уровня на входе /INT0.

1: Таймер 0 включен только тогда, когда TR0 = 1 и на входе /INT0 активный логический уровень; определяется битом IN0PL в регистре INT01CF.

Бит 2: **C/T0**: выбор режима таймера или счетчика для TC0.

0: TC0 работает как таймер: Таймер 0 инкрементируется от внутреннего сигнала тактирования, который задается битом T0M (CKCON.3).

1: TC0 работает как счетчик: Таймер 0 инкрементируется под воздействием перехода из 1 в 0 внешнего входного сигнала (T0).

Биты 1 и 0: **T0M1 и T0M0**: выбор режима работы Таймера 0.

Эти биты определяют режим работы Таймера 0:

00 Режим 0: 13-разрядный таймер-счетчик;

01 Режим 1: 16-разрядный таймер-счетчик;

10 Режим 2: 8-разрядный таймер-счетчик с автоперезагрузкой;

11 Режим 3: два 8-разрядных таймера-счетчика.

Б.4 Описание регистров управления АЦП

ADC0CF: Регистр конфигурации АЦП0:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 11111000
AD0SC4	AD0SC3	AD0SC2	AD0SC1	AD0SC0	AD0LJST	-	-	
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	SFR Адрес: 0xBC

Биты 7–3: **AD0SC4–0**: биты установки периода сигнала дискретизации АЦП0.

Частота сигнала дискретизации АЦП0 определяется частотой системного тактового сигнала в соответствии со следующим уравнением: AD0SC = = (SYSCLK/CLKSAR0) – 1,

где AD0SC – 5-разрядное значение, задаваемое битами AD0SC4-0;

CLKSAR0 – необходимая частота сигнала дискретизации АЦП0.

Бит 2: **AD0LJST**: бит выравнивания результата преобразования.

0: данные в регистровой паре ADC0H:ADC0L выровнены вправо.

1: данные в регистровой паре ADC0H:ADC0L выровнены влево.

Биты 1 и 0: не используются, читаются как 00b, запись не оказывает никакого влияния.

ADC0CN: Регистр управления АЦП0:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 00000000
AD0EN	AD0TM	AD0INT	AD0BUSY	AD0WINT	AD0CM2	AD0CM1	AD0CM0	
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0 (доступен в битовом режиме адресации)	SFR Адрес: 0xE8

Бит 7: **AD0EN**: бит включения АЦП0.

0: АЦП0 отключен, находится в режиме пониженного энергопотребления.

1: АЦП0 включен, находится в активном режиме и готов к преобразованию данных.

Бит 6: **AD0TM**: бит установки режима слежения (выборки-хранения) АЦП0.

0: нормальный режим: когда АЦП0 включен, слежение осуществляется всегда, за исключением момента преобразования.

1: энергосберегающий режим; режим слежения определяется битами AD0CM2-0.

Бит 5: **AD0INT**: флаг прерывания от АЦП0 (устанавливается при завершении преобразования).

0: АЦП0 не закончил преобразование данных.

1: АЦП0 закончил преобразование данных.

Бит 4: **AD0BUSY**: бит занятости АЦП0.

Чтение:

0: преобразование данных завершено или в данный момент преобразование не осуществляется.

При аппаратном обнулении этого бита флаг AD0INT устанавливается в 1.

1: идет процесс преобразования данных.

Запись:

0: не оказывает никакого влияния.

1: Инициирует запуск преобразования АЦП0, если биты AD0CM2-0 = 000b.

Бит 3: **AD0WINT**: флаг прерывания от детектора диапазона АЦП0.

0: преобразованные данные не соответствуют заданному диапазону (с момента последнего обнуления этого флага).

1: преобразованные данные соответствуют заданному диапазону.

Биты 2–0: **AD0CM1-0**: биты выбора режима запуска преобразования АЦП0.

Если AD0TM = 0:

000: запуск преобразования осуществляется установкой в 1 бита AD0BUSY;

001: запуск преобразования осуществляется при переполнении Таймера 0;

010: запуск преобразования осуществляется при переполнении Таймера 2;

011: запуск преобразования осуществляется при переполнении Таймера 1;

100: запуск преобразования осуществляется нарастающим фронтом внешнего сигнала CNVSTR;

101: запуск преобразования осуществляется при переполнении Таймера 3;

11x: зарезервировано.

Если AD0TM = 1:

000: слежение (выборка) начинается в момент установки в 1 бита AD0BUSY и длится 3 периода сигнала дискретизации АЦП0, затем начинается преобразование данных;

001: слежение (выборка) начинается при переполнении Таймера 0 и длится 3 периода сигнала дискретизации АЦП0, затем начинается преобразование данных;

010: слежение (выборка) начинается при переполнении Таймера 2 и длится 3 периода сигнала дискретизации АЦП0, затем начинается преобразование данных;

011: слежение (выборка) начинается при переполнении Таймера 1 и длится 3 периода сигнала дискретизации АЦП0, затем начинается преобразование данных;

100: слежение (выборка) происходит лишь при низком уровне сигнала на входе CNVSTR; преобразование запускается нарастающим фронтом сигнала на входе CNVSTR;

101: слежение (выборка) начинается при переполнении Таймера 3 и длится 3 периода сигнала дискретизации АЦП0; затем начинается преобразование данных;

11x: зарезервировано.

IDA0CN: Регистр управления АЦП0:

SFR-страница: **все страницы**
SFR-адрес: **0xB9**

R/W	R/W	R/W	R/W	R	R	R/W	R/W	Значение при сбросе: 01110010
IDA0EN	IDA0CM			–	–	IDA0OMD		
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	

Бит 7:

IDA0EN: бит включения ЦАП0.

0: ЦАП0 выключен.

1: ЦАП0 включен.

Биты 6–4: IDA0CM[2–0]: Биты выбора режима обновления выходного сигнала ЦАП0:

000: обновление выходного сигнала ЦАП при переполнении Таймера 0;

001: обновление выходного сигнала ЦАП при переполнении Таймера 1;

010: обновление выходного сигнала ЦАП при переполнении Таймера 2;

011: обновление выходного сигнала ЦАП при переполнении Таймера 3;

100: обновление выходного сигнала ЦАП по переднему фронту сигнала CNVSTR;

101: обновление выходного сигнала ЦАП по заднему фронту сигнала CNVSTR;

110: обновление выходного сигнала ЦАП по обоим фронтам сигнала CNVSTR;

111: обновление сигнала ЦАП при записи в регистр IDA0H (по умолчанию);

Биты 3–2: не используются, читаются как 00Ь, запись не оказывает никакого влияния.

Биты 1–0: IDAOOMD[1:0]: биты выбора максимального выходного тока ЦАП0:

00: выходной ток полной шкалы = 0,5 мА;

01: выходной ток полной шкалы = 1 мА;

1x: выходной ток полной шкалы = 2 мА (по умолчанию).

Б.5 Описание регистров специального назначения (Special Function Registers – SFR)

PSW: Слово состояния программы:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Значение при сбросе: 00000000 SFR Адрес: 0xD0
CY	AC	F0	RS1	RS0	OV	F1	PARITY	
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0 (доступен в битовом режиме адресации)	

Бит 7: **CY**: флаг переноса.

Этот бит устанавливается, если в результате последней арифметической операции произошел перенос (сложение) или заем (вычитание). Он сбрасывается в 0 всеми другими арифметическими операциями.

Бит 6: **AC**: флаг десятичного переноса.

Этот бит устанавливается, если в результате последней арифметической операции произошел перенос (сложение) в старший полубайт или заем (вычитание) из старшего полубайта. Он сбрасывается в 0 всеми другими арифметическими операциями.

Бит 5: **F0:** флаг пользователя 0.

Это доступный в битовом режиме адресации флаг общего назначения, предназначенный для использования под управлением программы.

Биты 4 и 3**: RS1 и RS0**: биты выбора банка регистров.

Эти биты определяют активный банк регистров.

Бит 2: **OV**: флаг переполнения.

Этот бит устанавливается в 1 в следующих случаях:

- если в результате выполнения команды ADD, ADDC или SUBB произошло переполнение с изменением знака;

- если в результате выполнения команды MUL произошло переполнение (результат превышает значение 255);

- если при выполнении команды DIV произошло деление на 0.

Бит OV сбрасывается в 0 командами ADD, ADDC, SUBB, MUL и DIV во всех других случаях.

Бит 1: **F1:** флаг пользователя 1.

Это доступный в битовом режиме адресации флаг общего назначения, предназначенный для использования под управлением программы.

Бит 0: **PARITY:** флаг четности.

Этот бит устанавливается в 1, если сумма 8 бит в аккумуляторе нечетная и сбрасывается, если сумма четная.

Б.6 Описание регистров управления ПМС

PCA0CN: Регистр управления ПМС:

R/W	R/W	R/W	R/W	R/W	R	R/W	R/W	Значение
CF	CR	-	CCF4	CCF3	CCF2	CCF1	CCF0	при сбросе: 00000000
Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит0	SFR Адрес: 0xD8

Бит 7: CF: флаг переполнения таймера-счетчика ПМС.

Устанавливается в 1 аппаратно, когда таймер-счетчик ПМС переполняется из состояния OxFFFF в состояние 0x0000. Если прерывание от таймера-счетчика ПМС (от флага CF) разрешено, то установка этого бита приведет к переходу на процедуру обслуживания прерывания от флага CF. Этот бит не сбрасывается аппаратно и должен быть сброшен программно.

Бит 6: CR: управление запуском таймера-счетчика ПМС.

Этот бит включает/отключает таймер-счетчик ПМС.

0: таймер-счетчик ПМС отключен.

1: таймер-счетчик ПМС включен.

Бит 5: не используется, читается как 0Ь, запись не оказывает никакого влияния.

Биты 4–0: CCF4–0: флаг захвата/сравнения модуля 4–0 ПМС.

Этот бит устанавливается в 1 аппаратно, если происходит захват или совпадение сравниваемых значений. Если прерывание от флага CCF разрешено, то установка этого бита приведет к переходу на процедуру обслуживания прерывания от флага CCF. Этот бит не сбрасывается аппаратно и должен быть сброшен программно.

Св. план 2016, поз. 12

Учебное издание

Давыдов Максим Викторович
Кракасевич Сергей Викторович
Давыдова Надежда Сергеевна
Меженная Марина Михайловна

ПРОГРАММНО-УПРАВЛЯЕМЫЕ МИКРОКОНТРОЛЛЕРНЫЕ УСТРОЙСТВА. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

ПОСОБИЕ

Редактор *М. А. Зайцева*
Корректор *Е. И. Герман*
Компьютерная правка, оригинал-макет *В. М. Задоля*

Подписано в печать **.**.2017. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. . Уч.-изд. л. 4,0. Тираж 90 экз. Заказ 110.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6