ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
Северо-Кавказский филиал
ордена Трудового Красного Знамени федерального государственного
бюджетного образовательного учреждения высшего образования
"Московский технический университет связи и информатики"

Методические указания
к лабораторным и практическим занятиям

МИКРОКОНТРОЛЛЕРЫ

Направление подготовки:
09.03.01 Информатика и вычислительная техника
11.03.02 Инфокоммуникационные технологии и системы связи

Ростов-на-Дону
2019

УДК 681.3.06 (076)
ББК 32.07

Чикалов А.Н. Микроконтроллеры. Методические указания к лабораторным и практическим занятиям. Ростов-на-Дону: Северо-Кавказский филиал МТУСИ, 2019.- 77 с.

В пособии изложены методические рекомендации, содержательные материалы и контрольные задания для проведения лабораторных и практических занятий по изучению архитектуры микроконтроллеров, приемов и методов программирования портов, счетчиков, системы прерываний микроконтроллеров семейства AVR. В качестве инструмента использована среда разработки AVR Studio и среда моделирования электронных схем Proteus. Пособие содержит необходимые справочные материалы.

Методические указания предназначены для студентов, обучающихся по направлениям подготовки 09.03.01 Информатика и вычислительная техника и 11.03.02 Инфокоммуникационные технологии и системы связи, профилей Многоканальные телекоммуникационные системы, Сети связи и системы коммутации, Защищенные системы и сети связи, Системы радиосвязи и радиодоступа, Вычислительные машины, комплексы, системы и сети, Программное обеспечение и интеллектуальные системы.

Пособие предназначено для использования при изучении дисциплин Специализированные процессоры, Микропроцессорные системы, Вычислительная техника и информационные технологии, а также может быть использовано преподавателями и студентами при изучении родственных дисциплин и в процессе самостоятельной работы.

Учебное пособие обсуждено и одобрено на заседании кафедры ИВТ.
Протокол №1 от 26.08.2019 г.

Рецензент Зав. кафедрой ИВТ д.т.н. профессор Соколов С.В.

СОДЕРЖАНИЕ

1. ИЗУЧЕНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ СПЕЦКОНТРОЛЛЕРА

Цель

1. Углубить и закрепить теоретические знания по принципам построения специализированных контроллеров;

2. Приобрести практические навыки самостоятельного изучения и анализа схем контроллеров;

3. Совершенствовать навыки анализа, обобщения и систематизации полученных результатов, навыки составления и оформления отчетных материалов, навыки точного и лаконичного представления докладов на вопросы технического характера.

Учебные вопросы

1.1. Назначение, основные технические характеристики микроконтроллера AVR;

1.2. Структурная схема и назначение периферийных устройств;

1.3. Программная модель микроконтроллера.

Литература для подготовки к занятию

1. Голубцов М.С. Микроконтроллеры AVR: от простого к сложному. М.: СОЛОН-Пресс, 2003. 288 с.
2. Редькин П.П. Микроконтроллеры Atmel архитектуры AVR32 семейства AT32UC3. Руководство пользователя. М.: Техносфера, 2010. - 784 с.
3. Белов А.В. Самоучитель разработчика устройств на микроконтроллерах AVR. - СПб.: Наука и техника, 2008. -544 с.
4. Ревич Ю.В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера. СПб.: БХВ-Петербург, 2011. -352с.
5. Евстифеев А.В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя. - М.: Издательский дом Додэка-XXI, 2007 . - 432с.

Содержание отчета

1. Название работы.

2. Название каждого учебного вопроса и краткий конспект в объеме практических заданий по вопросу.

Вопросы для подготовки к занятию

1. Алгоритм работы центрального процессора.
2. Принципы работы оперативной памяти и Flash-памяти.
3. Характеристика гарвардской и фон-Неймановской архитектур.
4. В чем отличие CISC и RISC структур системы команд.

Актуальность занятия

Спецконтроллер являются основными устройствами, осуществляющим управление и обработку данных в системах управления. Поэтому очень важно знать его характеристики, технические возможности взаимодействия с периферией и программную модель.

Задание 1.1. Назначение, основные технические характеристики микроконтроллера AVR

Микропроцессоры (МП) и микроконтроллеры (МК) для встраиваемых приложений *[embedded applications]* ориентированы на построение законченных систем управления и их узлов. В частности 8-разрядные МК в общем объеме процессорных микросхем составляет чуть меньше 30%. AVR, пожалуй, одно из самых интересных направлений, развиваемых корпорацией Atmel. Они представляют собой мощный инструмент для создания современных высокопроизводительных и экономичных многоцелевых контроллеров. На настоящий момент соотношение "цена - производительность - энергопотребление" для AVR является одним из лучших на мировом рынке 8-разрядных микроконтроллеров. Объемы продаж AVR в мире удваиваются ежегодно. В геометрической прогрессии растет число сторонних фирм, разрабатывающих и выпускающих разнообразные программные и аппаратные средства поддержки разработок для них. Можно считать, что AVR постепенно становится еще одним индустриальным стандартом среди 8-разрядных микроконтроллеров общего назначения.

МК отличают от других процессорных устройств такие существенные черты:

- развитые встроенные аппаратные узлы ввода/вывода, разгружающие процессор от рутинных действий и поддерживающие интерфейсы различных типов - дискретный, параллельный, последовательный, аналоговый;
- интегрированная на кристалл память программ и данных, в том числе, энергонезависимая;
- наличие большого числа разновидностей кристаллов, различающихся числом выводов и типом корпуса, объемами памяти, составом блоков ввода/вывода и пр.;
- встроенные узлы загрузки кодов программ и данных и узлы внутрисхемной отладки, работающие по последовательному интерфейсу;
- гибкая система тактирования и управления энергопотреблением, что актуально для мобильных устройств.

Основные различия между МК данной группы сводятся к производительности процессора, определяемой сочетанием следующих факторов: состав системы команд и режимов адресации, разрядность процессора (8/16/32), частота тактирования (десятки - сотни мегагерц). Рост разрядности сочетается с ростом частоты и снижением напряжения питания (с 5 В до 3.3 В, а ядра - до 1.2 В).

На отечественном рынке 8-разрядные МК представлены прежде всего «клонами» и «потомками» модели *MCS*-8051, семействами *PIC12*/14/16/18 (*MicroChip*), семейством *AVR* (фирма *Atmel*) и пр. Эти МК ориентированы на решение задач управления с преобладанием логических и несложных арифметических законов управления. Их максимальные тактовые частоты доходят сегодня до 16…25 МГц при питании 5 В. Уровень входных и выходных сигналов при этом имеет достаточный запас помехоустойчивости – примерно одна треть от напряжения питания, то есть не менее 1.7 В.

В частности, в данном курсе наибольшее внимание будет уделено семейству *AVR* фирмы *Atmel*, как одному из динамично развивающихся и доступному на отечественном рынке.

Семейством (или серией) МК называется ряд микросхем однородной разработки, имеющих идентичную архитектуру, но различающихся количеством ячеек памяти, выводов корпуса, составом и количественными характеристиками встроенных периферийных модулей. Различают в зависимости от аппаратного обеспечения семейства "AVR-tiny", "AVR-classic" и "AVR-mega" Выпуск МК для встраиваемых применений в виде семейства позволяет производителям снизить затраты на создание богатой номенклатуры микросхем (при разработке новых кристаллов используются готовые проверенные в производстве модули кристалла), а пользователям МК сокращает затраты времени на освоение моделей, наиболее подходящих для решения новых задач.

История и особенности семейства AVR-8

В середине 90-х годов (20 века) фирма *Atmel* выпустила клон МК *MCS-8051* по технологии К-МОП – семейство *AT89xx*, в котором реализовала два существенных на тот момент новшества:

- **интегрированная на кристалле память программ** типа *FlashROM* и энергонезависимая память данных типа *EEPROM*,

- **встроенный механизм загрузки/выгрузки** содержимого памяти программ (*FlashROM*) и данных (*EEPROM*) с использованием последовательного интерфейса (типа *SPI*).

Сочетание этих новаций обеспечило существенное удешевление и упрощение - отпала необходимость в использовании дорого керамического корпуса с кварцевым окном для ультрафиолетового стирания, и отпала необходимость использования не дешевого (не менее 300$) параллельного программатора, в который при каждой операции записи приходилось помещать корпус с микросхемой памяти программ. Новая технология стала называться внутрисистемным программированием *[In-System Programming - ISP]*, существенно упростила и удешевила процедуру редактирования содержимого памяти программ (ПП) и данных (ПД) - теперь стало возможным выполнять перепрограммирование МК прямо "в системе", то есть на целевой плате.

Следующим шагом фирмы *Atmel* в конце 90-х годов стала разработка нового семейства 8-разрядных МК *AVR*-8 (*AT90xxxx*), в котором кроме новшеств, внедренных при разработке *AT89xx*, добавились следующие:

- **интегрированные на кристалл узлы ввода аналоговых сигналов** - аналоговый компаратор и аналогово-цифровой преобразователь;

- **новая система команд *[RISC]*** сразу разрабатывалась с расчетом на использование языка *C/C++*,

- процессор поддерживает **4-шаговый конвейер,** что обеспечивает выполнение большинства команд (кроме команд перехода) за один машинный цикл,

- отказ от аккумулятора и использование достаточно **большого файла регистров общего назначения** (32*8 бит) снижает необходимость частых обращений к ПД (ОЗУ) с более долгими или длинными режимами адресации; в этом же файле разместились 16-разрядные указатели на ПД и ПП;

- отказ от использования внешней ПП при наличии функции внутрисистемного программирования позволил использовать корпуса с малым числом ног (менее 40), что еще более снижает стоимость микросхемы. Дополнительным преимуществом отказа от внешней ПП стала повышенная степень защиты содержимого ПП от несанкционированного копирования.

Последовательность разработки подсемейств *AVR*-8: *AT90*, *ATmega*, *[FPSLIC]*, *ATtiny*, *ATxmega.*

В отчете представить:

1. Основные технические характеристики МКК ATTiny 2313;
2. Отличие МКК от микропроцессора.

Задание 1.2. Структурная схема и назначение периферийных устройств

Архитектурой называют образ, видимый пользователю, это концептуальное представление для программиста или схемотехника, работающего с данным МК. Основными элементами архитектуры являются:

• система команд и способы адресации;

• организация памяти и регистров;

• формат разрядной сетки

• организация встроенных периферийных устройств, механизм обращения к ним и системы прерываний;

• типовые схемы включения с описанием назначения и расположения выводов и особенностей внутренней схемотехники выводов.

Знание архитектуры является минимально достаточным для понимания и практической работы с микроконтроллером. Сравнение однотипных черт и параметров архитектуры разных МК позволяет грамотно выбирать МК для решения конкретной задачи.

Структура МК отражает состав и связи его внутренних блоков в графическом виде. Знание структуры МК не является обязательным для работы с МК, но облегчает понимание и сравнение МК между собой за счет дополнительного визуального восприятия.

Архитектура AVR

Система команд 8-разрядных МК серии AVR содержит от 90 до 133 инструкций, число которых зависит от модели. Как и для большинства других МК, этот набор можно условно разделить на три основные группы:

- арифметико-логические, включая сдвиги (собственно вычисления),
- передачи данных между ячейками памяти МК,
- ветвлений (переходов) в программе (формируют структуру программы).

По формату обрабатываемых данных различают команды, оперирующие с байтами (8-битное слово) и с битами, отдельные команды работают с двухбайтными числами. Нередко группу команд работы с битами выделяют в самостоятельную четвертую группу.

Формат команды описывает структуру машинного слова команды. Часть бит выделяют под *код операции,* это обязательная часть любой команды. Остальные биты указывают *операнды*, то есть данное, адрес ячейки данных или адрес перехода. Различают команды безоперандные, одно- и двухоперандные. В двухоперандных командах обработки данных один из операндов называется источником *[source]*, его содержимое не изменяется, другой – приемником *[destination]*, его содержимое содержит результат операции.

Способы адресации определяют способы кодирования операндов, для кодирования используется двоичный код. Различают следующие виды адресации:

- *прямая* – адрес в виде константы, число бит n определяет число адресуемых ячеек $N = 2n$,
- *непосредственная* (или абсолютная) – данное в виде константы,
- несколько разновидностей *косвенной* – указывается адрес регистра-указателя (*X, Y, Z, PC, SP*), содержимое которого содержит адрес ячейки данных или перехода:
- простая косвенная,
- с преддекрементом – значение регистра указателя увеличивается до использования адреса,
- с постинкрементом – значение регистра указателя уменьшается после использования адреса,
- индексная – значение адреса вычисляется как сумма содержимого регистра указателя и константы,
- относительная – разновидность индексной с использованием счетчика команд как регистра указателя перехода.

Структура AVR

Микроконтроллер AVR содержит: быстрый RISK-процессор (Reduced Instruction Set Computer), два типа энергонезависимой памяти (Flash-память программ и память данных EEPROM), оперативную память RAM, порты ввода/вывода и различные периферийные интерфейсные схемы (рис.1.1).

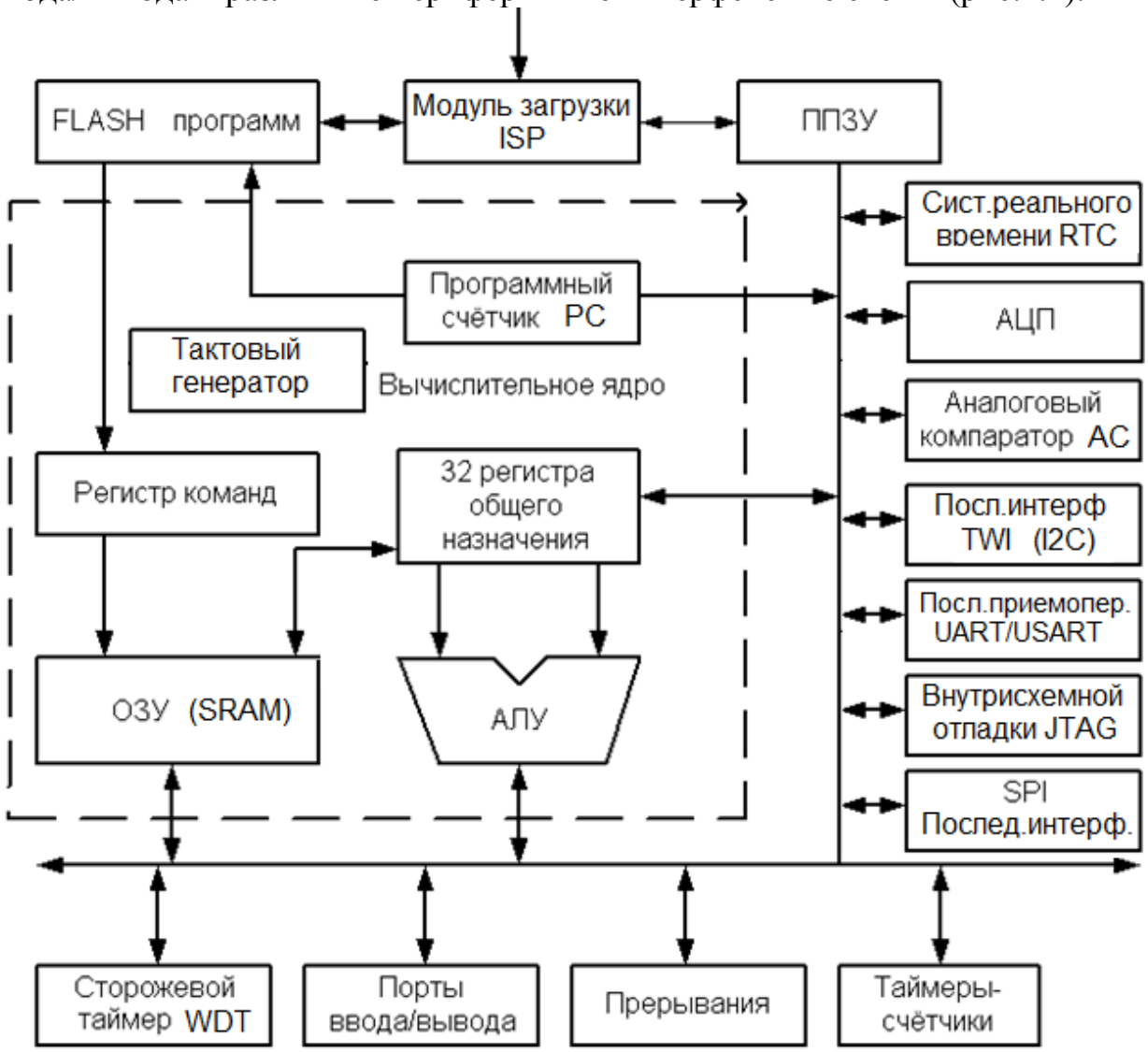

Рис.1.1. Структурная схема МКК AVR

Вычислительное ядро

Сердцем микроконтроллеров AVR является 8-битное вычислительное ядро или центральное процессорное устройство (ЦПУ), построенное на принципах RISK-архитектуры. Основой этого блока служит арифметико-логическое устройство (АЛУ). По системному тактовому сигналу из памяти программ в соответствии с содержимым счетчика команд PC (Program Counter) выбирается очередная команда и выполняется в АЛУ. Во время выбора команды из памяти программ происходит выполнение предыдущей выбранной команды, что и позволяет достичь быстродействия 1 MIPS на 1 МГц.

Память программ (Flash-память)

Память программ предназначена для хранения последовательности команд, управляющих функционированием микроконтроллера, и имеет 16-ти битную организацию. Все AVR имеют Flash-память программ, которая может быть различного размера – от 1 до 256 КБайт. Ее главное достоинство в том, что она построена на принципе электрической перепрограммируемости, т. е. допускает многократное стирание и запись информации. Программа заносится во Flash-память AVR как с помощью обычного программатора, так и с помощью SPI-интерфейса (последовательный периферийный интерфейс - Serial Peripheral Interface), в том числе непосредственно на собранной плате. Возможностью внутрисхемного программирования (функция ISP) через коммуникационный интерфейс SPI обладают все микроконтроллеры AVR, кроме Tiny11 и Tiny28.

Все микроконтроллеры семейства Mega имеют возможность *самопрограммирования,* т. е. самостоятельного изменения содержимого своей памяти программ. Эта особенность позволяет создавать на их основе очень гибкие системы, алгоритм работы которых будет меняться самим микроконтроллером в зависимости от каких-либо внутренних условий или внешних событий. Гарантированное число циклов перезаписи Flash-памяти у микроконтроллеров AVR второго поколения составляет не менее 10 тыс. циклов при типовом значении 100 тыс. циклов. (В официальной технической документации Atmel Corp. указывается значение 10 тыс. циклов.)

Память данных

Память данных разделена на три части:

- регистровая память,
- оперативная память (ОЗУ – оперативное запоминающее устройство или RAM)
- энергонезависимая память (ЭСППЗУ или EEPROM).

Регистровая память (РОН и РВВ)

*Регистров общего назначения (***РОН** - General Purpose Registers – GPR) всего 32, они имеют байтовый формат и объединены в файл. РОН находятся в начале адресного пространства оперативной памяти, но физически не являются ее частью. Поэтому к ним можно обращаться двумя способами (как к регистрам и как к памяти). Такое решение является особенностью AVR и повышает эффективность работы и производительность микроконтроллера.

*Служебные регистры ввода/вывода (***РВВ***)* объединяют регистры управления микроконтроллером, регистры состояния и т. п., а также регистры управления периферийными устройствами, входящими в состав микроконтроллера. По сути, управление микроконтроллером заключается в управлении этими регистрами.

И РОН и РВВ расположены в адресном пространстве ОЗУ, но не являются его частью. Отличие между регистрами и оперативной памятью состоит в том, что с регистрами можно производить любые операции (арифметические, логические, битовые), а в оперативную память можно лишь записывать данные из регистров.

Энергонезависимая память данных (EEPROM)

Для долговременного хранения различной информации, которая может изменяться в процессе функционирования микроконтроллерной системы, используется EEPROM-память - перепрограммируемое ПЗУ (ППЗУ). Все AVR имеют блок энергонезависимой электрически перезаписываемой памяти данных EEPROM от 64 Байт до 4 КБайт. Этот тип памяти, доступный программе микроконтроллера непосредственно в ходе ее выполнения, удобен для хранения промежуточных данных, различных констант, коэффициентов, серийных номеров, ключей и т.п. EEPROM может быть загружена извне как через SPI интерфейс, так и с помощью обычного программатора. Число циклов стирание/запись – не менее 100 тыс.

Два программируемых бита секретности позволяют защитить память программ и энергонезависимую память данных EEPROM от несанкционированного считывания

Оперативная память (ОЗУ или RAM)

Внутренняя оперативная статическая память Static RAM (SRAM) имеет байтовый формат и используется для оперативного хранения данных. Размер оперативной памяти может варьироваться у различных чипов от 64 Байт до 4 КБайт. Число циклов чтения и записи в RAM не ограничено, но при отключении питающего напряжения вся информация теряется. Для некоторых микроконтроллеров возможна организация подключения внешнего статического ОЗУ объемом до 64К.

В микроконтроллерах AVR реализована Гарвардская архитектура, в соответствии с которой разделены не только адресные пространства памяти программ и памяти данных, но и шины доступа к ним.

Периферия микроконтроллеров AVR включает: порты ввода и вывода, поддержку внешних прерываний, таймеры-счетчики, сторожевой таймер, аналоговые компараторы, 10-разрядный 8-канальный АЦП, интерфейсы UART, JTAG и SPI, устройство сброса по понижению питания, широтно-импульсные модуляторы.

Порты ввода/вывода (I/O)

Порты ввода/вывода AVR имеют число независимых линий “вход/выход” от 3 до 53. Каждая линия порта может быть запрограммирована на вход или на выход. Мощные выходные драйверы обеспечивают токовую нагрузочную способность 20 мА на линию порта (втекающий ток) при мак-

симальном значении 40 мА, что позволяет, например, непосредственно подключать к микроконтроллеру светодиоды и биполярные транзисторы. Общая токовая нагрузка на все линии одного порта не должна превышать 80 мА (все значения приведены для напряжения питания 5 В).

Архитектурная особенность построения портов ввода/вывода у AVR заключается в том, что для каждого физического вывода (пина) существует 3 бита контроля/управления, а не 2, как у распространенных 8-разрядных микроконтроллеров (Intel, Microchip, Motorola и т.д.). Это позволяет избежать необходимости иметь копию содержимого порта в памяти для безопасности и повышает скорость работы микроконтроллера при работе с внешними устройствами, особенно в условиях внешних электрических помех.

Прерывания (INTERRUPTS)

Система прерываний – одна из важнейших частей микроконтроллера. Все микроконтроллеры AVR имеют многоуровневую систему прерываний. Прерывание прекращает нормальный ход программы для выполнения приоритетной задачи, определяемой внутренним или внешним событием.

Для каждого такого события разрабатывается отдельная программа, которую называют подпрограммой обработки запроса на прерывание (для краткости – подпрограммой прерывания), и размещается в памяти программ. При возникновении события, вызывающего прерывание, микроконтроллер сохраняет содержимое счетчика команд, прерывает выполнение центральным процессором текущей программы и переходит к выполнению подпрограммы обработки прерывания.

После выполнения подпрограммы прерывания осуществляется восстановление предварительно сохраненного счетчика команд PC и процессор возвращается к выполнению прерванной программы.

Для каждого события может быть установлен приоритет. Понятие приоритет означает, что выполняемая подпрограмма прерывания может быть прервана другим событием только при условии, что оно имеет более высокий приоритет, чем текущее. В противном случае центральный процессор перейдет к обработке нового события только после окончания обработки предыдущего.

Таймеры/счетчики (TIMER/COUNTERS)

Микроконтроллеры AVR имеют в своем составе от 1 до 4 таймеров/счетчиков с разрядностью 8 или 16 бит, которые могут работать и как таймеры от внутреннего источника тактовой частоты, и как счетчики внешних событий.

Их можно использовать для точного формирования временных интервалов, подсчета импульсов на выводах микроконтроллера, формирования последовательности импульсов, тактирования приемопередатчика последовательного канала связи. В режиме широтно-импульсной модуляции ШИМ (PWM) таймер/счетчик может представлять собой широтно-импульсный модулятор и используется для генерирования сигнала с программируемыми

частотой и скважностью. Таймеры/счетчики способны вырабатывать запросы прерываний, переключая процессор на их обслуживание по событиям и освобождая его от необходимости периодического опроса состояния таймеров. Поскольку основное применение микроконтроллеры находят в системах реального времени, таймеры/счетчики являются одним из наиболее важных элементов.

Сторожевой таймер (WDT)

Сторожевой таймер (WatchDog Timer) предназначен для предотвращения катастрофических последствий от случайных сбоев программы. Он имеет свой собственный RC-генератор, работающий на частоте 1 МГц. Как и для основного внутреннего RC-генератора, значение 1 МГц является приближенным и зависит прежде всего от величины напряжения питания микроконтроллера и от температуры. Таймер снабжен своим собственным предделителем входной частоты с программируемым коэффициентом деления, что позволяет подстраивать временной интервал переполнения таймера и сброса микроконтроллера.

Идея использования сторожевого таймера предельно проста и состоит в регулярном его сбрасывании под управлением программы или внешнего воздействия до того, как закончится его выдержка времени и не произойдет сброс процессора. Если программа работает нормально, то команда сброса сторожевого таймера должна регулярно выполняться, предохраняя поцессор от сброса. Если же микропроцессор случайно вышел за пределы программы (например, от сильной помехи по цепи питания) либо зациклился на какомлибо участке программы, команда сброса сторожевого таймера скорее всего не будет выполнена в течение достаточного времени и произойдет полный сброс процессора, инициализирующий все регистры и приводящий систему в рабочее состояние.

Аналоговый компаратор (AC)

Аналоговый компаратор (Analog Comparator) сравнивает напряжения на двух выводах (пинах) микроконтроллера. Результатом сравнения будет логическое значение, которое может быть прочитано из программы.

Выход аналогового компаратора можно включить на прерывание от аналогового компаратора. Пользователь может установить срабатывание прерывания по нарастающему или спадающему фронту или по переключению. Присутствует у всех современных AVR, кроме Mega8515.

Аналого-цифровой преобразователь (A/D CONVERTER)

Аналого-цифровой преобразователь (АЦП) служит для получения числового значения напряжения, поданного на его вход. Этот результат сохраняется в регистре данных АЦП. Какой из выводов (пинов) микроконтроллера будет являться входом АЦП, определяется числом, занесенным в соответствующий регистр.

Универсальный последовательный приемопередатчик (UART или USART)

Универсальный асинхронный или универсальный синхронно/асинхронный приемопередатчик (Universal Synchronous/Asynchronous Receiver and Transmitter – UART или USART) – удобный и простой последовательный интерфейс для организации информационного канала обмена микроконтроллера с внешним миром. Способен работать в дуплексном режиме (одновременная передача и прием данных). Он поддерживает протокол стандарта RS-232, что обеспечивает возможность организации связи с персональным компьютером. (Для стыковки МК и компьютера обязательно понадобится схема сопряжения уровней сигналов. Для этого существуют специальные микросхемы, например MAX232.)

Последовательный периферийный интерфейс SPI

Последовательный периферийный трехпроводный интерфейс SPI (Serial Peripheral Interface) предназначен для организации синхронного обмена данными между двумя устройствами. С его помощью может осуществляться обмен данными между микроконтроллером и различными устройствами, такими, как цифровые потенциометры, ЦАП/АЦП, FLASH-ПЗУ и др. С помощью этого интерфейса удобно производить обмен данными между несколькими микроконтроллерами AVR. Кроме того, через интерфейс SPI может осуществляться программирование микроконтроллера.

Двухпроводной последовательный интерфейс TWI

Двухпроводной последовательный интерфейс TWI (Two-wire Serial Interface) является полным аналогом базовой версии интерфейса I2C (двухпроводная двунаправленная шина) фирмы Philips. Этот интерфейс позволяет объединить вместе до 128 различных устройств с помощью двунаправленной шины, состоящей из линии тактового сигнала (SCL) и линии данных (SDA).

Интерфейс JTAG

Joint Test Action Group — специализированный аппаратный интерфейс, разработанный для тестирования собранных печатных плат. Разработан группой ведущих специалистов по проблемам тестирования электронных компонентов. Из-за широкой функциональности JTAG стал использоваться для отладки и программирования. На данный момент JTAG-интерфейс применяется при периферийном сканировании (тестировании печатных плат с установленными на них процессорами на наличие в цепях коротких замыканий, непропаек, западаний на 0 или 1). Управление JTAG-интерфейсом описывается в т. н. BSDL-файле, который предоставляется разработчиком каждой конкретной микросхемы. В большинстве случаев эти файлы находятся на сайте производителя микросхемы. Интерфейс JTAG был зарегистрирован в качестве промышленного стандарта IEEE Std 1149.1-1990. Четырехпроводный интерфейс JTAG используется для тестирования печатных плат, внутрисхемной отладки, программирования микроконтроллеров. Многие микрокон-

троллеры семейства Mega имеют совместимый с IEEE Std 1149.1 интерфейс JTAG или debugWIRE для встроенной отладки. Кроме того, все микроконтроллеры Mega с флэш-памятью емкостью 16 кбайт и более могут программироваться через интерфейс JTAG.

Модуль загрузки ISP

Модуль загрузки или модуль внутрисхемного программирования (ISP-In-System Programmability) является основным способом программирования микроконтроллеров и программируемых логических интегральных схем и других устройств с памятью Flash или EEPROM. Модуль использует какой-либо стандартный интерфейс, встроенный в программируемую микросхему для последовательного занесения в нее кодов программ или требуемых данных.

Внутренний тактовый генератор

Тактовый генератор вырабатывает импульсы для синхронизации работы всех узлов микроконтроллера. Внутренний тактовый генератор AVR может запускаться от нескольких источников опорной частоты (внешний генератор, внешний кварцевый резонатор, внутренняя или внешняя RC-цепочка). Минимальная допустимая частота ничем не ограничена (вплоть до пошагового режима). Максимальная рабочая частота определяется конкретным типом микроконтроллера и указывается Atmel в его характеристиках, хотя практически любой AVR-микроконтроллер с заявленной рабочей частотой, например, в 10 МГц при комнатной температуре легко может быть “разогнан” до 12 МГц и выше. Микроконтроллер ATtiny15L. Он содержит блок PLL для аппаратного умножения основной тактовой частоты в 16 раз. При номинальном значении последней 1,6 МГц получаемая вспомогательная периферийная частота равна 25,6 МГц. Эта частота может служить источником для одного из таймеров/счетчиков микроконтроллера, значительно повышая временное разрешение его работы.

Система реального времени (RTC)

RTC (Real Tie Clock) реализована во всех микроконтроллерах Mega и в некоторых кристаллах “classic”. Таймер/счетчик RTC имеет отдельный предделитель, который может быть программным способом подключен или к источнику основной тактовой частоты, или к дополнительному асинхронному источнику опорной частоты (кварцевый резонатор или внешний синхросигнал). Для этой цели зарезервированы два вывода микросхемы. Внутренний осциллятор оптимизирован для работы с внешним “часовым” кварцевым резонатором 32,768 кГц.

В последней новации семейства AVR-8 – 8/16-разрядном подсемействе ATxmega, при наличии того же ядра и общей архитектуре, кроме усовершенствованных таймеров /счетчиков и устройств аналогового интерфейса, встречаются принципиально новые периферийные блоки:

1) Система управления событиями для разгрузки процессора от пересылки сигналов между блоками МК.

2) Контроллер прямого доступа к памяти - ориентирован на пересылку пакетов данных между блоками МК без участия процессора.

AVR функционируют при напряжениях питания от 1,8 до 6,0 Вольт. Ток потребления в активном режиме зависит от величины напряжения питания и частоты, на которой работает микроконтроллер, и составляет менее 1 мА для 500 кГц,5 … 6 мА для 5 МГц и 8 … 9 мА для частоты 12 МГц. AVR могут быть переведены программным путем в один из трех режимов пониженного энергопотребления.

Режим холостого хода (IDLE). Прекращает работу только процессор и фиксируется содержимое памяти данных, а внутренний генератор синхросигналов, таймеры, система прерываний и сторожевой таймер продолжают функционировать. Ток потребления не превышает 2,5 мА на частоте 12 МГц.

Стоповый режим (POWER DOWN). Сохраняется содержимое регистрового файла, но останавливается внутренний генератор синхросигналов, и, следовательно, останавливаются все функции, пока не поступит сигнал внешнего прерывания или аппаратного сброса. При включенном сторожевом таймере ток потребления в этом режиме составляет около 80 мкА, а при выключенном – менее 1 мкА. (Все приведенные значения справедливы для напряжения питания 5 В).

Экономичный режим (POWER SAVE). Продолжает работать только генератор таймера, что обеспечивает сохранность временной базы. Все остальные функции отключены.

Сброс при снижении напряжения питания (BOD). Схема BOD (Brown-Out Detection) отслеживает напряжение источника питания. Если схема включена, то при снижении питания ниже некоторого значения она переводит микроконтроллер в состояние сброса. Когда напряжение питания вновь увеличится до порогового значения, запускается таймер задержки сброса. После формирования задержки внутренний сигнал сброса снимается и происходит запуск микроконтроллера.

В отчете представить:

1. Структурную схему МКК ATTiny 2313;
2. Назначение основных элементов структурной схемы.

Задание 1.3. Программная модель микроконтроллера

Программная модель микропроцессора представляет собой совокупность программно доступных ресурсов. В программную модель микроконтроллеров семейства AVR входят РОН, регистры ввода-вывода, память программ, оперативная память данных и энергонезависимая память данных, которые показаны на рис.1.2.

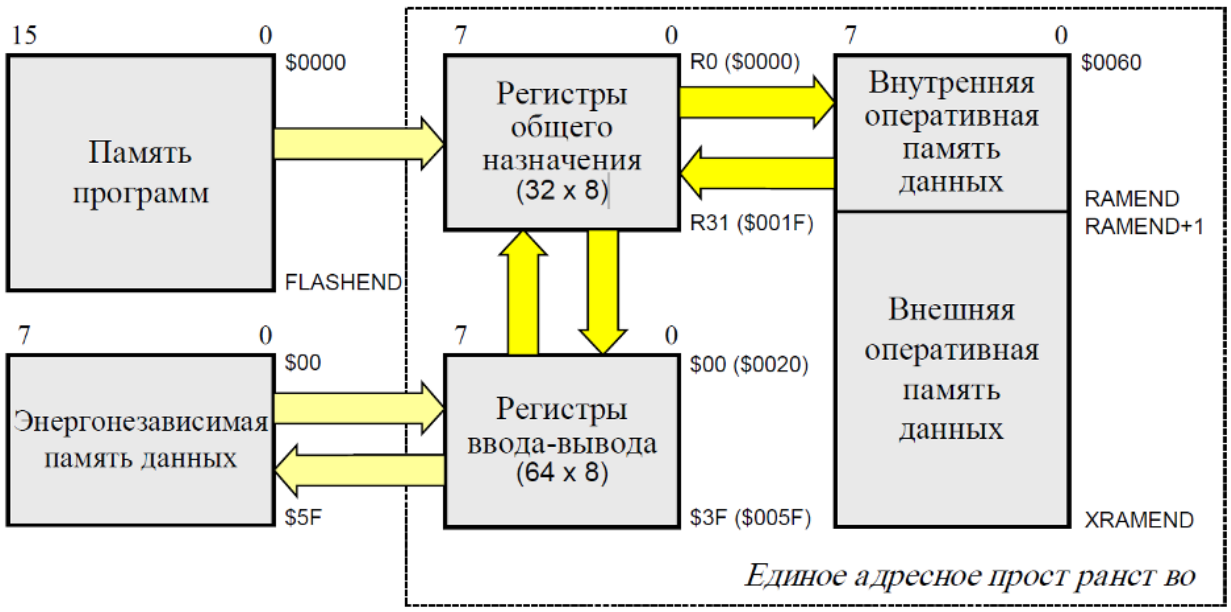

Рис.1.2. Программная модель AVR контроллера

РОН (R0…R31) могут использоваться в программе для хранения данных, адресов, констант и другой информации. Шесть старших регистров объединены попарно и составляют три 16-разрядных регистра X[R27:R26], Y[R29:R28] и Z[R31:R30] (рис.1.3).

РОН, регистры ввода-вывода и оперативная память данных образуют единое адресное пространство. Адресное пространство – это множество доступных ячеек памяти, различимых по адресам; адресом называется число, однозначно идентифицирующее ячейку памяти (регистр). Адреса ячеек памяти традиционно записываются в шестнадцатеричной системе счисления, на что указывает знак $ в обозначении адреса.

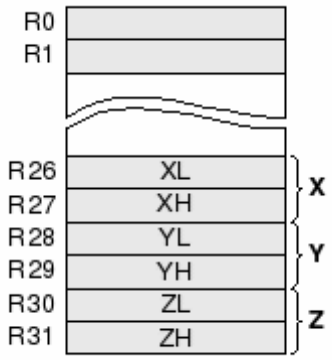

Рис.1.3. Регистры микроконтроллера AVR

Существует две конфигурации единого адресного пространства памяти AVR-микроконтроллеров. В конфигурации "A" младшие 32 адреса ($0000…$001F) соответствуют РОН, следующие 64 адреса ($0020…$005F) занимают регистры ввода-вывода, внутренняя оперативная память данных начинается с адреса $0060. В конфигурации "B" начиная с адреса $0060 размещаются 160 дополнительных регистров ввода-вывода; внутренняя оперативная память данных начинается с адреса $0100. Конфигурация Аиспользуется в младших моделях микроконтроллеров и в некоторых старших моделях в режиме совместимости с моделями, снятыми с производства; конфигурация B – в старших моделях.

В память программ, кроме собственно программы, могут быть записаны постоянные данные, которые не изменяются в процессе работы микро-

процессорной системы (константы, таблицы линеаризации датчиков и т. п.). Выполнение программы при включении питания или после сброса микроконтроллера начинается с команды, находящейся по адресу $0000 (т. е. в первой ячейке) памяти программ.

Энергонезависимая память данных предназначена для хранения информации, которая может изменяться непосредственно в процессе работы микропроцессорной системы (калибровочные коэффициенты, конфигурационные параметры и т. п.). Энергонезависимая память данных имеет отдельное адресное пространство и может быть считана и записана программным путём.

В отчете представить:

1. Программную модель МКК ATTiny 2313;
2. Уточненные параметры элементов программной модели: размеры типов памяти, разрядность регистров, счетчика команд, указателя стека и т.д..

Вопросы для самоконтроля

1. Чем отличается "структура" от "архитектуры"?
2. Какие внешние устройства могут быть в составе микроконтроллера?
3. Какие типы памяти есть в микроконтроллере и каково их назначение?
4. Каково назначение интерфейса SPI?
5. Для чего создается модуль загрузки ISP?
6. Как используются регистры общего назначения?
7. Каково назначение аналогового компаратора?
8. Каково назначение последовательного приемопередатчика?
9. Каково назначение последовательного интерфейса TWI?
10. Что входит в состав программной модели микроконтроллера?

2. УПРАВЛЕНИЕ ПОРТАМИ МИКРОКОНТРОЛЛЕРА

Цель:

1. Углубить и закрепить теоретические знания по принципам построения и работы специализированных контроллеров;

2. Приобрести практические навыки самостоятельного управления портами контроллера;

3. Совершенствовать навыки анализа, обобщения и систематизации полученных результатов, навыки составления и оформления отчетных материалов, навыки точного и лаконичного представления докладов на вопросы технического характера.

Учебные вопросы:

2.1. Программная модель портов микроконтроллера;
2.2. Способы адресации операндов;
2.3. Разработка программы управления портами.

Литература для подготовки к занятию

1. Белов А.В. Создаем устройства на микроконтроллерах. - СПб.: Наука и техника, 2007. - 304 с.
2. Евстифеев А.В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя. - М.: Издательский дом Додэка-XXI, 2007. - 432с.
3. Ревич Ю.В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера. СПб.: БХВ-Петербург, 2011. - 352с.
4. Егоров А.А. и др. Программирование микроконтроллеров семейства AVR. Учебно-методическое пособие. - М.: Издательство МАИ, 2004. - 72с.

Содержание отчета

1. Название работы.

2. Название каждого учебного вопроса и краткий конспект в объеме практических заданий по вопросу.

Вопросы для подготовки к занятию

1. Алгоритм работы центрального процессора;
2. Принципы работы оперативной памяти и Flash-памяти;
3. Характеристика гарвардской и фон-Неймановской архитектур;
4. В чем отличие CISC и RISC структур системы команд;
5. Чем для процессора является порт?
6. В чем отличие обращений процессора к порту и памяти?

Актуальность занятия

Спецконтроллер являются основными устройствами, осуществляющим управление и обработку данных в системах управления. Работа его в режиме включения и выключения нагрузки является базовой, такие алгоритмы используются практически во всех системах.

Задание 2.1. Программная модель портов микроконтроллера;

Изучить программную модель параллельных портов МКК ATtiny2313 с целью практического их программирования.

В отчете представить:

1. Программную модель портов;
2. Адреса портов;
3. Назначение регистров портов.

Назначение и возможности портов AVR

Порты ввода/вывода **(I/O)** AVR имеют число независимых линий “вход/выход” от 3 до 53. Каждая линия порта может быть запрограммирована на вход или на выход. Мощные выходные драйверы обеспечивают токовую нагрузочную способность 20 мА на линию порта (втекающий ток) при максимальном значении 40 мА, что позволяет, например, непосредственно подключать к микроконтроллеру светодиоды и биполярные транзисторы. Общая токовая нагрузка на все линии одного порта не должна превышать 80 мА (все значения приведены для напряжения питания 5 В).

Архитектурная особенность построения портов ввода/вывода у AVR заключается в том, что для каждого физического вывода (пина) существует 3 бита контроля/управления, а не 2, как у распространенных 8-разрядных микроконтроллеров (Intel, Microchip, Motorola и т.д.). Это позволяет избежать необходимости иметь копию содержимого порта в памяти для безопасности и повышает скорость работы микроконтроллера при работе с внешними устройствами, особенно в условиях внешних электрических помех.

Порты AVR имеют следующие особенности:

1. Прием и выдачу данных (для Tiny 6-20 выводов, для Mega - 23-86 выводов);
2. Программного изменения направления передачи;
3. Наличие триггеров Шмитта на входах, что обеспечивает восстановление прямоугольной формы сигналов после искажения в линиях связи (рис.2.1);
4. Возможность программного подключения подтягивающего резистора, что упрощает схему

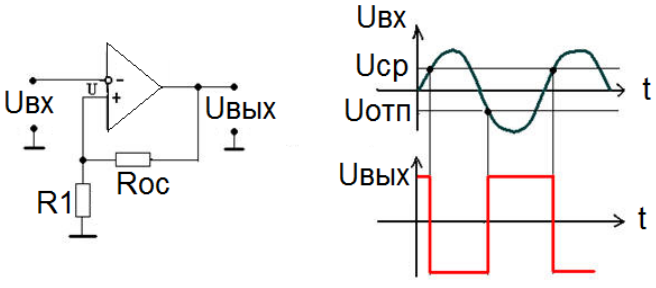

Рис.2.1. Диаграммы работы триггера Шмитта

и увеличивает гибкость управления;

5. Программное управление отдельными битами портов;

6. Совмещение возможности ввода-вывода и функций управления для выводов ИМС, что также организуется программно.

Программная модель портов AVR

В **AVR**-микроконтроллерах в зависимости от типа имеется от двух до шести универсальных двунаправленных портов ввода-вывода (порты **A**…**F**), содержащих семь или восемь сигнальных линий, которые соединены с соответствующими выводами БИС микроконтроллера. Например, в микроконтроллере **ATTiny2313** имеется три порта **A**, **B**, и **D**. Сигнальные линии порта **A** обозначаются **PA0**…**PA7**, порта **B** – **PB0**…**PB7** и т. д. Для МКК ATtiny2313 форматы параллельных портов показаны на рис.2.2.

С каждым портом связаны три регистра ввода-вывода:

- регистр направления передачи данных **DDR*x*.** Он задает направление передачи данных по каждой сигнальной линии порта (0 – разряд используется для ввода данных, 1 – для вывода данных);

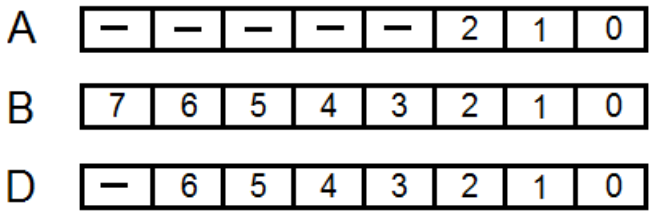

Рис.2.2. Форматы регистров ATtiny2313

- регистр данных порта **PORT*x*.** Он позволяет задавать состояние сигнальных линий порта при выводе информации;

- регистр выводов порта **PIN*x*.** Он позволяет считывать состояние сигнальных линий порта при вводе информации.

Для МКК ATtiny2313 имена и адреса таких портов следующие:

PORTA=$1D	PORTB=$18	PORTD=$12
DDRA=$1A	DDRB=$17	DDRD=$11
PINA=$19	PINB=$16	PIND=$10

С помощью указанных регистров реализуется требуемый протокол обмена данными. Здесь под словом протокол понимается совокупность правил передачи или прима кодированной информации. Основными процедурами при реализации некоторого протокола являются операции инициализации порта, вывода данных в порт, ввода данных из порта, а также формирования временных задержек.

Для обращения к отдельным разрядам портов зарезервированы имена этих разрядов (x - это одно из имен портов: A, B, D):

- для регистра PINx: PINx0 ... PINx7;
- для регистра DDRx: DDx0 ... DDx7;
- для регистра PORTx: PORTx0 ... PORTx7 или Px0 ... Px7.

Инициализация порта состоит в назначении направления обмена данными по сигнальным линиям порта. Например, если сигнальные линии **PA0**, **PA2**, порта **A** служат для вывода данных, а сигнальные линии **PA1** – для ввода данных, инициализация порта может быть произведена следующим образом:

```
ldi     R16, 0b00000101
out     DDRA, R16
```

Вывод данных в порт производится путём записи значений в регистр **PORT*x***, например:

```
ldi     R16, 0b10011101
out     PORTA, R16        ; вывод содержимого регистра R16 в порт A.
```

Для изменения состояния только одного разряда порта ввода-вывода удобно использовать команды установки SBI (*Set Bit in I/O Register*) и сброса CBI (*Clear Bit in I/O Register*) бита в регистре ввода-вывода.

Ввод данных из порта производится путём чтения содержимого регистра **PIN*x*** выводов порта, например:

```
in      R16, PINA         ; считывание состояния порта A в регистр R16.
```

Регистр данных ***PORTx*** в зависимости от режима работы порта выполняет различные функции:

- если порт запрограммирован на вывод, то в нем хранятся выдаваемые данные;
- если порт запрограммирован на ввод, то
  1 - подключает подтягивающий резистор;
  0 - отключает подтягивающий резистор

Варианты конфигурирования портов AVR можно свести в таблицу 2.1.

Таблица 2.1

Состояние элементов портов в различных режимах

Направление DDRB	Данные PORTB	Подключен. R PUD	Rподтяг	Вывод
1 = вых	D	*	Откл	Вых = D
0 = вход	1	0=вкл	Подключен	Ист. тока
		1=откл	Откл	Z-состояние
	0	*	Откл	Z-состояние

По умолчанию разряд ***PUD*** (*Pull Up Disable*) равен **0** (R – подтягивающий резистор подключен)**.** ***PUD*** **–** 7-й разряд регистра MCUCR=$35 (*MCU Control Register*) - Общий регистр управления микроконтроллера.

Исходное состояние портов (устройств) по умолчанию

После формирования команды RESTART исходное положение аппаратуры следующее:

- устройства выключены;
- порты запрограммированы на ввод (DDRx=0);
- подтягивающие резисторы подключены;
- значения разрядов обнулены.

Задание 2.2. Способы адресации операндов

Изучить команды пересылки МКК, уточнить особенности выполнения каждой команды в среде AVR Studio и сделать выводы об используемых способах адресации операндов в МКК. При необходимости воспользоваться директивами ассемблера, упомянутыми в тексте и изученными ранее. Список команд ассемблера представлен в приложении.

В отчете представить:

1. Примеры записи групп команд пересылки;
2. Диапазон доступных адресов в команде;
3. Указание способа адресации операндов в команде.

В зависимости от количества используемых операндов возможны три типа команд AVR-микроконтроллера: безадресные, одноадресные и двухадресные [4]. В первом типе команд присутствует только код операции (КОП), определяющий выполняемую командой функцию. В командах второго и третьего типов помимо кода операции содержится адресная часть, устанавливающая способ доступа соответственно к одному или двум участвующим в команде операндам (аргументам команды). Способ формирования адреса операнда, указание на который содержится в команде, называется адресацией (addressing). С помощью того или иного способа адресации вычисляется физический адрес, который поступает на шину адреса процессора для выбора ячейки памяти или регистра, используемых в команде.

В соответствии с типом адресуемой памяти способы адресации в AVR-микроконтроллерах можно разделить на способы адресации РОН и регистров ввода-вывода, способы адресации оперативной памяти данных (ОЗУ) и способы адресации памяти программ. Возможность использования различных способов адресации позволяет сократить размер и время выполнения программ.

Для адресации РОН и регистров ввода-вывода предусмотрен всего один режим – прямая регистровая адресация.

При *прямой регистровой адресации РОН* операндом является содержимое регистра общего назначения, указанного в команде. Команды с прямой регистровой адресацией могут адресовать один (**Rd**) или два (**Rr** и **Rd**) РОН. Во втором случае результат выполнения команды сохраняется в регистре **Rd**. Прямая регистровая адресация РОН применяется во всех арифметических и логических командах, а также в некоторых командах работы с битами, т. к. эти команды выполняются в АЛУ только над содержимым РОН. Команды, вторым операндом которых является константа, могут использовать в качестве первого операнда только регистры из старшей половины РОН (**R16**…**R31**).

При *прямой регистровой адресации регистра ввода-вывода* операнд содержится в регистре ввода-вывода, указанном в команде. Адрес регистра ввода-вывода хранится в шести разрядах слова команды. Прямая регистровая адресация регистров ввода-вывода используется в командах чтения ***IN*** и записи ***OUT*** регистра ввода-вывода, а также в ряде других команд работы с регистрами ввода-вывода.

Для адресации оперативной памяти данных используются пять способов адресации: непосредственная, косвенная, косвенная со смещением, косвенная с преддекрементом и косвенная с постинкрементом.

1. При *непосредственной адресации оперативной памяти данных* операндом является содержимое ячейки ОЗУ, адрес которой указан в команде. Адрес операнда содержится в 16 младших разрядах 32-разрядной команды. Непосредственная адресация используется в команде ***LDS*** (*Load Direct from Data Space*) загрузки из ОЗУ и в команде ***STS*** (*Store Direct to Data Space*) загрузки в ОЗУ. Зарезервировать байты в ОЗУ позволяет директива ***.byte***. Для того чтобы на выделенную область памяти можно было ссылаться, директиве ***.byte*** должна предшествовать метка. Директива ***.byte*** имеет один параметр – количество выделяемых байт и может использоваться только в сегменте данных, определяемом с помощью директивы ***.dseg*** (*data segment*). Задать требуемое размещение выделяемой области памяти позволяет директива ***.org*** (*origin* – смещение). Начало программного сегмента указывается с помощью директивы ***.cseg*** (*code segment*). Директивы ***.dseg*** и ***.cseg*** не имеют параметров. Выделенные в оперативной памяти данных байты не инициализируются.

2. При *косвенной адресации оперативной памяти данных* операндом является содержимое ячейки ОЗУ, адрес которой находится в регистре **X**, **Y** или **Z**. Косвенная адресация используется в команде ***LD*** (*Load Indirect*) кос-венной загрузки из ОЗУ и команде ***ST*** (*Store Indirect*) косвенной загрузки в ОЗУ.

3. При *косвенной адресации оперативной памяти данных со смещением* адрес операнда в оперативной памяти данных вычисляется путём прибав-

ления к содержимому регистра **Y** или **Z** смещения, указанного в команде. Смещение содержится в шести разрядах слова команды. Косвенная адресация со смещением используется в команде ***LDD*** (*Load Indirect with Displacement*) косвенной загрузки из ОЗУ со смещением и в команде ***STD*** (*Store Indirect with Dis-placement*) косвенной загрузки в ОЗУ со смещением.

4. При *косвенной адресации оперативной памяти данных с предекрементом* (лат. *decrementum* – уменьшение, убыль) перед выполнением команды содержимое указанного в команде регистра **X**, **Y** или **Z** декрементируется (уменьшается на единицу); декрементированное содержимое регистра X, Y или Z является адресом операнда в оперативной памяти данных. Косвенная адресация с предекрементом используется в команде LD косвенной загрузки из ОЗУ и команде ST косвенной загрузки в ОЗУ.

5. При *косвенной адресации оперативной памяти данных с постинкрементом* (лат. *incrementum* – увеличение, рост) адресом операнда в оперативной памяти данных является содержимое регистра **X**, **Y** или **Z**, указанного в команде; после выполнения команды содержимое регистра **X**, **Y** или **Z** инкрементируется, т. е. увеличивается на единицу. Косвенная адресация с постинкрементом используется в команде ***LD*** косвенной загрузки из ОЗУ и команде ***ST*** косвенной загрузки в ОЗУ.

Для адресации памяти программ используется непосредственная адресация, косвенная адресация, относительная адресация, адресация константы и адресация константы с постинкрементом.

При *непосредственной адресации памяти программ* выполнение программы продолжается с адреса, указанного в команде. Непосредственная адресация памяти программ используется в командах ***JMP*** и ***CALL***.

При *косвенной адресации памяти программ* выполнение программы продолжается с адреса, содержащегося в регистре Z, т. е. в программный счётчик загружается содержимое регистра Z. Косвенная адресация памяти программ используется в командах ***IJMP*** и ***ICALL***.

При *относительной адресации памяти программ* выполнение программы продолжается с адреса (**PC** + k + 1), где **PC** – содержимое программного счётчика; k – указанный в команде относительный адрес, который может принимать значения от –2048 до 2047. Относительная адресация памяти программ используется в командах ***RJMP*** и ***RCALL.***

При *адресации константы* адрес байта константы содержится в регистре **Z**. Адресация константы в памяти программ используется в командах ***LPM*** (*Load Program Memory*), которая загружает адресованный регистром **Z** байт в указанный в команде регистр. Если регистр-приёмник не указан (команда используется без операндов), байт загружается в регистр **R0**. Команда ***ELPM*** (*Extended Load Program Memory*) служит для загрузки константы из памяти программ объёмом более 64К слов. При этом для расширения регистра-указателя **Z** используется регистр **RAMPZ**, связанный с регистром **Z**.

Задать данные в память программ позволяет директива ***.db*** (*define bytes*). Для того, чтобы на заданные ячейки памяти можно было ссылаться, директиве должна предшествовать метка. Параметры директивы – последовательность выражений, разделённых запятыми; каждое выражение должно быть числом в диапазоне –128…255 или в результате вычисления давать результат в этом же диапазоне, в противном случае число усекается до байта. Директива ***.db*** размещается в программном сегменте и может использоваться совместно с директивой ***.org.*** Задавать положение данных в памяти программ следует таким образом, чтобы была исключена возможность непреднамеренного перехода к выполнению их как команд микроконтроллера.

При *адресации константы в памяти программ с постинкрементом* адрес байта константы содержится в регистре **Z** и формируется так же, как и при адресации константы. Адресованный регистром **Z** байт загружается в указанный регистр; после выполнения команды содержимое регистра **Z** инкрементируется. Адресация константы в памяти программ с постинкрементом используется в командах ***LPM*** и ***ELPM***.

В диалоговом окне ***AVR Simulator Options*** в разделе ***Device Selection*** установить тактовую частоту моделирования работы микроконтроллера, равную 8,0 МГц (поле ***Frequency***).

Выполнить трансляцию и отладку созданных программ. По данным, выводимым после трансляции на закладке ***Build*** окна ***Output***, проанализировать использование памяти программ (***Program memory usage***) под код программы (***Code***) и константы (***Constants***), оценить объём неиспользованной (***Unused***) и общей занятой (***Total***) памяти.

При отладке программ использовать средства наблюдения за содержимым регистров и ячеек памяти. По полю ***Cycle Counter*** объекта ***Processor*** закладки ***I/O*** окна ***Workspace*** определить число тактов выполнения программы, по полю ***Stop Watch*** – время выполнения программы (до выполнения команды, организующей бесконечный цикл). Зафиксировать эти сведения в отчёте. По результатам выполнения программ сделать выводы о правильности выполнения команд.

Задание 2.3. Разработка программы управления портами

Разработать схему алгоритма и программу в соответствии с последующими заданиями. Выполнить трансляцию и отладку программы. Продемонстрировать работу программы на эмуляторе и в программе Proteus. Представить данные в отчете в соответствии с каждым заданием.

Задание 2.3.1. Управление светодиодом от кнопки

Разработать устройство управления одним светодиодом при помощи одной кнопки. При нажатии кнопки светодиод должен зажечься, при отпускании - погаснуть.

Разработка алгоритма управления опирается на конкретную схему аппаратных средств. Поэтому решение должно начинаться с разработки схемы.

Управление светодиодом и работа с кнопкой осуществляется с помощью портов. Желательно порты назначить разные (рис.2.3). Это позволит упростить настройку портов: светодиод подключается к порту, настроенному на вывод, а кнопка к порту, настроенному на ввод.

Светодиод VD1 включен через токоограничивающий резистор R3 и зажигается при наличии нуля на выходе контакта PB.0. При появлении на выводе логической единицы потенциалы на светодиоде выравниваются, и он не светится. При необходимости зажигать светодиод высоким потенциалом, его следовало подключить к общему проводу.

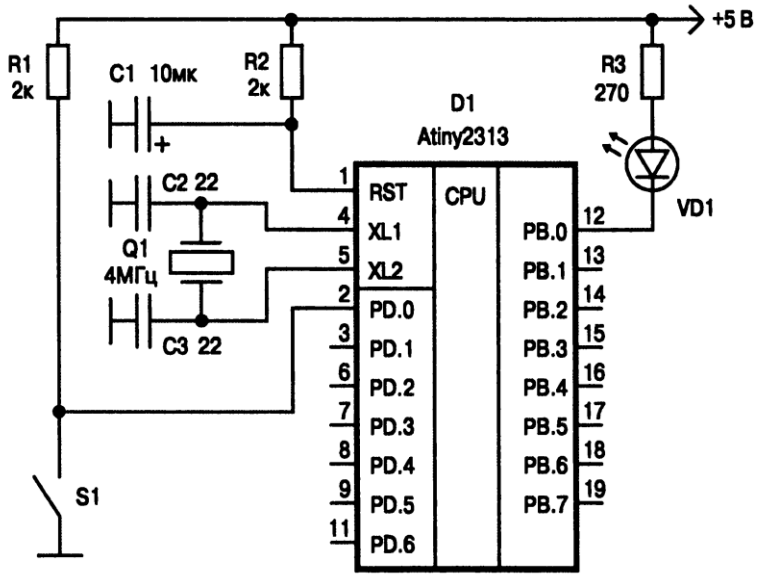

Рис.2.3. Схема подключения светодиода и кнопки

В исходном состоянии контакты кнопки S1 разомкнуты и через резистор R1 на вход PD.0 подается плюс источника питания, что соответствует логической единице. При нажатии кнопки потенциал контакта падает до нуля, что соответствует логическому нулю. Это классическое подключение кнопки.

Для контроллеров AVR схему можно упростить, подключив вместо внешнего резистора резистор встроенный (подтягивающий).

Кроме того, схема начального сброса C1R2 также может быть исключена, потому что в контроллерах AVR имеется внутренняя схема сброса, которая обеспечивает стабильный сброс, а высоких помех и нестабильного питания не предполагается.

Можно обойтись и без внешнего кварца, воспользовавшись встроенным генератором. Но его можно оставить в расчете на будущие задачи. Все настройки внешних цепей можно обеспечить fuse-переключателями. Поэтому схема должна получиться как показана на рис.2.4. Квар для упрощения следует также исключить.

Алгоритм управления светодиодом.

Выполнить операции начальной настройки. Далее в непрерывном цикле опрашивать кнопку и по ее состоянию управлять светодиодом.

Операции начальной настройки:

1. Инициализировать стек;
2. Настроить порт B на вывод;
3. Подать на выход PB.0 единицу (потушить светодиод);
4. Сконфигурировать пор D на ввод;
5. Подключить подтягивающие резисторы;

Операции тела цикла:

6. Прочитать состояние кнопки: разряда PD.0;
7. Если состояние разряда равно единице (кнопка не нажата), выключить светодиод;
8. Если значение разряда PD.0 равно нулю (кнопка нажата), включить светодиод;
9. Перейти на начало цикла.

Структура программы управления светодиодом на Ассемблере показана на рис.2.5.

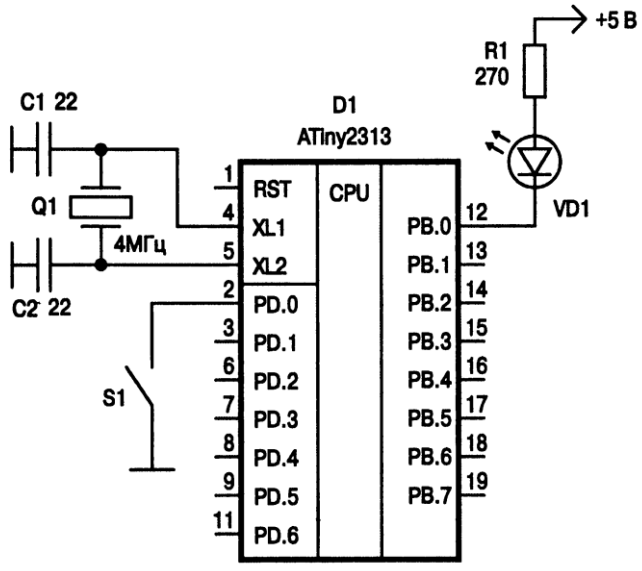

Рис.2.4. Упрощенная схема подключения кнопки и светодиода

```
;###############################################
;##                Пример 1                   ##
;##     Программа управления светодиодом      ##
;###############################################

,------------------------ Команды управления

1  .include "tn2313def.inc"          ; Присоединение файла описаний
2  .list                             ; Включение листинга

3  .def   temp = R16                 ; Определение главного рабочего регистра

;------------------------ Начало программного кода

4            .cseg                   ; Выбор сегмента программного кода
5            .org    0               ; Установка текущего адреса на ноль

;------------------------ Инициализация стека

6            ldi     temp, RAMEND    ; Выбор адреса вершины стека
7            out     SPL, temp       ; Запись его в регистр стека

;------------------------ Инициализация портов ВВ

8            ldi     temp, 0         ; Записываем 0 в регистр temp
9            out     DDRD, temp      ; Записываем этот ноль в DDRD (порт PD на ввод)

10           ldi     temp, 0xFF      ; Записываем число $FF в регистр temp
11           out     DDRB, temp      ; Записываем temp в DDRB (порт PB на вывод)
12           out     PORTB, temp     ; Записываем temp в PORTB (потушить светодиод)
13           out     PORTD, temp     ; Записываем temp в PORTD (включаем внутр. резист.)

;------------------------ Инициализация компаратора

14           ldi     temp, 0x80      ; Выключение компаратора
15           out     ACSR, temp

;------------------------ Основной цикл

16 main:     in      temp, PIND      ; Читаем содержимое порта PD
17           out     PORTB, temp     ; Пересылаем в порт PB
18           rjmp    main            ; К началу цикла
```

Рис.2.5. Структура программы на Ассемблере

.include "tn2313def.inc" - псевдооператор (директива ассемблера) присоединения текста. Текст вставляется в том месте, где установлен оператор. Параметром является файл. В данном случае это файл описаний микроконтроллера. Стандартное место его размещения ***C:\Program Files\Atmel\AVR Tools\AvrAssembler\Appnotes\.*** В частности из указанного файла транслятор узнает имена и параметры конкретного микроконтроллера:

```
RAMEND  EQU     0x00DF
DDRD    EQU     $11
SPL     EQU     $3D  и т.д.
```

.list - включение генерации листинга;

.def temp = R16 - макроопределение. Присваивает регистрам содержательные имена для увеличения наглядности и читабельности текста программ.

.cseg - выбор сегмента кодов в памяти программ Flash. До объявления другого сегмента все коды будут располагаться в выбранной памяти.

Остальные строки программы необходимо разобрать по справочнику команд и уяснить логику работы программы.

Практическое задание

1. Набрать программу в среде разработки, запустить и устранить выявленные ошибки.
2. Промоделировать схему с помощью эмулятора AVR Studio.
3. Набрать схему в среде Proteus без внешнего кварца и промоделировать работу программы.

В отчете представить:

1. Схему подключения кнопки и светодиода (желательно в среде Proteus).
2. Программу на языке ассемблера с необходимыми пояснениями.
3. Выбранные типы элементов схемы в программе Proteus.

Структура программы на языке С

Вариант программы

```
/*****************************************
ПРИМЕР МИГАНИЯ СВЕТОДИОДОМ 10 РАЗ
*****************************************/

  #define F_CPU 1000000UL      // указываем частоту в герцах
  #include<avr/io.h>           // подключение вв-выв по типу МКК
  #include<util/delay.h>       // подкл. функций циклов задержки

// ОПРЕДЕЛЕНИЕ ГЛОБАЛЬНЫХ ПЕРЕМЕННЫХ

     int main(void)            // начало основной программы
     {
     CLKPR = 0x80;             //Коэфделен.частоты сист. генератора
     CLKPR = 0x00;             //Коэф=

     PORTA = 0x00;             //Инициализация порта ВВ А(на вход)
     DDRA = 0x00;              //R подтягивающие ОТКЛ (Z состояние)
```

```
                                //Инициализация порта B
                                //
        PORTD = 0xFF;           //Инициализация порта BB D (на выход)
        DDRD = 0x00;            //Все единицы на выход

        TCCR0A = 0x00;          //Инициализация таймера/счетчика 0
        TCCR0B = 0x00;          //Источник – системный генератор
        TCNT0 = 0x00;           //Значение частоты – таймер 0 остановл.
                                //Режим Normal макс значение FFh
        OCR0A = 0x00;           //Выход OCR0A - отключен
        OCR0B = 0x00;           // Выход OCR0B–отключен

        TCCR1A = 0x00;          //Инициализация таймера/счетчика1
        TCCR1B = 0x00;          //Источник – системный генератор
        TCNT1H = 0x00;          //Значение частоты – таймер 1остановл.
        TCNT1L = 0x00;          //Режим Normal макс значение FFFFh
        ICR1H = 0x00;           //Noise Canceler: Off
        ICR1L = 0x00;           //Input Capture on Falling Edge
        OCR1AH = 0x00;          //Выход OCR0A –отключен
        OCR1AL = 0x00;          //
        OCR1BH = 0x00;          // ВыходOCR0B – отключен
        OCR1BL = 0x00;          //

        GIMSK = 0x00;           //Инициализация внешних прерываний
        MCUCR = 0x00;           //INTO0 – выкл;  INTO1 – выкл

        TIMSK = 0x00;           //Инициализация прерываний от таймеров

        USICR = 0x00;           //Инициализацунив последов интерф USI
                                //режим - выкл
                                //Clock source: Register&Counter=no clk
                                //USI Counter OverflovInterrupt: off

        ACSR = 0x80;            //Инициализация аналогового компаратора
                                //Компаратор выключен
                                //Вход от Сч 1 выключен



        int i;                  // объявляем переменную i
        DDRD = 0xff;            // все выводы порта D на выход
Cikl:   for (i = 1; i <= 10; i++) // цикл "для", повтор 10 раз
```

```
{
    PORTD |= _BV(PD1);        //"1" на PD1 (зажечь светодиод)
    _delay_ms(250);           //задержка 0.25 сек.
    PORTD &= ~_BV(PD1);       //"0" на PD1 (погасить светодиод)
    _delay_ms(250);           // задержка 0.25 сек.
}                             // закрывающая скобка цикла
      //  goto Cikl;
      }                       // закрывающая скобка осн.Прогр.
```

После присоединения файла описаний размещается описание глобальных переменных.

Настройка системы по умолчанию производится автоматически после системного сброса. Однако включение команд начальной настройки обеспечивает правильную работу программы в случае ошибки программы и передачи управления на начало программы, а также при срабатывании сторожевого таймера.

Настройка производится всех портов, аналогично тому, как это делалось на ассемблере.

Далее задается непрерывный цикл. Это обязательный элемент программы любого микроконтроллера.

Функция **_BV()** выполняет поразрядный сдвиг на число разрядов, указанное в скобках. Например:

```
PORTD |= _BV(PD3);       // установить "1" в разряде 3 порта D;
PORTD &= ~_BV(PD4);   // установить "0" на линии 4 порта D.
```

Эта запись аналогична операндам:

```
PORTD |= 1<<3;
PORTD &= ~(1<<4).
```

Практическое задание:

1. Реализовать вариант программы на языке С, убедиться в правильности компиляции. Использовать только строки инициализации требуемых портов и стека.

2. Устранить выявленные ошибки (при необходимости).

3. Проверить работу программы на эмуляторе AVR Studio. Добиться ее работоспособности.

3. Проверить работу программы на языке С с использованием команды ***goto Cikl*** и без нее. Сделать вывод о работоспособности обоих вариантов. Указать возможную причину получившегося результата;

4. Проверить инициализацию стека без использования явных команд инициализации.

5. Промоделировать работу программы в среде Proteus. Зафиксировать использованные элементы схемы.

В отчете представить:

1. Схему алгоритма задачи;
2. Текст программы на языке С (с указанием только строк инициализации портов);
3. Схему подключения внешних дискретных элементов на эмуляторе. Типы используемых компонентов.
4. Значение указателя стека в программе на языке С.

Задание 2.3.2. Управление переключением светодиода от кнопки

Используя предыдущую схему, обеспечить при каждом нажатии кнопки переключение светодиода в противоположное состояние.

Алгоритм управления от кнопки:

1. Прочитать состояние младшего разряда PD.0;
2. Если значение разряда равно единице (кнопка не нажата), то перейти к началу цикла;
3. Если значение разряда равно нулю (кнопка нажата), то изменить состояние выхода PB.0 на противоположное. Для переключения светодиода предварительно необходимо проверить его текущее состояние;
4. Перейти на начало цикла.

Практическое задание

1. Составить схему алгоритма.
2. Разработать программу и добиться правильной компиляции.
3. Проверить работоспособность программы на эмуляторе AVR Studio.
4. Промоделировать работу программы в среде Proteus. Зафиксировать использованные элементы схемы. В качестве кнопки можно использовать компонент BUTTON ACTIVE.

Если светодиод мигает часто и устойчивое его состояние назначается хаотически, то, вероятно, это происходит потому, что за время нажатия он успевает многократно переключиться, а момент отпускания является случайным. Для устранения этого явления следует после переключения светодиода добавить цикл ожидания отпускания кнопки (когда PD.0 будет равен единице). Добавить в программу этот фрагмент и проверить работоспособность устройства.

В отчете представить:

1. Схему алгоритма задачи;
2. Текст программы на языке программирования (только основной цикл);

3. Схему подключения внешних дискретных элементов на эмуляторе, если она изменилась по отношению к предыдущему заданию.

Задание 2.3.3. Управление переключением светодиода от кнопки с защитой от дребезга

Используя предыдущую схему, обеспечить при каждом нажатии кнопки переключение светодиода в противоположное состояние. Алгоритм должен предусмотреть исключение дребезга контактов кнопки.

Дребезг это эффект выдачи от элемента коммутации при одном нажатии серии импульсов за счет упругих свойств механических элементов конструкции. Это приводит к тому, что на вход порта поступает целая серия переключений вместо одного. Самый простой программный способ борьбы с этим явлением - введение задержек. Обнаружив первое переключение, программа должна выдержать задержку не менее времени дребезга контактов.

В отчете представить:

1. Схему алгоритма задачи;
2. Текст программы на языке программирования (только основного цикла);
3. Уточнить схему подключения внешних дискретных элементов на эмуляторе, если она изменилась по отношению к предыдущему заданию.

Задание 2.3.4. Управление переключением мигающего светодиода

Используя исходную схему, нажатием кнопки необходимо включать и выключать мигание светодиода. Частота мигания светодиода 5 Гц (для обеспечения возможности наблюдения мигания частоту допускается изменить, но с указанием ее реального значения).

Алгоритм:

1. Прочитать порт PD;
2. Если младший разряд равен нулю, включить алгоритм мигания;
3. Если младший разряд равен единицы, выключить алгоритм мигания и выключить светодиод;
4. Перейти к началу основного цикла (п.1).

Алгоритм мигания:

1. Зажечь светодиод;
2. Выдержать паузу;
3. Потушить светодиод;
4. Выдержать паузу;

5. Перейти к п.1.

В отчете представить:

1. Схему алгоритма задачи;
2. Текст программы на языке программирования (с указанием только основного блока);
3. Уточнить схему подключения внешних дискретных элементов на эмуляторе, если она изменилась по отношению к предыдущему заданию.

Задание 2.3.5. Бегущие огни

Обеспечить бегущий огонь из одного светодиода по гирлянде из восьми светодиодов. Направление движения гирлянды должно изменяться нажатием кнопки. Частота мигания светодиода 1 Гц.

Задача решается в том же порядке, как и предыдущие.

В отчете представить:

1. Схему алгоритма задачи;
2. Текст программы на заданном языке программирования (с указанием только основного блока);
3. Уточнить схему подключения внешних дискретных элементов на эмуляторе, если она изменилась по отношению к предыдущему заданию.

Вопросы для самоконтроля

1. Какие адреса занимают порты ввода-вывода контроллера?
2. Какие регистры управляют портами?
3. Какие режимы можно задать портам?
4. Какова типовая программная модель порта?
5. Какие способы адресация РОН и регистров ввода-вывода **AVR**-микроконтроллеров существуют?
6. Какие способы адресации памяти данных **AVR**-микроконтроллеров используются на практике?
7. Какие используются способы адресации памяти программ **AVR**-микроконтроллеров?
8. Особенности выполнения арифметических и логических операций в **AVR**-микроконтроллерах.
9. В чем отличие команд безусловного перехода ассемблера КР580 и микроконтроллера?
10. Для чего предназначена директива .def ?
11. Назначение и использование регистров **X**, **Y** и **Z**.
12. Как осуществляется инициализация стека на языке ассемблера и языке С?
13. Какова структура программы на языке ассемблера и языке С.

14. Как записать на ассемблере команду или команды, эквивалентные команде присваивания на языке С?
15. Как записать на ассемблере команду или команды, эквивалентные командам цикла на языке С?
16. Просмотрите машинные коды, получающиеся после компиляции с языка С.
17. Какое состояние имеют порты регистров РОН и РВВ по умолчанию?
18. Каково назначение функции _BV()?
19. Каково назначение функции _delay() ?

3. УПРАВЛЕНИЕ СЧЕТЧИКАМИ МИКРОКОНТРОЛЛЕРА

Цель:

1. Углубить и закрепить теоретические знания по принципам построения и работы специализированных контроллеров;

2. Приобрести практические навыки самостоятельного управления счетчиками контроллера;

3. Совершенствовать навыки анализа, обобщения и систематизации полученных результатов, навыки составления и оформления отчетных материалов, навыки точного и лаконичного представления докладов на вопросы технического характера.

Учебные вопросы:

3.1. Программная модель таймеров-счетчиков микроконтроллера;
3.2. Задание параметров счетного сигнала;
3.3. Разработка программы управления счетчиками в режиме Normal;
3.4. Разработка программы управления счетчиками в режиме CTC;
3.5. Разработка программы управления счетчиками в режиме PWM.

Литература для подготовки к занятию

1. Белов А.В. Создаем устройства на микроконтроллерах. - СПб.: Наука и техника, 2007. - 304 с.
2. Евстифеев А.В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя. - М.: Издательский дом Додэка-XXI, 2007. - 432с.
3. Ревич Ю.В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера. СПб.: БХВ-Петербург, 2011. - 352с.
4. Егоров А.А. и др. Программирование микроконтроллеров семейства AVR. Учебно-методическое пособие. - М.: Издательство МАИ, 2004. - 72с.

Содержание отчета

1. Название работы.

2. Название каждого учебного вопроса и краткий конспект в объеме практических заданий по вопросу.

Вопросы для подготовки к занятию

1. Как работает счетчик? Какие режимы реализует счетчик?
2. Какие существуют методы построения делителей?
3. Какие сигналы выдает схема сравнения?
4. На что влияет разрядность счетчика?
5. Как посчитать максимальный код счетчика?

6. Что такое прерывание? Как оно отрабатывается процессором?

Актуальность занятия

Спецконтроллер являются основными устройствами, осуществляющим управление и обработку данных в системах управления. Контроль временных интервалов, реакция на окончание времени таймеров, формирование широтно-импульсных последовательностей для управления объектами являются основными управляющими функциями микроконтроллера. Поэтому понимание этих процессов и умение их программировать является обязательной для специалиста.

Задание 3.1. Программная модель таймеров-счетчиков микроконтроллера

Изучить программную модель счетчиков МКК ATtiny2313 с целью практического их программирования.

В отчете представить:

1. Программную модель счетчиков с указанием всех портов и их адресов;
2. Структуру регистра управления и таблицы задания режимов работы счетчиков;
3. Назначение внешних выводов счетчиков.

Назначение и возможности счетчиков AVR

Счетчики контроллера решают различные задачи при выполнении управленческих функций. Все они связаны в основном с необходимостью формирования точных временных интервалов, причем без дополнительной нагрузки на процессор, а также с подсчетом числа событий:

- измерение и задание временных интервалов;
- подсчет количества внешних событий (импульсов);
- формирование одиночных и периодических импульсов с постоянными и переменными параметрами (частота, длительность импульса, количество импульсов, фазы);
- преобразование временного интервала в код для АЦП;
- фиксация состояния счетчика по внешнему сигналу (захват) и т.д.

В микроконтроллерах бывает до десяти счетчиков. В ATtiny2313 имеется два счетчика: однобайтный T0 и двухбайтный T1.

Программная модель счетчиков AVR

В **AVR**-микроконтроллере ATtiny2313 имеется два счетчика: 8-разрядный T0 и 16-разрядный T1. Структура их почти совпадает, за исключением разрядности и средств реализации захвата счетчика для T1. На рис.3.1. представлена внутренняя структура счетчика T1.

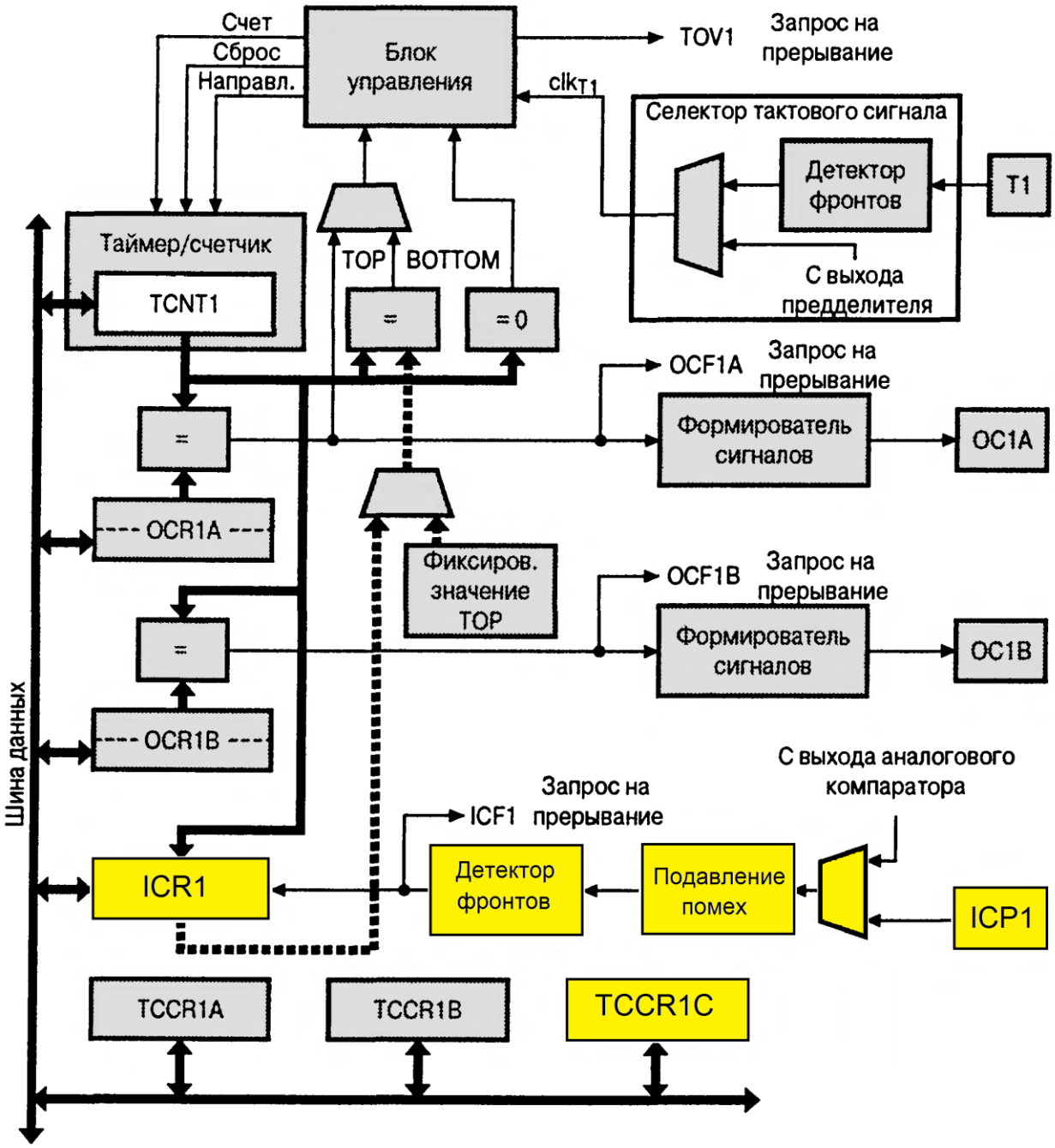

Рис.3.1. Структурная схема счетчика T1 ATtiny2313

Назначение внешних выводов таймера T1 (в скобках указаны аналогичные выводы для таймера T0):

- T1 (T0) - вход внешнего сигнала таймера. На этот вход подаются внешние подсчитываемые сигналы;

- OC1A (OC0A), OC1B (OC0B) - выход схемы сравнения регистра A и B таймера соответственно. Сигнал формируется при совпадении кода счетчика и кода в регистре A или B;

- ICP1 - вход захвата таймера T1. По этому сигналу код счетчика запоминается в регистре ICR1 и может быть прочитан при выполнении обработчика прерывания ICP. В счетчике T0 такого входа и режима не предусмотрено.

Расположение выводов таймеров микроконтроллера показано на рис.3.2. Выводы таймеров совпадают с выводами цифровых портов контроллера и подключаются к счетчикам при установке единиц, настраивающих поведение этих выводов, в регистрах управления счетчиками (см. табл.3.3)

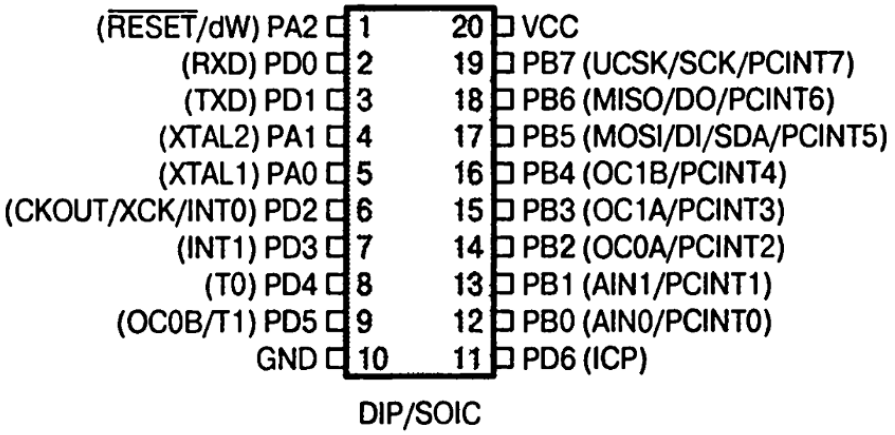

Рис.3.2. Выводы микроконтроллера ATtiny2313

Количество регистров ввода-вывода, имеющихся в конкретном таймере-счетчике, зависит от сложности, разрядности и возможностей последнего, количества генерируемых прерываний.

Таймер-счетчик T1 (в скобках приведены данные для счетчика-таймера T0) имеет следующие порты, их адреса и назначение (см. рис.3.1).

Счетчик таймера TCNT1 (TCNT0)

TCNT1 (TCNT0=$32) - реверсивный 16-разрядный (8-разрядный) счетчик таймера T1 (T0). В исходном положении (после подачи питания) счетчик обнулен. Может быть сброшен, выполнять инкремент или декремент по каждому тактовому сигналу. Доступен для чтения и записи в любой момент времени. В зависимости от режима счета счетчик при достижении максимального или минимального значения устанавливается флаг TOV1 (TOV0) и генерируется прерывание. 16-разрядный счетчик TCNT1 представлен двумя 8-разрядными портами: старший бай - TCNT1H=$2D и младший байт TCNT1L=$2C. Такое деление обусловлено 8-разрядной шиной данных. За-

пись данных в счетчик должно осуществляться двумя последовательными посылками.

Регистры совпадения OCR1A, OCR1B

OCR1A и **OCR1B** (OCR0B=$3C) - 16-разрядные (восьмиразрядный) регистры совпадения. В каждом такте счета происходит непрерывное сравнение их значений со счетчиком таймера TCNT1 (TCNT0). При совпадении - в следующем такте устанавливается флаг OC1A (OC0A) или OC1B (OC0B) и генерируется прерывание. Кроме того, при совпадении может меняться состояние выводов OC1A (OC0A) и OC1B (OC0B), если они сконфигурированы как выходы (этот бит в регистре DDRx должен быть равен 1). Аналогично счетчику таймера T1, 16-разрядные регистры совпадения счетчика T1 адресуются побайтно: регистр OCR1A представлен старшим байтом OCR1AH=$2B и младшим байтом OCR1AL=$2A, регистр OCR1B соответственно - OCR1BH=$29 и OCR1BL=$28.

Регистр захвата таймера ICR1

ICR1 - *Input Capture Registers* - регистр захвата таймера T1. Представлен двумя портами: старший байт - ICR1H=$25 и младший байт - ICR1L=$24. Регистр предназначен для сохранения состояния счетчика в режиме захвата при поступлении активного фронта (тип фронта может быть настроен) от входа ICP1, либо по сигналу аналогового компаратора. Одновременно с записью в регистр захвата устанавливается флаг ICF1 - *Input Capture Flag* - и формируется запрос на прерывание. Для захвата по сигналу ICP1 этот вывод должен быть сконфигурирован на вход (бит регистра DDRx должен иметь 0). Копирование счетчика в регистр захвата происходит с задержкой от 2,5 до 3,5 тактов от момента поступления первого фронта. Задержку вносят синхронизатор и схема подавления помех. Если же включена схема подавления помех, которая контролирует еще 4 выборки с частотой тактового генератора, то задержка увеличивается еще на 4 такта.

Доступ к 16-разрядным регистрам

Для чтения и записи в 16-разрядные регистры через 8-разрядную шину данных используется регистр временного хранения в автоматическом режиме. Первая пересылка осуществляется с временным регистром, а вторая завершает обмен всех 16 разрядов.

Поэтому для записи в 16-разрядный регистр необходимо сначала записать старший байт, а затем младший. При чтении 16-разрядного регистра сначала нужно прочитать младший байт, а затем старший.

Регистры управления TCCR1A, TCCR1B, TCCR1C

TCCR1A=$2F (TCCR0A=$30), **TCCR1B**=$2E (TCCR0B=$33), **TCCR1C**=$22 - регистры управления счетчиком-таймером. Они задают возможные режимы работы таймеров и осуществляют все другие настройки счетчиков-таймеров: форму сигналов на выходе, частоту счета, источник тактового сигнала, особенности работы блока сравнения и т.д.

Регистры управления для таймера T0 и назначение разрядов показаны на рис.3.3. Аналогичные регистры существуют для T1 - на рис.3.4.

TCCR0A=$30 Регистр A управления счетчика T0
Timer/Counter Control Register

7	6	5	4	3	2	1	0
COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00
Режим работы блока сравнения A (вывод OC0A)		Режим работы блока сравнения B (вывод OC0B)				Режим работы таймера/счетчика	

TCCR0B=$33 Регистр B управления счетчика T0
Timer/Counter Control Register

7	6	5	4	3	2	1	0
FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
Принудительное изменение состояния вывода OC0A	Принудительное изменение состояния вывода OC0B			Режим работы таймера/счетчика	Управление тактовым сигналом (источник сигнала)		

Рис.3.3. Регистры управления таймером T0

TCCR1A=$2F Регистр A управления счетчика T1 (аналог TCCR0A=$30) *Timer/Counter Control Register*							
7	6	5	4	3	2	1	0
COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10
Режим работы блока сравнения A (вывод OC1A)		Режим работы блока сравнения B (вывод OC1B)				Режим работы таймера/счетчика T1	

TCCR1B=$2E Регистр B управления счетчика T1 (частично аналог TCCR0B=$33) *Timer/Counter Control Register*							
7	6	5	4	3	2	1	0
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
Включение схемы подавления помех	Выбор фронта сигнала захвата		Режим работы таймера/счетчика		Управление тактовым сигналом (источник сигнала)		

TCCR1C=$22 Регистр C управления счетчика T1 *Timer/Counter Control Register*							
7	6	5	4	3	2	1	0
FOC1A	FOC1B	-	-	-	-	-	-
Принудительное изменение вывода OC1A (по COM1A1:COM1A0)	Принудительное изменение вывода OC1B (по COM1B1:COM1B0)						

Рис.3.4. Регистры управления счетчиком T1

Таймер T0 может работать в шести различных режимах, которые можно установить битами WGM0-WGM2 (Wave Generator Mode). Комбинации этих битов и соответствующие им режимы представлены в таблице 3.1.

Таблица 3.1

Режимы работы таймера T0

Номер режима	WGM02	WGM01	WGM00	Режим работы таймера/счетчика T0	Модуль счета (TOP)	Обновление регистров OCR0x	Момент установки флага TOV0
0	0	0	0	Normal	$FF	Немедленно	$FF
1	0	0	1	Phase correct PWM	$FF	При TOP	$00
2	0	1	0	CTC (сброс при совпадении)	OCR0A	Немедленно	$FF
3	0	1	1	Fast PWM	$FF	При TOP	$FF
4	1	0	0	Зарезервировано	—	—	—
5	1	0	1	Phase correct PWM	OCR0A	При TOP	$00
6	1	1	0	Зарезервировано	—	—	—
7	1	1	1	Fast PWM	OCR0A	При TOP	TOP

Счетчик Т1 является 16-разрядным. В отличие от Т0 счетчик реализует дополнительно режим захвата. Элементы структуры, обеспечивающие этот режим, показаны желтым цветом (тонкими линиями) на рис.3.1.

Возможные режимы счетчика Т1 представлены в таблице 3.2.

Режим Normal (таймера)

Это наиболее простой режим, который имеет место в любом контроллере:

- счетчик работает как суммирующий от тактового сигнала до максимального значения;
- при переполнении счетчика Т0 (код $FF меняется на $00) устанавливается флаг переполнения TOV0=1. Аналогично происходит в счетчике Т1 (код $FFFF меняется на код $0000, устанавливается флаг TOV1);
- при совпадении кода счетчика с регистрами сравнения устанавливается соответствующий флаг OCF0A или OCF0B (OCF1A или OCF1B) и генерируется прерывание (если есть маска и общее разрешение прерывания);
- при подключении выводов OC0A или OC0B (OC1A или OC1B) на них формируется сигнал, форма которого программируется битами COM (Compare Match Output Mode) в соответствии с таблицей 3.3.

Если хоть один из этих битов установлен в 1, то вывод OC0 перестает функционировать как обычный вывод общего назначения и подключается к схеме сравнения таймера счетчика. Однако при этом он должен быть еще настроен как выход. Настройка OC0A и OC0B (OC1A и OC1B) должна быть выполнена до конфигурирования выводов на выход.

Таблица 3.2

Режимы работы счетчика T1

Номер режима	WGM13	WGM12	WGM11	WGM10	Режим работы таймера/счетчика T1	Модуль счета (TOP)	Обновление регистров OCR1*x*	Момент установки флага TOV1
0	0	0	0	0	Normal	$FFFF	Немедленно	$FFFF
1	0	0	0	1	Phase correct PWM, 8-битный	$00FF	При TOP	$0000
2	0	0	1	0	Phase correct PWM, 9-битный	$01FF	При TOP	$0000
3	0	0	1	1	Phase correct PWM, 10-битный	$03FF	При TOP	$0000
4	0	1	0	0	CTC (сброс при совпадении)	OCR1A	Немедленно	$FFFF
5	0	1	0	1	Fast PWM, 8-битный	$00FF	При TOP	При TOP
6	0	1	1	0	Fast PWM, 9-битный	$01FF	При TOP	При TOP
7	0	1	1	1	Fast PWM, 10-битный	$03FF	При TOP	При TOP
8	1	0	0	0	Phase and Frequency Correct PWM	ICR1	$0000	$0000
9	1	0	0	1	Phase and Frequency Correct PWM	OCR1A	$0000	$0000
10	1	0	1	0	Phase correct PWM	ICR1	При TOP	$0000
11	1	0	1	1	Phase correct PWM	OCR1A	При TOP	$0000
12	1	1	0	0	CTC (сброс при совпадении)	ICR1	Немедленно	$FFFF
13	1	1	0	1	Зарезервировано	—	—	—
14	1	1	1	0	Fast PWM	ICR1	При TOP	При TOP
15	1	1	1	1	Fast PWM	OCR1A	При TOP	При TOP

Таблица 3.3

Управление выводами OC0A и OC0B в режиме Normal

COM0A1 (COM0B1)	COM0A0 (COM0B0)	Описание
0	0	Таймер/счетчик T0 отключен от вывода OC0A (OC0B)
0	1	Состояние вывода меняется на противоположное
1	0	Вывод сбрасывается в 0
1	1	Вывод устанавливается в 1

При необходимости состояние выводов OC0A и OC0B (OC1A и OC1B) могут быть изменены в соответствии с табл.5.3 принудительно (Force Output Compare) записью 1 в бит FOC0A и FOC0B (FOC1A и FOC1B) регистра TCCR0B (TCCR1B). Прерывание при этом не генерируется и в режиме CTC не сбрасывается счетчик. При чтении этот вывод будет нулем.

Режим CTC

Режим CTC (*Clear Timer on Compare* - сброс при совпадении) используется для генерации сигналов заданной частоты. Режим характеризуется следующими процессами:

- счетчик работает как суммирующий по тактовому сигналу;
- предельное значение счетчика определяется регистром сравнения A, затем сброс и опять счет начинается с нуля;
- после совпадения с регистром устанавливается флаг OCF0A (OCF1A), что может при разрешении сопровождаться прерыванием;
- одновременно с установкой флага может изменяться и состояние выводов OC0A (OC1A) микроконтроллера. Характер изменения зависит от бит COM0A1- COM0A0 (COM1A1- COM1A0). Управление выводами в этом режиме аналогично режиму Normal (см. табл.3.3);
- совпадение с регистром сравнения B, если его значение не превышает значения регистра A, также сопровождается выработкой прерывания и изменением состояния выхода OC0B (OC1B), но счетчик не сбрасывается.

Диаграммы режима CTC представлены на рис.3.5. Такие диаграммы получаются при задании режима переключения сигналов на выводах (биты управления выводами соответствуют 01). Легко видеть, что период генерации увеличивается по мере увеличения кода в регистрах сравнения. Наклонная линия иллюстрирует увеличение кода счетчика.

Для расчета частоты генерации F используется выражение

$$F = \frac{f_{CLK}}{2 \cdot N \cdot (1 + OCR0A)},$$

где F_{CLK} - тактовая частота микроконтроллера;
N - коэффициент деления предделителя;
OCR0A - код в регистре сравнения.

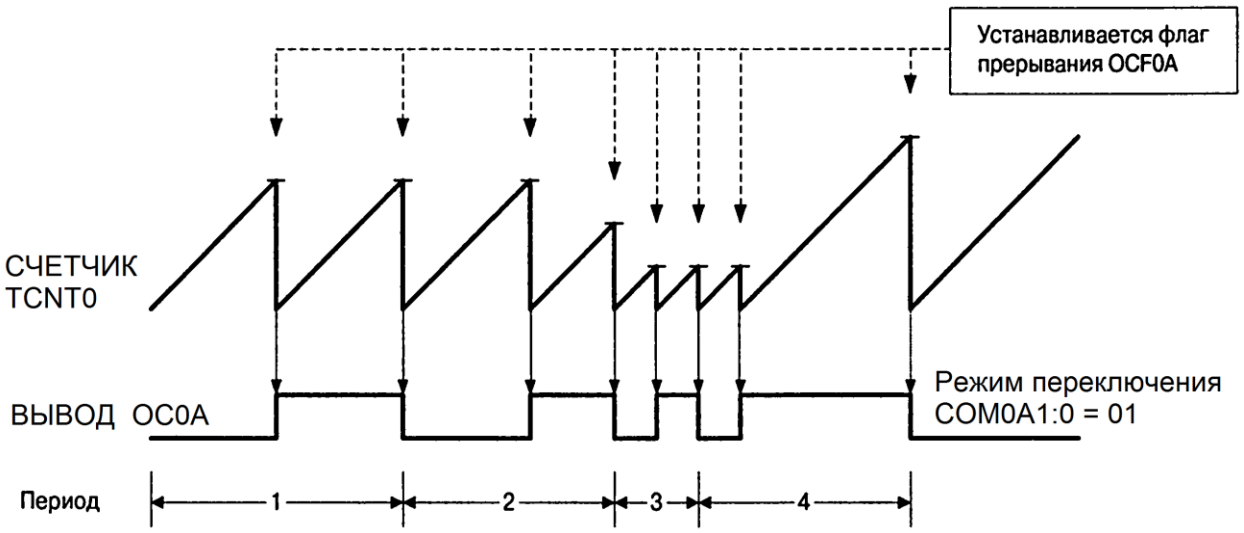

Рис.3.5. Временные диаграммы работы в режиме CTC

Режим Fast PWM

Режим Fast PWM (*Fast Puls-Wedth Modulation* - быстродействующей широтно-импульсной модуляции - ШИМ) используется для регулирования мощности, выпрямления, цифро-аналогового преобразования и т.д.

Режим характеризуется следующими процессами:

- счетчик работает в режиме суммирования по тактовому сигналу до максимального значения $FF. В альтернативном варианте при режиме №7 счетчика T0 (при WGM02=1) - до значения регистра сравнения OCR0A. В этом режиме запись в регистр сравнения происходит только при достижении счетчиком состояния $FF. Это называется двойной буферизацией записи. В противном случае появились бы несимметричные импульсы на выходе модулятора (формирователя сигналов, см. рис.3.1);

- при переполнении (переход через $FF) устанавливается флаг переполнения TOV0;

- при равенстве содержимого счетчика и регистра сравнения устанавливается флаг совпадения OCF0A или OCF0B;

- при равенстве содержимого счетчика и регистра сравнения еще происходит управление выводами в соответствии с таблицей 3.4.

Таблица 3.4

Управление выводами OC0A и OC0B в режиме Fast PWM

COM0A1 (COM0B1)	COM0A0 (COM0B0)	Описание
0	0	Таймер/счетчик T0 отключен от вывода OC0A (OC0B)
0	1	**OC0A:** WGM02 = 0 — таймер/счетчик T0 отключен от вывода OC0A; WGM02 = 1 — состояние вывода меняется на противоположное при равенстве регистров TCNT0 и OCR0A. **OC0B:** Зарезервировано
1	0	Сбрасывается в 0 при равенстве регистров TCNT0 и OCR0*x*. Устанавливается в 1 при достижении счетчиком максимального значения (неинвертированный ШИМ-сигнал)
1	1	Устанавливается в 1 при равенстве регистров TCNT0 и OCR0*x*. Сбрасывается в 0 при достижении счетчиком максимального значения (инвертированный ШИМ-сигнал)

Диаграммы сигналов в режиме Fast PWM представлены на рис.3.6.

Легко видеть, что при счете до $FF увеличением кода в регистре сравнения увеличивается степень заполнения единицами периода генерации. При необходимости сигнал на выходе можно проинвертировать, поменяв биты COM0A1- COM0A0 с 10 на 11.

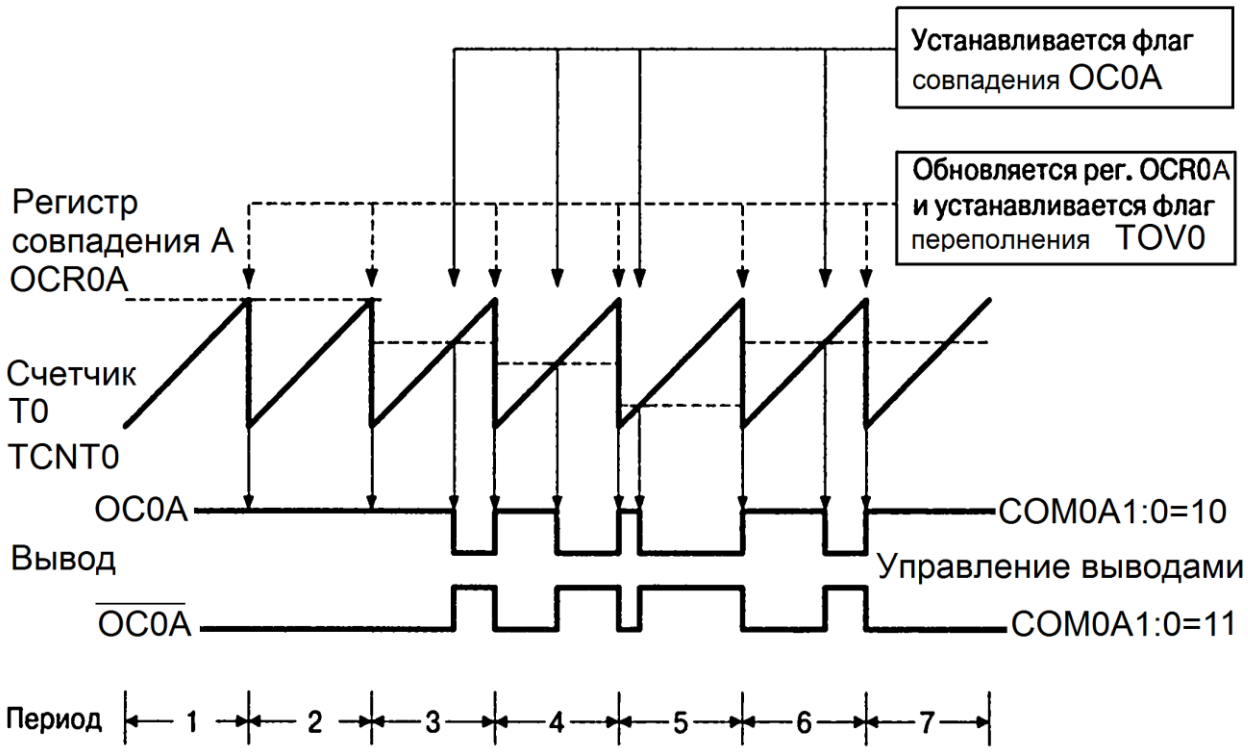

Рис.3.6. Диаграммы сигналов в режиме Fast PWM

Частота генерируемого сигнала определяется выражением:

$$F_{OC0A} = \frac{f_{CLK}}{N \cdot 256},$$

где N - коэффициент деления предделителя;
f_{CLK} - тактовая частота микроконтроллера.

Режим Fast Correct PWM

Режим Fast Correct PWM (ШИМ с точной фазой) предпочтительней для управления двигателями благодаря симметричности изменения состояния счетчика. В этом режиме реализуемые процессы очень похожи на режим Fast PWM, но с учетом выполнения реверсивного счета:

- счетчик суммирует от нуля до максимального значения, затем вычитает до нуля. Возможность ограничения максимального кода счетчика регистром сравнения также присутствует;
- при достижении кода $00 формируется флаг переполнения TOV0;
- при совпадении с регистром сравнения формируется флаг совпадения OC0A или OC0B;
- при совпадении с регистром сравнения происходит изменение сигнала на выходе в соответствии с таблицей 3.5.

Таблица 3.5

Управление выводами, OC0A, OC0B в режиме Fast Correct PWM

COM0A1 (COM0B1)	COM0A0 (COM0B0)	Описание
0	0	Таймер/счетчик T0 отключен от вывода OC0A (OC0B)
0	1	**OC0A:** WGM02 = 0 — таймер/счетчик T0 отключен от вывода OC0A; WGM02 = 1 — состояние вывода меняется на противоположное при равенстве регистров TCNT0 и OCR0A. **OC0B:** Зарезервировано
1	0	Сбрасывается в 0 при прямом счете и устанавливается в 1 при обратном счете (неинвертированный ШИМ-сигнал)
1	1	Устанавливается в 1 при прямом счете и сбрасывается в 0 при обратном счете (инвертированный ШИМ-сигнал)

С учетом реверсивного счета (за период счетчик два раза перебирает свои состояния) частота выходного сигнал будет в 2 раза меньше частоты в режиме Fast PWM:

$$F_{OC0A} = \frac{f_{CLK}}{N \cdot 512}$$

Биты FOC0A и FOC0B принудительно изменяют состояние вывода OC0A и OC0B (в режимах Normal и CTC). При записи 1 в этот бит состояние вывода изменяется в соответствии с установками битов COM0A1-COM0A0. Прерывание при этом не генерируется и сброс таймера (в режиме CTC) не производится.

Диаграмма сигналов в режиме Fast Correct PWM показана на рис.3.7.

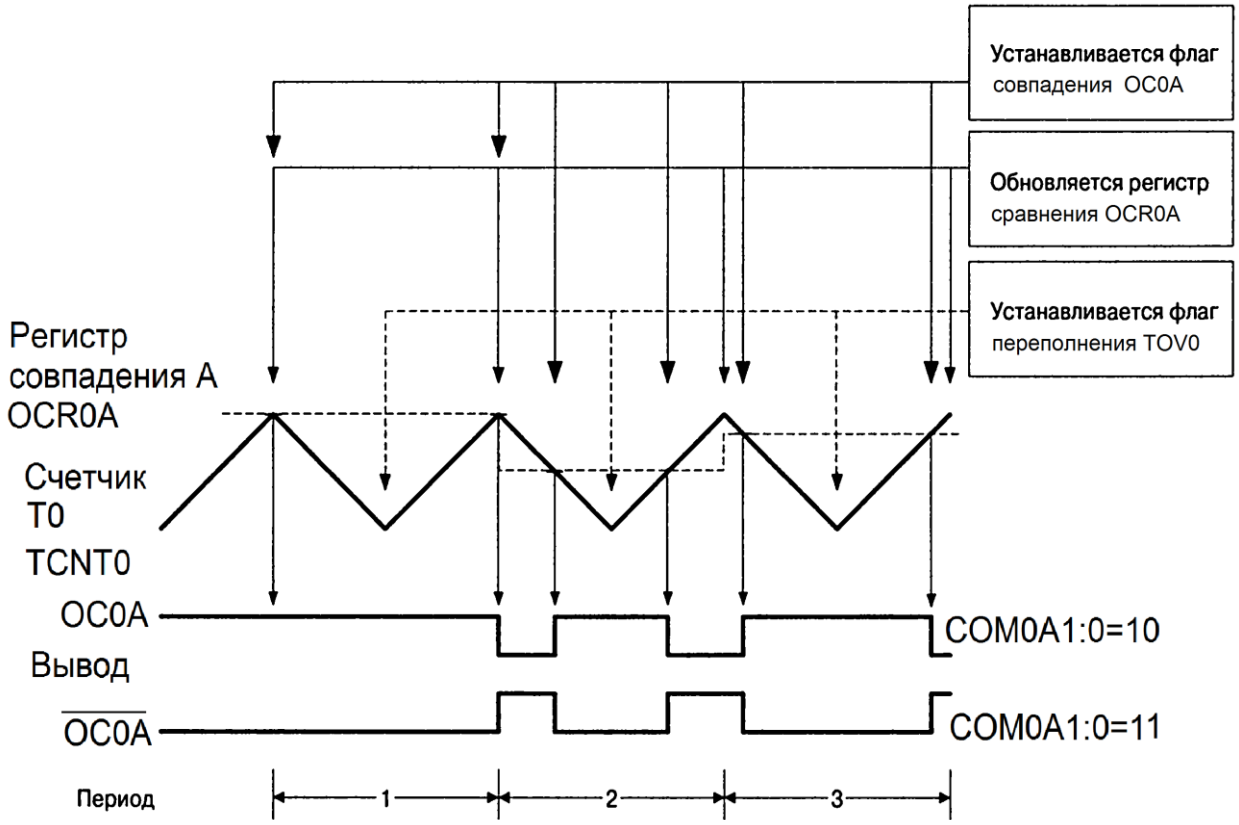

Рис.3.7. Диаграмма сигналов в режиме Fast Correct PWM

Работа счетчика T1 ATtiny2313 аналогична счетчику T0. Но в нем реализован дополнительно режим захвата и используются двухбайтные порты, что увеличивает диапазон используемых кодов. Структурные дополнения T1 по сравнению со счетчиком T0 выделены желтым цветом на рис.3.1.

Запись в двухбайтные регистры счетчика T1 осуществляется за два обращения. Сначала записывается старший байт, который сохраняется в буфере. При записи младшего байта происходит размещение двухбайтного кода в регистр таймера. Содержимое буфера, содержащего старший байт, можно использовать многократно при адресации к младшему байту выбранного двухбайтного регистра.

При чтении 16-битного регистра процесс должен начинаться с младшего байта. При этом старший байт записывается в буфер, из которого и происходит чтение старшего байта.

При обращении к 16-битным регистрам прерывания необходимо запретить во избежание изменения состояния регистров до завершения обращения.

Уточнить детали работы таймера T1 можно по литературе [2].

Исходное состояние счетчиков по умолчанию

После формирования команды RESTART исходное положение аппаратуры следующее:

- устройства выключены;
- предделитель выключен (счетчик не работает);
- внешние выходы отключены.

Задание 3.2. Задание параметров счетного сигнала

Изучить программную модель управления частой работы таймеров-счетчиков, научиться программировать требуемые параметры тактовых сигналов счетчиков-таймеров.

В отчете представить:

1. Адреса и имена регистров ввода-вывода, обеспечивающих управление предделителем;
2. Диапазон коэффициентов деления предделителя;
3. Назначение битов регистров управления предделителя.

Предделитель таймеров предназначен для формирования тактовых сигналов таймеров-счетчиков. Функциональная схема предделителя представлена на рис.3.8. Оба таймера используют один предделитель, но управляются тактовые сигналы индивидуально.

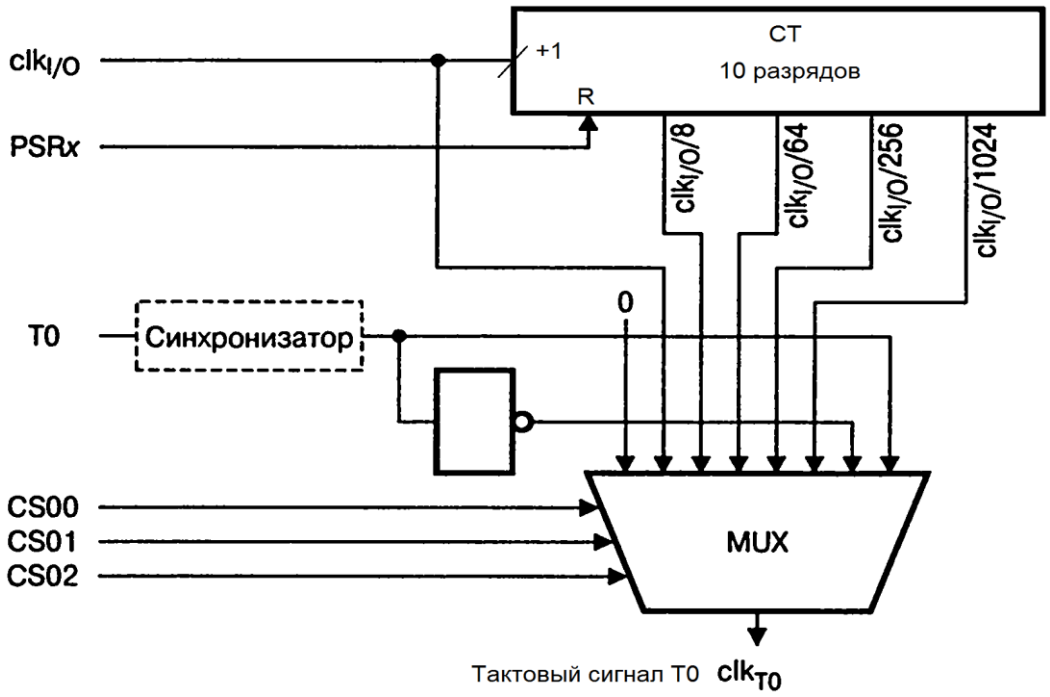

Рис.3.8. Функциональная схема предделителя таймеров T0 и T1

Управление предделителем заключается в задании коэффициента деления тактовой частоты контроллера, остановке таймеров, переключении на счет внешних сигналов. Возможные источники сигналов для таймера T0 назначаются битами регистра TCCR0B=$33 (рис.3.3) в соответствии с таблицей 3.6. Для счетчика T1 таблица аналогичная, но управление осуществляется битами в регистре TCCR1B=$2E (рис.3.4).

Таблица 3.6

Выбор источника тактового сигнала счетчиков T0 и T1

CS02	CS01	CS00	Источник тактового сигнала T0
0	0	0	Таймер/счетчик остановлен
0	0	1	$clk_{I/O}$
0	1	0	$clk_{I/O}/8$
0	1	1	$clk_{I/O}/64$
1	0	0	$clk_{I/O}/256$
1	0	1	$clk_{I/O}/1024$
1	1	0	Вывод T0, счет осуществляется по спадающему фронту импульсов
1	1	1	Вывод T0, счет осуществляется по нарастающему фронту импульсов

Установкой в 1 бита PSR10 (Prescaler Reset Timer/Counter) Общего регистра управления таймерами/счетчиками GTCCR (General Timer/Counter Control Register) можно сбросить предделитель в ноль.

Задание 3.3. Разработка программы управления счетчиками в режиме Normal

Разработать схему алгоритма и программу в соответствии с последующими заданиями. Выполнить трансляцию и отладку программы. Продемонстрировать работу программы на эмуляторе и в программе Proteus. Представить данные в отчете в соответствие с каждым заданием.

В самом общем случае инициализация таймера в режиме Normal (в остальных режимах аналогично) состоит из следующих шагов:

- остановка таймера в TCCR0B (обнуление разрядов CS00 - CS02):

```
TCCR0B (1<<CS02)|(1<<CS01)|(1<<CS00);
```

,

- задание режима работы Normal в TCCR0A и TCCR0B без старта (обнуление разрядов WGM02 . . .WGM00):

```
TCCR0A  &= ~ (1<<WGM01)|(1<<WGM00); //По умолчанию
TCCR0B &= ~ (1<<WGM02);
```

- задание режима работы выводов счетчика в TCCR0A (выводы COM0A и COM0B), например, в режим "изменение состояния на противоположное":

```
TCCR0A  &= ~ (1<<COM0A0)|(1<< COM0B0);
```

- установка цифровых портов "на выход" для подключения выводов счетчика OC0A и OC0B (регистры DDRB, DDRD):

```
DDRB |= (1<<PB2)       //для вывода OC0A
DDRD |= (1<<PD5)       //для вывода OC0B;
```

- установка начального значения счетчика TCNT0 (TCNT1) или регистров сравнения OCR0 (OCR1):

```
TCNT0 = const;                  //Или OCR0A=...  OCR0B= ...;
```

- сброс флагов счетчика-таймера T0 в регистре TIFR:

```
TIFR |= (1<<OCF0B)|(1<<TOV0)|(1<<OCF0A);
```

- разрешение прерываний счетчика-таймера T0 в TIMSK:

```
TIMSK |= (1<<TOIE0)|(1<<OCIE0B)|(1<<OCIE0A);
```

- установка предделителя счетчика-таймера T0 в TCCR0B, то есть старт таймера:

```
TCCR0B |= (1<<CS02)|(0<<CS01)|(1<<CS00).
```

Такие шаги могут быть выполнены в полном объеме или частично, исходя из текущих решаемых задач.

Задание 3.3.1. Генератор сигналов на основе таймера T0

Запрограммировать счетчик T0 в режиме Normal с максимальной частотой генерации с выхода OC0A. Режим выхода - изменение сигнала на противоположный. Тактовая частота процессора - 4 МГц.

В отчете представить:

1. Выражения для расчета частоты и периода генератора;
2. Программу на языке С. Структуру данных управляющих регистров таймера T0;
3. Схему моделирования с использованием осциллографа;
4. Расчеты частоты генерации по данным AVR Studio;

5. Результаты измерения частоты и периода генерации с помощью осциллографа. Вывод о соответствии данных расчета и измерения.

Задача 3.3.2. Изменить программу для генерации сигналов на выходах OC0A и OC0B со сдвигом на четверть периода. Сохранить в программе прежние настройки регистров управления с помощью знаков комментариев.

В отчете представить:

1. Изменения в программу задачи 3.3.1.
2. Выражения для задания режимов работы счетчика.
3. Вывод о фазовом сдвиге сигналов.

Задача 3.3.3. Уменьшить в 8 раз частоту генерации на выходах OC0A и OC0B. Доказать факт изменения частоты по сигналам на осциллографе и параметрам AVR Studio.

В отчете представить:

1. Изменения в программу задачи 3.3.2.
2. Данные расчета частоты генерации по данным AVR Studio и настройкам осциллографа. Вывод об их соответствии.

Задание 3.4. Разработка программы управления счетчиками в режиме CTC

Сигнал должен генерироваться на выходах OC0A или OC1A. Целесообразно использовать режим CTC, специально предназначенный для генерации сигналов настраиваемой частоты. Используя конкретный выбранный счетчик контроллера, обосновать возможность и запрограммировать контроллер на выдачу сигнала на выходе регистра A с частотой 1 Гц. Тактовая частота контроллера 4 МГц.

Пример расчета параметров настройки проектируемого генератора на счетчике T1 для частоты задающего генератора f_{CLK}= 8 МГц для выработки сигнала прерывания по переполнению с частотой 1 Гц.

Для расчета следует использовать следующие этапы:

1. Количество заполнений счетчика за 1 сек:

$$8\ 000\ 000 / 65536 = 122 \text{ раз};$$

2. Если генератор должен переключается всякий раз по переполнению, то переключений (миганий, периодов за секунду) получится:

$$122 / 2 = 61 \text{ раз};$$

3. Для приближения к 1 Гц необходимо уменьшить исходную тактовую частоту процессора в 64 раза. Это можно выполнить за счет коэффициента предделителя К=64. Поэтому получится частота:

61 / 64 = 0,954 Гц;

4. Соответственно, для получения точно 1 Гц следует уменьшить количество отсчетов счетчика в 0,954 раза:

65536 * 0,954 = 62521;

5. Следовательно, если до переполнения счетчик должен получить 62521 фронт тактового сигнала, то код предустановки:

65536 - 62521 = 3015 = 0BC7H.

При переключении генератора по сигналу прерывания по переполнению для получения частоты переключения 1 Гц необходимо будет этот код (0BC7H) записывать в счетчик T1 как код предустановки. В обработчике можно с этой же частотой управлять любым внешним устройством (за 1с будет выдаваться два сигнала переполнения). Кроме того на выходе OC1A при установке кода в регистре сравнения OCR1A, например, 0000H также будут происходить переключения с той же частотой в момент сброса счетчика.

Для расчета данных для режима CTC в рамках исходного задания необходимо учесть, что число тактов до сброса определяется кодом в регистре сравнения. Поэтому полученный при расчете код необходимо просто занести в регистр сравнения. Заполнение счетчика будет происходить от нуля до значения регистра сравнения, затем снова с нуля.

Кроме того при инициализации 16-разрядных регистров следует учитывать порядок записи и чтения младшего и старшего байта.

Проверку расчетов следует проводить по данным процессора в AVR Studio. Измерения частоты по данным осциллографа в системе моделирования будут искажены из-за чрезмерной нагрузки процессора.

В отчете представить:

1. Расчеты для получения необходимой частоты;
2. Программу для микроконтроллера на языке C;
3. Доказательства правильности расчетов.

Задание 3.5. Разработка программы управления счетчиками в режиме PWM.

Запрограммировать счетчик T0 в режиме ***Fast PWM*** с максимальной частотой генерации с выхода OC0A. Длительность единичного сигнала - 50% периода. Тактовая частота процессора - 4 МГц. После отладки и подтверждения результатов на выходе OC0Aзапрограммировать формирование сигнала на выходе OC0B с заполнением 20% и сравнить результаты в среде Proteus.

В отчете представить:

1. Выражения для расчета частоты и периода генератора для двух выходов;
2. Программу на языке С. Структуру данных управляющих регистров таймера Т0;
3. Расчеты частоты генерации по данным AVR Studio.
4. Диаграмму моделирования с использованием осциллографа;
5. Результаты измерения частоты и периода генерации с помощью осциллографа. Вывод о соответствии данных расчета и измерения.

Задание 3.6. Бегущие огни (задержка таймером без использования прерываний)

Проанализировать представленную программу. При отсутствии замечаний получить код для программирования микроконтроллера. Смоделировать работу бегущих огней в программе Proteus. При невозможности наблюдать эффект бегущих огней выявить причину и предложить способы устранения этого недостатка.

```
#include <avr/io.h>
void wait1 (void)          //  Функция задержки
{
      TCNT1=0;
      while (TCNT1<780) {};
}
int  main(void)            //              Главная функция программы
{
unsigned char rab;         // Вводим переменную rab

PORTB=0xFF;                // Инициализация порта B
DDRB=0xFF;

PORTD=0x7F;                // Инициализация порта D
DDRD=0x00;

TCCR1A=0x00;               // Инициализация таймера/счетчика 1
TCCR1B=0x05;

ACSR=0x80;                 // Инициализация аналогового компаратора

while (1)
```

```
{
if ((PIND &(1<<PD0)))     // Проверка состояния переключателя
        {                           // Сдвиг вправо
        rab = 0b100000000;      // Запись начального значения
        while (rab!=0)
                {
                PORTB=rab^0xFF;         // Запись а порт с инверсией
                rab = rab >> 1;          // Сдвиг разрядов
                wait1 ();                // Задержка в 200 мСек
                }
        }
else
        {                               // Сдвиг влево
        rab = 0b00000001;               // Запись начального значения
        while (rab!=0)
                {
                PORTB=rab^0xFF;         // Запись в порт с инверсией
                rab = rab << 1;         // Сдвиг разрядов
                wait1 ();               // Задержка в 200 мСек
                }
        }
}
}
```

В отчете представить:

1. Алгоритм решения задачи;
2. Предложения по усовершенствованию схемы;
3. Схему устройства в программе Proteus.

Вопросы для самоконтроля

1. Какое назначение имеют таймеры/счетчики?
2. Какие адреса занимают порты счетчиков контроллера?
3. Какие регистры управляют счетчиками?
4. Какие режимы можно задать счетчикам?
5. Какова типовая программная модель счетчиков?
6. Как осуществляется инициализация стека на языке ассемблера и языке С?
7. Какое состояние имеют регистры ввода-вывода счетчиков?
8. Какие режимы и каково исходное состояние счетчиков после команды RESET?

9. Почему вместо программного формирования временного интервала используют автономный счетчик?
10. Какими способами можно изменить частоту переполнения счетчиков?
11. Докажите, что на счетчике T0 построить генератор 1 Гц невозможно.
12. В каком случае вырабатывается сигнал сравнения с регистром B счетчика в режиме CTC?

4. ИССЛЕДОВАНИЕ ПРЕРЫВАНИЙ МИКРОКОНТРОЛЛЕРА

Цель:

1. Углубить и закрепить теоретические знания по принципам организации прерываний специализированных контроллеров;

2. Приобрести практические навыки самостоятельного использования прерываний для решения задач;

3. Совершенствовать навыки анализа, обобщения и систематизации полученных результатов, навыки составления и оформления отчетных материалов, навыки точного и лаконичного представления докладов на вопросы технического характера.

Учебные вопросы:

4.1. Организация прерываний в AVR микроконтроллерах;
4.2. Использование внешних прерываний;
4.3. Использование прерываний счетчиков.

Литература для подготовки к занятию

1. Белов А.В. Создаем устройства на микроконтроллерах. - СПб.: Наука и техника, 2007. - 304 с.
2. Евстифеев А.В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя. - М.: Издательский дом Додэка-XXI, 2007. - 432с.
3. Ревич Ю.В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера. СПб.: БХВ-Петербург, 2011. - 352с.
4. Егоров А.А. и др. Программирование микроконтроллеров семейства AVR. Учебно-методическое пособие. - М.: Издательство МАИ, 2004. - 72с.

Содержание отчета

1. Название работы.

2. Название каждого учебного вопроса и краткий конспект в объеме практических заданий по вопросу.

Вопросы для подготовки к занятию

1. Что называется прерыванием?
2. Источники прерываний?
3. Что называется контекстом при осуществлении прерывания?
4. Что такое маска прерывания? С какой целью используется?
5. Что называется приоритетом прерываний?
6. Какие прерывания используются в современных ЭВМ?
7. Этапы механизма прерывания.

Актуальность занятия

Микроконтроллер работает в режиме реального времени. Это требует реализации точных временных интервалов, при этом реализация их должна осуществляться без дополнительной загрузки процессора. Кроме того актуальными являются задачи генерации сигналов произвольной частоты, сигналов широтно-импульсной модуляции, осуществления АЦП и ЦАП преобразований и т.д. Все эти задачи решают встроенные таймеры-счетчики.

Задание 4.1. Организация прерываний в AVR микроконтроллерах

Изучить организацию прерываний в микроконтроллерах AVR, состав программных средств для управления прерываниями МКК ATtiny2313, способы получения справочной информации.

В отчете представить:

1. Размещение и структуру таблицы векторов прерываний;
2. Способы организации прерываний на языке ассемблера и С в виде примеров программ;
3. Состав регистров для управления прерываниями в МКК ATtiny2313;
4. Путь с макрорасширению с описанием векторов прерываний МКК.

Назначение и механизм реализации прерываний AVR

Прерывание (Interrupt) – сигнал, свидетельствующий о наступлении какого-либо события. При этом выполнение текущей последовательности команд приостанавливается, и управление передаётся процедуре обработки прерывания (обработчику), соответствующей данному событию. После завершения обработчика исполнение кода продолжается с команды, которая является следующей после той, на которой произошло прерывание.

Прерывания позволяют организовать и синхронизировать параллельно (одновременно) протекающие процессы.

Для реализации данной логики работы процессор выполняет следующие действия:

1. По сигналу прерывания процессор аппаратно запрещает все прерывания (сбрасывает глобальный флаг I в регистре состояния SREG) и сохраняет в стеке адрес следующей команды. Адрес следующей команды определяется как сумма текущего состоянию счетчика команд плюс длина последней команды, на которой произошло прерывание. Локальный флаг, вызвавший прерывание, должен быть сброшен программно в обработчике прерывания;
2. В счетчик команд загружается адрес вектора прерывания, соответствующий данному событию. Все события, вызывающие прерывания, имеют свой уникальный и неизменный адрес в таблице векторов. ***Вектором преры-***

вания называют код в таблице векторов, который закреплен за данным событием (это фиксированная строка программной области, определяющая, куда переходит программа в случае возникновения прерывания). По этому адресу, как правило, находится команда ***безусловного перехода к подпрограмме*** обработки прерывания (обработчику). Адреса векторов прерываний для микроконтроллера ATtiny2313 приведены в таблице 4.1. Всего в этом контроллере 19 прерываний. Прерывание с номером 0 особое, оно не имеет флага, не может быть маскировано. Уровень приоритета уменьшается по мере увеличения адреса размещения вектора прерывания.

Таблица 4.1

Прерывания микроконтроллера ATtiny2313

Номер вектора	Адрес перехода	Источник	Описание прерывания
0	0x0000	RESET	Внешний сброс, сброс при включении питания, сброс по срабатыванию охранного таймера
1	0x0001	INT0	Внешний запрос на прерывание по входу INT0
2	0x0002	INT1	Внешний запрос на прерывание по входу INT1
3	0x0003	TIMER1_CAPT1	Прерывание по захвату таймера/счетчика 1
4	0x0004	TIMER1_COMPA	Прерывание по совпадению таймера/счетчика 1, канал A(1=? в f)
5	0x0005	T1MER1_OVF1	Прерывание по переполнению таймера/счетчика 1
6	0x0006	TIMER0_OVF0	Прерывание по переполнению таймера/счетчика 0
7	0x0007	UART_RX	USART, прием завершен
8	0x0008	UART_UDRE	USART, буфер данных пуст
9	0x0009	UART_TX	USART, передача завершена
10	0x000A	ANA_COMP	Прерывание от аналогового компаратора
11	0x000B	PCINT	Прерывание по изменению на любом из выводов*
12	0x000C	TIMER1_COMPB	Прерывание по совпадению таймера/счетчика1, канал B*
13	0x000D	TIMER0_COMPA	Прерывание по совпадению таймера/счетчика0, канал A*
14	0x000E	TIMER0_COMPB	Прерывание по совпадению таймера/счетчика 0,канал B*
15	0x000F	USI_START	Прерывание по USI. Готовность к

			старту*
16	0x0010	USI_OVERFLOW	Прерывание по USI, Переполнение*
17	0x0011	EE_READY	Готовность EEPROM
18	0x0012	WDT_OVERFLOW	Переполнение охранного таймера

Имена прерываний и различных констант можно уточнить в заголовочном файле данного микроконтроллера (***C:\Program Files\Atmel\AVR Tools\AVR Toolchain\avr\include\avr\io2313.h***).

Имен векторов может быть несколько, но сделано это исключительно для совместимости в разных версиях компиляторов.

3. Занесением адреса вектора прерывания в счетчик команд фактически заканчивается выполнение аппаратных действий процессора на сигнал прерывания. На деле это означает, что следующей командой будет выполняться команда, содержащаяся в таблице векторов по адресу, закрепленному за этим событием (номером): безусловный переход к подпрограмме обработки данного прерывания;

4. Подпрограмма обработки прерывания (обработчик) выполняет все необходимые команды для свершившегося события и заканчивается командой выхода из подпрограммы обработчика прерывания ***RETI -*** *RETurn from Interrupt.* По этой команде восстанавливается общее разрешение на прерывания (флаг I регистра SREG устанавливается в 1), и верхние данные стека заносятся в счетчик команд. Если стек не нарушен, то там должен находиться адрес возврата из подпрограммы обработки прерывания. Поэтому эти действия эквивалентны передаче управления на команду, которая является следующей после той, на которой прерывание произошло. Выполнение этой команды занимает четыре такта. После выхода из прерывания процессор всегда выполняет одну команду основной программы, прежде чем обслужить любое отложенное прерывание.

Особенности конструирования обработчиков

Наступление прерывания в произвольный момент времени определяет, что любой обработчик не может принимать и передавать каких-либо параметров. Просто потому, что момент прерывания случаен и событие, вызвавшее прерывание, также неизвестно заранее.

Внутри обработчика прерывания можно программно разрешить или запретить общее прерывание. Для этого соответственно командой ***SEI*** (*Set Enable Interrupt)* установить 1 во флаг общего прерывания прерывания I регистра состояния SREG или программно запретить прерывания командой ***CLI*** (*CLear Interrupt)* - очистить, т.е. сбросить в 0 флаг I. На языке С эти команды соответственно записываются в виде макроса ***sei()*** или ***cli().*** Макросы просто подставят в исходный текст соответствующую ассемблерную команду. Но

при этом, разрешив прерывания, можно получить вложенные прерывания с непредсказуемыми последствиями или разрушение стека. Поэтому пользоваться этим следует с ясным пониманием последствий. В частности, понимая, что запрещая прерывания, закладываем запаздывание в обработке возможных прерываний, по крайней мере, до окончания выполняющегося обработчика.

Другие прерывания не потеряются. Все последующие прерывания, которые могут поступить во время выполнения обработчика и глобального запрета обработки прерываний, установят свои флаги. После разрешения общего прерывания они сгенерируют свои прерывания и будут обработаны. Обработка осуществляется по приоритету. Первым обрабатывается то прерывание, чей вектор меньше по адресу, ближе к началу памяти. Если за время запрета поступило несколько прерываний от одного источника, то обработка произойдет, естественно, только один раз. Очереди одинаковых прерываний не предусмотрены.

При возникновении прерывания устанавливается флаг, соответствующий этому событию (его бит устанавливается в 1). При запуске подпрограммы обработки прерывания он аппаратно сбрасывается 0 для фиксации последующих событий. ***Программно сбросить флаг в ноль можно, записав в него 1. Именно 1 !!!***

Если прерывание произойдет в спящем режиме микроконтроллера (sleep mode), то время реакции помимо обычных четырех тактов увеличивается еще на четыре такта, плюс время, заложенное во фьюз битах SUT1 и SUT0 (Start-Up Time).

Момент поступления прерывания неизвестен. Поэтому все, что использовалось в прерванном процессе должно быть обязательно сохранено после завершения обработчика. Для этого в начале обработчика последовательно сохраняют в стеке (команда PUSH) регистр состояния SREG и все те регистры, которые задействованы в обработчике. Перед завершением обработчика в обратном порядке их восстанавливают (команда POP).

Прерывания отвлекают процессор и блокируют другие прерывания. Поэтому обработчики должны быть максимально короткими и быстрыми. Все другие длинные процедуры должны выполняться в фоновом режиме. Поэтому при необходимости обработки массивов данных следует обработчиком сохранить банные в буфере, а обработку осуществлять за пределами обработчика.

Прерывания происходят в произвольный момент времени и зависят от автономно протекающих процессов, а результат их работы еще и от момента возникновения в основном коде. По этой причине некорректно организован-

ные прерывания могут носить характер сбоев и очень трудно поддаваться диагностированию. Поэтому все, что связано с прерываниями должно быть изучено глубоко и очень конкретно.

Таблица векторов прерываний

Таблица векторов прерываний (упорядоченный по приоритету список векторов прерываний) располагается с адреса 0x00 области памяти программ. Он не стирается при пропадании питания.

Адресуемая строка в программной области представлена 2-байтным словом. В семействе ATtiny строка вектора прерывания занимает 2 байта, а в семействе Mega занимают 4 байта. С учетом этого для ATtiny2313 программный код может выглядеть следующим образом:

```
.include "tn2313def.inc"
.cseg                       ;Выбор сегмента кодов (FLASH)
.org 0                      ;Указатель текущего адреса PC

rjmp  Reset                 ; Внешний сброс
rjmp  INT_0                 ; Внешний запрос по входу INT0
rjmp   INT_1                ; Внешний запрос по входу INT1
rjmp  Timer1_capt1          ; Захват таймера/счетчика 1
rjmp  Timer1_compA          ; Совпадению таймера/счетчика 1 канал A
rjmp  Timer1_OVF1           ; Переполнение таймера/счетчика 1
rjmp  Timer0_OVF0           ; Переполнение таймера/счетчика 0
rjmp   UART_RX              ; USART, прием завершен
rjmp  UART_UDRE             ; USART, буфер данных пуст
rjmp   UART_TX              ; USART, передача завершена
rjmp   ANA_COMP             ; Аналоговый компаратор
rjmp  PCINT                 ; Изменение на любом из выводов*
rjmp  Timer1_compB          ; Совпадению T1, канал B*
rjmp  Timer0_compA          ; Совпадению T0, канал A*
rjmp  Timer0_compB          ; Совпадению T0, канал B*
rjmp  USI_START             ; USI. Готовность к старту*
rjmp  USI_OVERFLOW          ; USI, Переполнение*
rjmp  EE_READY              ; Готовность EEPROM
rjmp  WDT_ OVERFLOW         ; Переполнение сторожевого таймера
```

Прерывание Reset (Сброс), в отличие от всех остальных, нельзя запретить. Такие прерывания еще называют немаскируемымиNMI (Non-maskable Interrupts). Это прерывание вызывается пятью событиями:

- включение питания (Power-on Reset). Микроконтроллер находится в состоянии сброса, пока напряжение питания находится ниже порогового значения VPOT;

- внешний сброс (External Reset). Микроконтроллер находится в состоянии сброса, пока на вывод RESET подан низкий уровень;

- переполнение сторожевого таймера (Watchdog Reset). Микроконтроллер сбрасывается через некоторый период времени, заданный в сторожевом таймере;

- понижение напряжения (Brown-Out Reset). Микроконтроллер сбрасывается, когда напряжение питания VCC опускается ниже порогового значения VBOT и когда Brown-Out детектор включен;

- JTAG сброс. Микроконтроллер находится в состоянии сброса, пока в однобитном регистре Reset Register находится значение логической единицы. Используется при программировании.

Если используются не все прерывания, то в таблице векторов вместо команд относительного перехода ***rjmp METKA*** для всех неиспользуемых прерываний записывается команда выхода из обработчика ***reti***. В этом случае создаются так называемые заглушки: обращение к таблице векторов происходит, а там сразу выполняется выход из обработчика, что обеспечивает реализацию всего механизма обработки прерывания, но по факту ничего не происходит (выполняется пустой обработчик).

Более компактная запись может получиться при использовании псевдокоманды указателя текущего адреса ***ORG***. С его помощью задается адрес в таблице векторов для используемого прерывания и определяется начало адресов кодов основной программы:

```
.cseg                       ; Выбор сегмента кодов (FLASH)
.org 0                      ; Указатель текущего адреса PC

rjmp    Reset               ; Внешний сброс

.org    2                   ;Указание адреса INT_1
;Альтернативный вариант указания адреса
; .org   INT1addr            Адрес внешнего INT_1
; В файле описаний:  .equ  INT1addr=$002
rjmp    INT_1               ; Внешний запрос по входу INT1
. . .
.org    INT_VECTORS_SIZE    ; Конец таблицы прерываний
                            ;Для ATtuny2313 = 0x0013

INT_1:                      ; Обработчик INT_1
        . . .
```

```
        . . .
        reti

RESET:  ldi   R16,RAMEND  ;Инициализация стека =0xDF
        out   SPL,R16
        . . .

; инициализация внешнего прерывания INT1 по переднему фронту
    ldi   R17, (1<<ISC011)|(1<<ISC10)   ; Код 0x0C
    out   MCUCR, R17
    ldi   R16, (1<<INTF1)   ; Очистка флагов внеш. прерываний
    out    EIFR, R16
    ldi   R16, 1<<INT1      ; разрешение внеш. прер.INT1
    out    GIMSK, R16       ; Код 0x80

    sei                     ; Глобальное разрешение прерыв.


forever: nop                ; Пустая команда (no operation)
        rjmp      forever   ; Бесконечный цикл
```

Легко заметить, что обработчики прерываний располагаются за таблицей векторов. В самом низу размещается секция инициализации и основное тело программы.

В исходном состоянии МКК (после RESET) все прерывания запрещены (бит I регистра SREG равен 0) - глобальный запрет. Кроме того, запрещены все прерывания на локальном уровне - сброшены в 0 все биты маски каждого прерывания индивидуально. Разрешение прерываний осуществляется индивидуально. Такие разрешения на глобальном и локальном уровнях следует делать программно. Пример такого разрешения для INT1 представлен в предыдущем варианте программы.

Организация прерываний на языке С

Для использования прерываний необходимо подключить библиотеку прерываний с помощью макросов, в которых описаны порты и все векторы прерываний:

```
#include <avr/io.h>
#include <avr/interrupt.h>
```

Процедура обработки прерывания не может ничего принимать в качестве аргумента, а также не может ничего возвращать. Это связано с тем, что все прерывания в AVR асинхронные, а источники и приемники данных не известны. Кроме того, это сокращает время обработки прерывания.

Процедуры обработчиков прерываний в AVR Studio (AVR GCC), объявляются перед основной программой с использованием макроопределения ***ISR*** (*Interrupt Service Routine* - обработчик прерываний):

```
ISR (<имя вектора прерывания_vect>)
{
                        /*код обработчика прерывания*/
}
```

Каждому прерыванию соответствует вектор прерывания. Например, обработчик прерывания по совпадению значения таймера со значением регистра OCR1A будет иметь следующий вид:

```
ISR (TIMER1_COMPA_vect)
{
                /*код обработчика*/
}
```

Если используется несколько обработчиков с одним кодом, то можно сослаться на одну и ту же процедуру обработки прерывания:

```
ISR(INT0_vect)
{
                // Обработчик
}
ISR(INT1_vect, ISR_ALIASOF(INT0_vect));
```

Таблица векторов строится компилятором самостоятельно. Неиспользуемые векторы часто заполняются по адресу вектора RESET. Поэтому случайные вызовы несуществующего прерывания приводит к сбросу МКК.

Большинство Си-компиляторов самостоятельно вычисляют занятые в обработчике регистры и перед началом и завершением выполнения тела процедуры обработки прерывания вставляют дополнительные "подпрограммы сохранения/восстановления регистров” общего назначения (РОН) и регистра состояния (SREG).

Программно управление глобальным прерыванием осуществляется с помощью макросов ***sei()*** - разрешить и ***cli()*** - запретить. Локальные прерыва-

ния управляются, как и на ассемблере, с помощью записи битов в регистры маски прерываний соответствующих источников.

Программная модель контроллера для управления прерываниями

Разрешение/запрещение прерываний в контроллере осуществляется глобально путем установки в 1/0 флага общего разрешения прерываний - бит ***I*** - *Interrupt* Регистра состояния SREG (рис.4.1). Манипуляции выполняется командами sei/cli, а также локальным (индивидуальным) разрешением/запрещением путем установки 1/0 в бит маски соответствующего источника прерываний.

Кроме того, сброс конкретного локального и глобального разрешений осуществляется аппаратно при вызове конкретного прерывания, а затем аппаратно разрешается глобальное прерывание при выходе из обработчика.

SREG=$3F Регистр состояния							
7	6	5	4	3	2	1	0
I	T	H	S	V	N	Z	C
Общее разрешение прерываний	Хранение копируемого бита	Половинный перенос	Флаг знака	Переполнение дополнительного кода	Отрицательное значение	Флаг нуля	Флаг переноса

Рис.4.1. Регистр состояния ***SREG*** - *Status Register*

Для управления прерываниями таймеров используются регистры:

- регистр маски прерываний от счетчиков TIMSK и
- регистр флагов прерываний от счетчиков TIFR (рис.4.2).

TIMSK=$39 Регистр МАСКИ прерываний от счетчика *Timer/Counter Interrupt Mask Register*							
7	6	5	4	3	2	1	0
TOIE1	OCIE1A	OCIE1B	-	TICIE1	OCIE0B	TOIE0	OCIE0A
Разрешен прерыван по переполн T1	Разрешен прерыван по совпад A T1	Разрешен прерыван по совпад B T1		Разрешен прерыван по Захвату T1	Разрешен прерыван по совпад B T0	Разрешен прерыван по переполн T0	Разрешен прерыван по совпад A T0

TIFR=$38 Регистр ФЛАГОВ прерываний от счетчика *Timer/Counter Interrupt Flag Register*							
7	6	5	4	3	2	1	0
TOV1	OCF1A	OCF1B	-	TICF1	OCF0B	TOV0	OCF0A
Флаг прерыван по переполн T1 №5	Флаг прерыван по совпад A T1 №4	Флаг прерыван по совпад B T1 №12		Флаг прерыван по Захвату T1 №3	Флаг прерыван по совпад B T0 №14	Флаг прерыван по переполн T0 №6	Флаг прерыван по совпад A T0 №13

Рис.4.2. Регистры управления прерываниями счетчиков T0 и T1

Источниками прерываний от счетчиков могут быть:

- переполнение счетчика T0 и T1 (флаг ***TOV*** - *Timer/Counter Overflow Flag* и разрешение прерывания ***TOIE*** - *Timer/Counter Overflow Interrupt Enable);*

- совпадение кода счетчика с регистром A и B счетчиков T0 и T1 (флаг ***OCF*** - *Output Compare Flag* и разрешение прерывания ***OCIE*** - *Output Compare Interrupt Enable);*

- захват таймера счетчика T1 (флаг ***TICF*** - *Timer/Counter Input Compare Flag* и разрешение прерывания ***TICIE*** - *Timer/Counter Input Compare Interrupt Enable)*. Осуществляется это прерывание по сигналу внутреннего компаратора или внешнего сигнала счетчика ***ICP*** - *Input Capture Pin*.

Для управления внешними прерываниями используются регистры:

- общий регистр маски прерываний GIMSK и
- регистр флагов внешних прерываний EIFR (рис.4.3);
- регистр состояния микроконтроллера MCUCR (рис.4.4).

Источниками внешних прерываний могут быть:

- сигналы на входах внешних прерываний ***INT0*** и ***INT1*** (флаг ***INTF*** - *Interrupt Flag* и разрешение прерывания ***INT*** - *Interrupt)*. Тип активного сигнала на этих входах определяется битами регистра MCUCR (см. рис.4.4), комбинации которых приведены в таблице 4.2;

- сигналы изменения состояния выводов (флаг ***PCIF*** - *Pin Change Interrupt Flag* и разрешение прерывания ***PCIE*** - *Pin Change Interrupt Enable).*

GIMSK=$3B Общий регистр МАСКИ прерываний *General Interrupt Mask Register*							
7	6	5	4	3	2	1	0
INT1	INT0	PCIE					
Разрешение внешнего прерывания INT1	Разрешение внешнего прерывания INT0	Разрешение прерывания по изменению выводов 0					

EIFR=$3A Регистр ФЛАГОВ внешних прерываний *External Interrupt Flag Register*							
7	6	5	4	3	2	1	0
INTF1	INTF0	PCIF					
Флаг внешнего прерывания INT1	Флаг внешнего прерывания INT0	Флаг прерывания по изменению выводов					

Рис.4.3. Регистры управление внешними прерываниями

MCUCR=$35 Общий регистр управления микроконтроллера *MCU Control Register*							
7	6	5	4	3	2	1	0
-	-	-	-	ISC11	ISC10	ISC01	ISC00
				Условия генерации внешнего прерывания INT1		Условия генерации внешнего прерывания INT0	

Рис.4.4. Общий регистр управления микроконтроллером

Таблица 4.2

Типы активных сигналов для генерации внешних прерываний

Бит	**Описание**		
ISC01:ISC00 ISC11:ISC10	Определяют условие генерации внешних прерываний INT0 и INT1 следующим образом:		
	ISC*n*1	**ISC*n*0**	**Условие**
	0	0	По НИЗКОМУ уровню на выводе INT*n*
	0	1	При любом изменении сигнала на выводе INT*n*
	1	0	По спадающему фронту сигнала на выводе INT*n*
	1	1	По нарастающему фронту сигнала на выводе INT*n*

ISC - *Interrupt Sense Control* - управление чувствительностью.

Изменение сигналов для формирования прерываний фиксируется на входах, совпадающих с портом B микроконтроллера. Эти сигналы могут

маскироваться с помощью Регистра маски прерываний по изменению состояний выводов PCMSK (рис.4.5), и не формировать прерывания.

PCMSK=$20 Регистр МАСКИ прерываний по изменению состояний выводов *Pin Change Mask Register*							
7	6	5	4	3	2	1	0
PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
Разрешение прерывания по изменению сост.	Разрешение прерывания по изменению сост.	Разрешение прерывания по изменению сост.	Разрешение прерывания по изменению сост.	Разрешение прерывания по изменению сост.	Разрешение прерывания по изменению сост.	Разрешение прерывания по изменению сост.	Разрешение прерывания по изменению сост.

Рис.4.5. Регистр маски прерываний по изменению состояний выводов

При наличии единицы в разряде PCINTx (*Pin Change Interrupt)* изменение состояния на входе для этого бита вызовет прерывание.

Внешние прерывания генерируются даже при конфигурировании соответствующих выводов на выход. Это дает возможность генерировать прерывания программно. Выводы микроконтроллера представлены на рис.4.6.

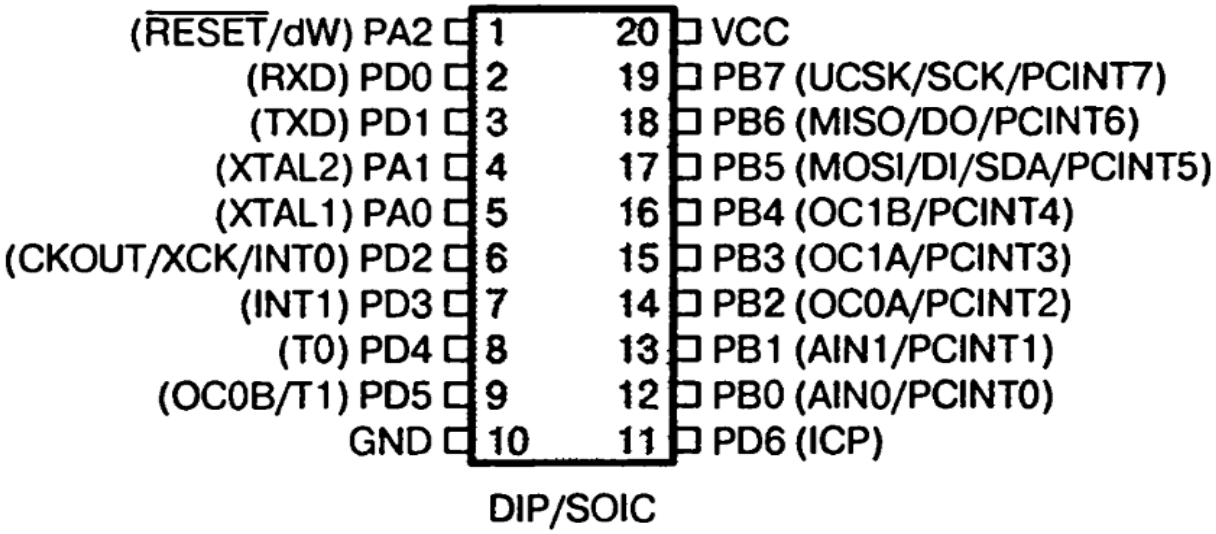

Рис.4.6. Выводы микроконтроллера ATtiny2313

Задание 4.2. Использование внешних прерываний

Задание 4.2.1. Зажигание и гашение светодиода кнопкой

Нажатием кнопки необходимо зажигать, а следующим нажатием гасить светодиод. Одиночным нажатием считается нажатие кнопки с последующим отпусканием. Для фиксации нажатия кнопки использовать прерывание INT1. Оно должно изменить флаг, по значениям которого должен включаться и вы-

ключаться светодиод. Флаг должен быть задан глобально. Разряд для управления светодиодом выбрать самостоятельно.

Проанализировать представленную программу. При отсутствии замечаний получить код для программирования микроконтроллера и проверить его работу в отладчике. Смоделировать работу кнопки в программе Proteus.

Если схема работает неправильно, то доработать так, чтобы зажигание и погасание светодиода происходило по заднему фронту, формируемому кнопкой.

Вариант программы:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

int num = 1;                          // Глобальная переменная
ISR(INT1_vect){                       //Обработчик INT1
if(num == 1)
        num = 0;
else
        num = 1;
}

int main (void)                       //Основная процедура
{
DDRB = 0xFF;                          // Порт B - выход
MCUCR = (0<<ISC11) | (1<<ISC10);      //Настройка на любое изменение
GIMSK |= (1<<INT1);                   // Разрешение локальное,  INT1

sei();                                // Разрешение прерыв. глобальное

    while(1){
if(num == 1)
        PORTB |= (1<<PB0);            // Включаем PB0
else
        PORTB &= ~(1<<PB0);           // Выключаем PB0

_delay_ms(100);             // Ждём 100мс
    }

}
```

В отчете представить:
1. Алгоритм и программу решения задачи;
2. Предложения по усовершенствованию схемы;
3. Электрическую схему в Proteus.

Задание 4.2.2. Зажигание и гашение светодиода разными кнопками

Разработать и отладить программу, которая позволяет включать светодиод одной кнопкой, а гасить - другой. Для подключения кнопок использовать внешние прерывания INT0, INT1. Работу программы проверить в программе Proteus.

В отчете представить:
1. Алгоритм и программу решения задачи;
2. Предложения по усовершенствованию схемы;
3. Электрическую схему в Proteus (доработать схему из задания 4.2.1).

Задание 4.3. Использование прерываний счетчиков

Задание 4.3.1. Прерывание счетчика по совпадению

Написать программу, обеспечивающую мигание светодиодом с частотой 1Гц с использованием прерываний счетчика по совпадению регистра A. Режим счетчика - Normal. Собрать схему в Proteus и убедиться в ее работоспособности. Тактовая частота процессора 4 МГц.

Если коэффициент предделителя выбрать 64, то частота тактирования счетчика TCNT1 вычисляется: 4МГц/64=62,5КГц, что соответствует периоду 16 мкс.

Для периода прерывания по совпадению 500 мс таймер получит значение: 500мс/16мкс=31250, или 0x7A12. Это число вполне допустимо для 16-битного регистра.

Отладить схему. Увеличить период мигания до 1с. Замерить период с помощью осциллографа и сравнить с расчетными данными AVR Studio. Проверить работу программы с обработчиком:

Первичный вариант программы:

```
#include <avr/io.h>
#include <avr/interrupt.h>
```

```
ISR (TIMER1_COMPA_vect);     //Обработчик прерывания
{
 TCNT1=0;                    //Обнуление счетчика

if(PORTB &(1<<PB0))
   PORTB&=~(1<<PB0);
else
   PORTB|=(1<<PB0);
}

int main(void)                          //Основная программа
{
 DDRB=0xFF;
 PORTB=0;

 OCR1A=0x7A12;               //Код совпадения в регистре А
 TIMSK1=(1<<OCIE1A);   //Разрешить по совпадению
 TCCR1B=(1<<CS11)|(1<<CS10)        ;Предделитель: К=64

sei();                 //Глобальное разрешение прерываний

while(1);
}
```

В отчете представить:

1. Схему подключения светодиода;
2. Расчетные соотношения для периода 1 сек;
3. Схему алгоритма программы;
4. Откорректированный текст программы с комментариями.

Задание 4.3.2. Генерация ШИМ с помощью таймера

Разработать и отладить программу и схему, обеспечивающих плавное изменение яркости горения светодиода. Использовать режим широтно-импульсной модуляции (ШИМ) Fast PWM.

Проверить работу программы в отладчике и программе Proteus.

Учесть, что ШИМ характеризуется коэффициентом заполнения D (величина обратная D называется скважностью):

$$D = \frac{t}{T}$$

В выражении t - время заполнения в границе периода T.

Действующее напряжение (или среднее) U_{CP} сигнала прямо пропорционально коэффициенту заполнения и амплитудному значению U_{AM}

$$U_{CP}=D*U_{AM}.$$

Если такой сигнал подать на фильтр низких частот, то получится аналоговый сигнал, величина которого пропорциональна величине заполнения. Это простой способ получения аналогового сигнала, простейший ЦАП.

Вариант программы:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>

volatile uint8_t pwm_value=0,dn_count=0;

ISR (TIMER1_COMPA_vect)
{
 TCNT1=0;

if(dn_count)              //плавно меняем яркость диода, по шагу за раз
pwm_value--;
else
   pwm_value++;

if(pwm_value==0)          //проверка границ, переключение
                          //разгорание/затухание
dn_count=0;
if(pwm_value==0xFF)
dn_count=1;

 OCR0A=pwm_value;         //устанавливаем новый коэфф. заполнения
}

void main(void)
{
 DDRD=0xFF;               //настройка порта на выход
 PORTD=0;

 OCR1A=0x7A12;            //конст., определяющая частоту прерываний
 TIMSK1=(1<<OCIE1A);      //разрешаем прерыв. по совпадению канала A
```

```
TCCR1B=(1<<CS11)|(1<<CS10);//запускаем таймер 1

TCCR0A=(1<<COM0A1)|(1<<WGM01)|(1<<WGM00);//таймер 0 будет
                        //генерировать ШИМ
OCR0A=128;              //начальное значение ШИМ
TCCR0B=(1<<CS00);       //запускаем таймер 0

sei();                  //разрешаем прерывания

while(1);               //дальше процесс идет на прерываниях и
                        //аппаратном ШИМе
}
```

В этой программе используется макрос ***stdint.h.*** В нем объявлены некоторые типы данных с явно выраженной разрядностью:

uint8_t — unsigned 8-bit integer type;
uint16_t — unsigned 16-bit integer type;
uint32_t — unsigned 32-bit integer type;
int8_t — signed 8-bit integer type.

Эти описания повышают читаемость кода, гарантируют переносимость кода для других микроконтроллеров, иногда сокращают написание при объявлении переменных. Но применение этого макрорасширения не обязательно.

Модификатор ***volatile*** означает, что компилятору запрещается оптимизировать данную переменную. Это делает работу компилятора предсказуемой. Для предотвращения работы оптимизатора по отношению ко всей программе можно поставить уровень оптимизации -0s в свойствах проекта.

В отчете представить:

1. Схему подключения светодиода (если схема не совпадает с предыдущими схемами);
2. Расчетные соотношения для периода 1 сек;
3. Откорректированный текст программы с комментариями.
4. Расчеты для определения времени полного загорания и погасания.

Вопросы для самоконтроля

1. Какие адреса занимают порты ввода-вывода управления прерываниями?
2. Назовите основные источники прерываний для МКК.
3. Какие прерывания имеются для счетчиков в МКК ATtiny2313?
4. Назначение маски прерываний счетчиков?
5. Какими командами разрешаются и запрещаются прерывания?
6. С какой целью запрещаются прерывания?

7. Что составляет контекст процесса?
8. Как сохранить контекст? С какой целью его сохраняют?
9. Предложите алгоритм формирования длительной задержки с помощью таймеров?
10. Почему таймеры самостоятельно не могут организовать минутной задержки?
11. Какое макрорасширение используется для работы с прерываниями на языке С?
12. Как написать обработчик на языке С?
13. Что представляет собой реакция микроконтроллера на сигнал прерывания?
14. Каким образом процессор завершает выполнение обработчика? Что при этом выполняется?
15. Каким образом с помощью таймера организовать задержку большой длительности (например, минуты)?
16. Какие настройки выполняются для организации прерываний?
17. Какую роль выполняют флаги прерываний?
18. Как определяется приоритет прерываний микроконтроллера?
19. Как можно уточнить зарезервированные имена обработчиков;
20. Что нужно сделать, чтобы обработчик не выполнялся?
21. Сколько ячеек памяти занимает таблица векторов?
22. Как долго существует флаг прерывания?
23. Какова реакция контроллера на несколько прерываний одного объекта до их обработки?
24. При каких условиях могут образоваться вложенные прерывания?
25. Как можно организовать программные прерывания?
26. С какой целью предусмотрен захват таймера?
27. Как осуществляется инициализация таблицы векторов прерываний счетчиков?
28. Настройку каких регистров необходимо произвести для разрешения прерываний от таймер/счетчиков?
29. Можно ли отменить очередь прерываний?
30. Исходя из чего определяется очередность обработки прерываний при возникновении нескольких прерываний одновременно?
31. Какие прерывания могут быть утрачены после разрешения появления?
32. Какой регистр предназначен для глобального разрешения/запрещения прерывания?
33. Какие регистры используются для управления прерываниями счетчиков?
34. Какие регистры используются для управления внешними прерываниями?
35. Чем различаются между собой внешние прерывания?