Смирнов В.И.

ПРОЕКТИРОВАНИЕ и схемотехническое моделирование микропроцессорных устройств

2013

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

В. И. Смирнов

ПРОЕКТИРОВАНИЕ И СХЕМОТЕХНИЧЕСКОЕ МОДЕЛИРОВАНИЕ МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ

Учебное пособие

для студентов, обучающихся по направлению 211000 –
«Конструирование и технология электронных средств»

Ульяновск
УлГТУ
2013

УДК 004.31-22.53 (075)
ББК 32.973.26-04 я73
С 50

Рецензенты: Ульяновский филиал Института радиотехники и электроники им. В.А. Котельникова РАН (директор д-р техн. наук В. А. Сергеев);
доцент кафедры «Авиационная техника» Ульяновского высшего авиационного училища гражданской авиации (института),
канд. техн. наук А. В. Ефимов

Утверждено редакционно-издательским советом университета
в качестве учебного пособия

Смирнов, В. И.

С 50 Проектирование и схемотехническое моделирование микропроцессорных устройств : учебное пособие для студентов, обучающихся по направлению 211000 – «Конструирование и технология электронных средств» / В. И. Смирнов. – Ульяновск : УлГТУ, 2013. – 119 с.

ISBN 978-5-9795-1164-1

Изложены вопросы проектирования микропроцессорных устройств, включая разработку программного обеспечения для микроконтроллеров. Рассмотрение архитектуры микроконтроллеров и особенностей функционирования периферийных устройств ведется на примере популярного микроконтроллера ATmega128 семейства AVR. Для формирования практических навыков работы с микроконтроллерами активно используется среда схемотехнического моделирования Proteus. Пособие снабжено большим количеством примеров, в которых с помощью микроконтроллера решаются типовые задачи, присущие измерительным и управляющим микропроцессорным устройствам.

Пособие предназначено для магистрантов, обучающихся по направлению 211000.68 – «Конструирование и технология электронных средств» и профилю подготовки – «Элементы и устройства электронно-вычислительных средств».

УДК 004.31-22.53 (075)
ББК 32.973.26-04 я73

ISBN 978-5-9795-1164-1

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Микроконтроллеры повсеместно вошли в нашу жизнь. Их можно встретить практически в любом техническом изделии, в котором требуется решать задачи измерения, обработки информации и управления. Это может быть электробытовая техника, измерительные приборы или средства коммуникации, а также такие сложные объекты управления, как автомобили или самолеты. Круг решаемых микроконтроллерами задач очень широк, начиная от включения звуковой или световой сигнализации и заканчивая сложной математической обработкой и анализом информации, поступающей от различных датчиков, с последующим формированием управляющих воздействий на объект управления.

Промышленностью выпускается огромное количество различных типов микроконтроллеров, отличающихся по техническим характеристикам, архитектуре центрального процессорного модуля, наличию тех или иных периферийных устройств. Понять, что в них общего и в чем различие, бывает совсем непросто, поскольку какой-то общепринятой строгой классификации микроконтроллеров нет. Имеется ряд признаков, по которым их можно классифицировать. Среди этих признаков можно выделить следующие.

Разрядность шины данных. Различаются 8-, 16- и 32-разрядные микроконтроллеры. Имеются и 4-разрядные, но их область применения весьма ограничена. Естественно, что производительность 32-разрядных микроконтроллеров выше, чем у 8-разрядных и в последние годы их производство резко возросло. Но и 8-разрядные микроконтроллеры еще долго будут востребованы на рынке, поскольку для многих задач очень высокая производительность и не требуется. Кроме того, надо учитывать огромный опыт использования этих микроконтроллеров, богатые библиотеки прикладных программ и т.д., да и в плане первоначального обучения работы с микроконтроллерами они предпочтительнее 32-разрядных.

Организации памяти программ и памяти данных. Различают Принстонскую и Гарвардскую архитектуры. В Принстонской архитектуре память для хранения команд (инструкций) и данных является общей, что позволяет гибко перераспределять ресурсы памяти между командами и данными. В Гарвардской архитектуре память программ и память данных физически разделены и адресуются посредством разных шин. Это дает возможность процессорному ядру одновременно читать инструкции и выполнять доступ к памяти данных, что повышает быстродействие микроконтроллера.

Архитектура вычислительной системы. Различают CISC-архитектуру с комплексным набором команд (CISC-Complex Instruction Set Computing) и RISC-архитектуру с сокращенным набором команд (RISC-Reduced Instruction Set Computing). Особенностью CISC-архитектуры является то, что команды выполняются поочерёдно друг за другом и имеют разную длину и структуру. Выборка команды из памяти осуществляется побайтно и выполняется за несколько тактов. Основная же идея RISC-архитектуры заключается в замене

сложных команд однотипными простыми и выполнении их единым потоком на параллельном конвейере. Все команды имеют фиксированную длину и в идеале должны выполняться за один, а не за несколько, тактов, чем достигается повышенное быстродействие. В настоящее время в подавляющем большинстве микроконтроллеров используется RISC-архитектура.

Выполняемые функции. Согласно этому признаку микроконтроллеры делятся на универсальные и специализированные. Специализированные в основном предназначены для наиболее эффективного решения конкретных задач, например, управления электродвигателями, обеспечения беспроводного интерфейса или высокоскоростной обработки больших массивов данных. Это не означает, что универсальные микроконтроллеры не могут решать аналогичные задачи. Просто у них для этого обычно необходимы соответствующие внешние периферийные устройства, что усложняет проектирование и программирование таких устройств.

Построение центрального процессорного модуля – ядра микроконтроллера. Ядро определяет систему команд, шинный интерфейс, архитектуру памяти и т.д. Различают микроконтроллеры с ядрами AVR, PIC, MCS-51, ARM, 68HC, MSP-430, Z80 и т.д. На одном и том же ядре разными фирмами могут производиться различные семейства микроконтроллеров, у которых примерно одинаковый набор программных и периферийных функций. Одни и те же фирмы могут выпускать различные семейства микроконтроллеров на разных ядрах. Например, фирма Atmel производит 8- и 32-разрядные микроконтроллеры на ядрах AVR, MCS-51 и ARM.

В настоящее время количество фирм-производителей микроконтроллеров исчисляется десятками. Среди них можно отметить Analog Device, Atmel, Cypress, Infineon, Microchip Technology, Motorola, National Semiconductor, NEC, Philips, Renesas, Samsung, SiliconLabs, STMicroelectronics, Texas Instruments, Toshiba. Технические характеристики и набор периферии у них на любой вкус. Очень часто выбор наиболее подходящего микроконтроллера становится очень непростой задачей.

Для первоначального обучения в наибольшей степени подходят 8-разрядные микроконтроллеры. Как показывают всевозможные опросы специалистов, наиболее популярными среди этого класса микроконтроллеров являются семейства AVR фирмы Atmel и семейства PIC фирмы Microchip Technology. В настоящей книге объектом для первоначального изучения выбран микроконтроллер ATmega128. Микроконтроллеры семейства ATmega обладают хорошим набором периферийных устройств и, соответственно, широкими функциональными возможностями; по соотношению производительность/цена они являются одними из лидеров; они отлично адаптированы к языкам программирования высокого уровня (исходный код на языке С после компиляции получается очень компактным); для них имеются доступные средства отладки программного обеспечения.

Начальный этап разработки любого микропроцессорного устройства включает в себя проектирование электрической принципиальной схемы устройства, разработку программного обеспечения для микроконтроллера и его отладку. С учетом этого в первой главе представлены основные сведения о микроконтроллере ATmega128 и особенностях работы его основных периферийных устройств. Содержание главы основано на информации, представленной фирмой-производителем микроконтроллеров ATmega128 в его техническом описании [1]. В отличие от технического описания, ориентированного на специалистов, материал в ней адаптирован под начинающего разработчика микропроцессорных устройств. При желании получить более полную информацию о микроконтроллере следует обратиться к его техническому описанию.

Во второй главе приводятся краткие сведения о языке С, поскольку в настоящее время в большинстве случаев программное обеспечение для микроконтроллеров разрабатывается не на ассемблере, а на языках высокого уровня. Приведено описание интегрированной среды программирования ICCAVR, служащей для разработки управляющей программы и ее компиляции в исполняемый микроконтроллером файл.

Третья глава посвящена рассмотрению системы схемотехнического моделирования Proteus, которая позволяет осуществлять анализ работы проектируемых микропроцессорных устройств. Возможность проверить корректность функционирования микроконтроллера без использования реального макетирования позволяет в кратчайшие сроки получить навыки проектирования и программирования микропроцессорных устройств. Для формирования таких навыков в четвертой главе приведен ряд практических примеров взаимодействия микроконтроллера с внешними периферийными устройствами, такими как персональный компьютер, жидкокристаллический индикатор и аналого-цифровой преобразователь.

1. МИКРОКОНТРОЛЛЕРЫ СЕМЕЙСТВА AVR

1.1. Общая характеристика микроконтроллеров семейства AVR

Восьмиразрядные микроконтроллеры семейства AVR по праву считаются одним из самых интересных направлений, активно развиваемых корпорацией Atmel. Они представляют собой мощный инструмент для создания современных высокопроизводительных и экономичных многоцелевых контроллеров. Что отличает эти микроконтроллеры от других 8-разрядных микроконтроллеров? AVR-микроконтроллеры имеют Гарвардскую архитектуру, согласно которой память программ и память данных физически и логически разделены, адресуются посредством различных шин, используя для этого разные управляющие сигналы. Они относятся к RISC микроконтроллерам, в которых объединена достаточно мощная система команд с 32 регистрами общего назначения и одноуровневым конвейером выборки команд из памяти программ. Наличие конвейера позволяет объединить в одном цикле две операции – выполнение текущей команды и выборку следующей.

Важной отличительной чертой архитектуры AVR является регистровый файл, включающий в себя 32 регистра общего назначения, которые напрямую связаны с арифметико-логическим устройством, что позволяет выполнять обращение к двум независимым регистрам и возвращать результат одной командой, выполняемой за один цикл. Выполняя команды за один тактовый цикл, микроконтроллер обеспечивает производительность на уровне 1 миллиона операций в секунду на 1 МГц тактовой частоты.

Разработка прикладных программ возможна как на языках высокого уровня, так и на ассемблере. Все микроконтроллеры AVR имеют Flash-память программ, которая может быть загружена как с помощью обычного программатора, так и с помощью последовательного периферийного интерфейса SPI, в том числе непосредственно на рабочей плате (внутрисистемное программирование). Поскольку все команды имеют 16-разрядный или 32-разрядный форматы, то Flash-память имеет 16-разрядную организацию. Некоторые из микроконтроллеров имеют возможность самопрограммирования. Это означает, что микроконтроллер способен самостоятельно, без какого-либо внешнего программатора, изменять содержимое памяти программ. Иначе говоря, новые AVR могут менять алгоритмы своего функционирования и программы, заложенные в них, после чего могут продолжать работу уже по измененному алгоритму или новой программе.

В качестве памяти данных используется статическое ОЗУ (SRAM). Кроме этого все AVR-микроконтроллеры имеют блок энергонезависимой электрически стираемой памяти данных EEPROM. Этот тип памяти, доступный программе микроконтроллера непосредственно в ходе ее выполнения, удобен для хранения промежуточных данных, различных констант, таблиц перекодировок, калибровочных коэффициентов и т.п. EEPROM также может быть загружена извне как через SPI интерфейс, так и с помощью обычного программатора. Число

циклов перезаписи – не менее 100 000. Программируемые биты секретности (Lock-биты) позволяют защитить память программ и энергонезависимую память данных EEPROM от несанкционированного считывания. Кроме битов секретности имеются программируемые конфигурационные биты (Fuse-биты), позволяющие настраивать микроконтроллер на различные режимы работы.

AVR имеют в своем составе типичные для 8-разрядных микроконтроллеров периферийные узлы: двунаправленные порты ввода/вывода, таймеры/счетчики, средства обмена данными с другими микроконтроллерами в мультипроцессорных системах. Выходные драйверы портов обеспечивают токовую нагрузочную способность 20 мА на каждую линию порта (втекающий ток) при максимальном значении 40 мА, что позволяет непосредственно подключать к микроконтроллеру светодиоды и биполярные транзисторы. Некоторые представители данного семейства имеют интегрированные на кристалле АЦП, аналоговый компаратор, генератор импульсов с широтно-импульсной модуляцией PWM, счетчик реального времени с отдельным генератором и ряд других устройств.

Для предотвращения сбоев в работе и последующего «зависания» имеется программируемый сторожевой таймер WDT, который в процессе функционирования микроконтроллера периодически сбрасывается контролируемой системой. Если сброса не произошло в течение некоторого интервала времени, то происходит принудительная перезагрузка системы. Сторожевой таймер WDT тактируется RC-генератором с предделителем входной частоты и программируемым коэффициентом деления, что позволяет подстраивать временной интервал переполнения таймера и сброса микроконтроллера. Сторожевой таймер может быть отключен программным образом во время работы микроконтроллера как в активном режиме, так и в любом из режимов пониженного энергопотребления. В последнем случае это приводит к значительному снижению потребляемого тока. Все микроконтроллеры имеют несколько режимов энергосбережения, в которые они могут перейти программно, а выйти из них в активный режим – по внешним сигналам.

В новых микроконтроллерах значительно переработан блок задающего генератора. Добавлена возможность работы от низкочастотного кварцевого резонатора (с частотой до 32 768 Гц), внешней частотозадающей RC-цепи или встроенного калиброванного RC-генератора. В большинстве представителей семейства AVR имеются такие периферийные узлы, как аппаратный умножитель (команда умножения выполняется за 2 такта); двухпроводный периферийный интерфейс TWI; интерфейс JTAG для внутрисхемной отладки и программирования. В некоторых микроконтроллерах появились дифференциальные АЦП с программируемым коэффициентом усиления на входе, контроллер символьного жидкокристаллического индикатора LCD и модуль контроля напряжения питания BOD (Brown out Detector). Появились также интерфейсы LIN и CAN, высокоточный ШИМ для приложений управления электродвигателями. Существенный прогресс в производительности был достигнут в 8-разрядном

микроконтроллере XMEGA, который содержит несколько каналов прямого доступа к памяти (DMA), быстродействующие 12-разрядные АЦП и ЦАП, систему событий, криптомодули и ряд других новых периферийных узлов.

Типичным представителем семейства AVR является микроконтроллер ATmega128, обладающий высокой производительностью, широким набором периферийных устройств, большой энергонезависимой памятью программ и данных, несколькими режимами снижения энергопотребления, возможностью внутрисистемного программирования. В то же время у него отличное соотношение производительность/стоимость. Все это определило его исключительно высокую популярность среди разработчиков микропроцессорных устройств. Поэтому в дальнейшем все особенности функционирования микроконтроллеров семейства AVR будут рассматриваться на примере ATmega128.

1.2. Отличительные особенности микроконтроллера ATmega128

ATmega128 является одним из наиболее производительных микроконтроллеров семейства AVR. Он выпускается в двух вариантах: ATmega128 – для обычного диапазона питающих напряжений и ATmega128L – для низковольтного питания. Из его отличительных особенностей можно выделить следующие.

Прогрессивная RISC архитектура:

- 133 высокопроизводительные команды, большинство из которых выполняется за один тактовый цикл;
- 32 восьмиразрядных рабочих регистра общего назначения и дополнительно к ним – регистры управления периферией;
- высокая производительность, приближающаяся к 16 млн операций в секунду (при тактовой частоте 16 МГц);
- встроенный аппаратный умножитель, выполняющий умножение за два тактовых цикла.

Большой объем памяти программ и данных:

- 128 Кбайт перепрограммируемой Flash-памяти программ;
- 4 Кбайта встроенной оперативной памяти данных SRAM;
- 4 Кбайта электрически стираемой перепрограммируемой памяти EEPROM;
- до 64 Кбайтов дополнительной внешней памяти данных.

Большой набор встроенных периферийных устройств:

- два 8-разрядных таймера/счетчика с отдельным предварительным делителем;
- два 16-разрядных таймера/счетчика с расширенными возможностями;
- счетчик реального времени с отдельным генератором;
- два 8-разрядных канала с широтно-импульсной модуляцией (PWM);
- шесть каналов PWM с возможностью программирования разрешения от 1 до 16 разрядов;

• 8-канальный 10-разрядный аналого-цифровой преобразователь с расширенными функциями (7 дифференциальных каналов, из них 2 канала с программируемым коэффициентом усиления);
• встроенный аналоговый компаратор;
• программируемый сторожевой таймер WDT с отдельным встроенным генератором;
• 53 программируемые линии ввода/вывода данных, объединенные в 7 портов.

Большой выбор последовательных интерфейсов:

• JTAG-интерфейс, позволяющий осуществлять загрузку и отладку программного обеспечения, а также тестирование периферии;
• SPI интерфейс для внутрисистемного программирования;
• сдвоенный программируемый последовательный приемопередатчик USART для синхронного и асинхронного приема/передачи данных;
• двухпроводный последовательный интерфейс TWI.

Дополнительные возможности и функции микроконтроллера:

• формирование сигнала сброса при подаче питания;
• наличие программируемой схемы сброса при снижении напряжения питания;
• наличие встроенного калиброванного RC-генератора;
• возможность программной установки тактовой частоты;
• наличие внутренних и внешних источников прерываний;
• шесть режимов пониженного энергопотребления: холостой ход (Idle), экономичный (Power-save), выключение (Power-down), дежурный (Standby), расширенный дежурный (Extended Standby), режим снижения шумов при работе АЦП (ADC Noise Reduction);
• возможность программирования битов секретности (Lock-биты), обеспечивающих защиту программ от несанкционированного доступа;
• возможность программирования конфигурационных битов (Fuse-биты), обеспечивающих настройку микроконтроллера на различные режимы работы;
• внутрисистемное программирование встроенной программой загрузки;
• возможность обеспечения совместимости с ATmega103 с помощью программирования специального конфигурационного бита.

Диапазон напряжений питания составляет 2,7 – 5,5 В (для ATmega128L) и 4,5 – 5,5 В (для ATmega128). Диапазон тактовых частот генератора 0 – 8 МГц (для ATmega128L) и 0 – 16 МГц (для ATmega128).

1.3. Структурная организация ATmega128

1.3.1. Особенности архитектуры и назначение выводов

Структурная схема микроконтроллеров ATmega128 представлена на рис. 1.1. В ней можно выделить арифметико-логическое устройство АЛУ, с которым непосредственно связан файл регистров быстрого доступа (регистровый файл). Регистровый файл содержит 32 восьмиразрядных рабочих регистра

общего назначения. За один тактовый цикл из регистрового файла выбираются два операнда, выполняется арифметическая или логическая операция и результат вновь возвращается в регистровый файл.

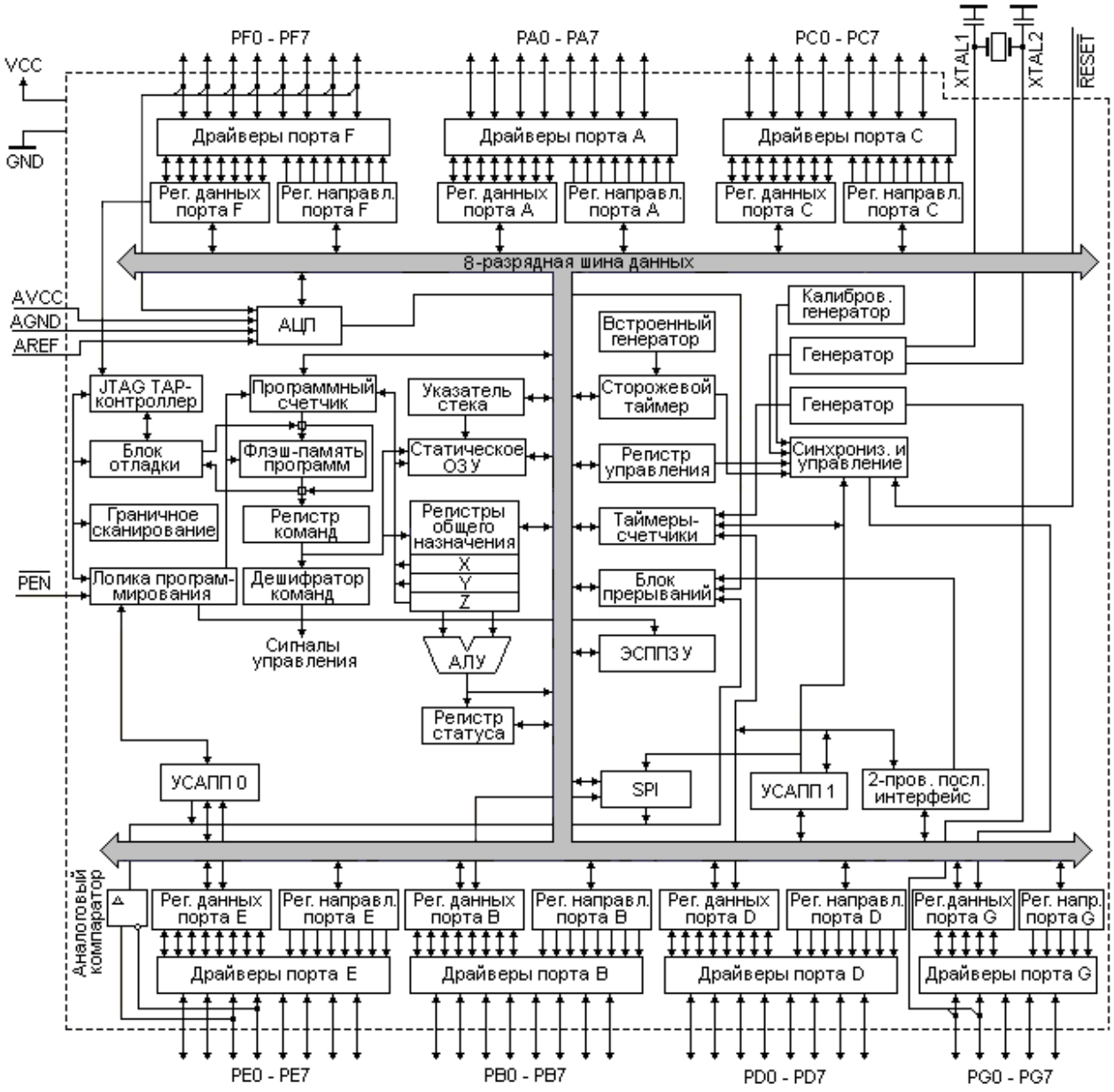

Рис. 1.1. Структурная схема микроконтроллера ATmega128

Шесть из 32 регистров могут быть использованы как три 16-разрядных регистра указателя косвенной адресации при обращении к памяти данных (X-, Y- и Z-регистры). Один из этих указателей адреса может также использоваться как указатель адреса для доступа к таблице преобразования во Flash-памяти программ. Данные 16-разрядные регистры называются X-регистр, Y-регистр и Z-регистр и описываются ниже.

К регистрам общего назначения регистрового файла можно обращаться непосредственно (регистры R0, R1,…, R31) либо использовать адресацию как к обычным ячейкам памяти данных. Адресное пространство регистрового файла занимает адреса внутреннего статического ОЗУ в диапазоне от $00 до $1F. К адресному пространству регистрового файла примыкают адреса памяти ввода/вывода (память I/O). Пространство памяти I/O содержит 64 адреса регистров, определяющих работу различных периферийных устройств микроконтроллера. К памяти I/O можно обращаться непосредственно или как к ячейкам памяти данных, соответствующим адресам $20…$5F. Кроме того, ATmega128 имеет пространство расширенной памяти I/O по адресам $60 – $FF в статическом ОЗУ.

АЛУ поддерживает арифметические и логические операции между регистрами, а также между константой и регистром. Кроме того, АЛУ поддерживает действия с одним регистром. После выполнения арифметической операции регистр статуса обновляется для отображения результата выполнения операции.

Очередная команда (инструкция) выбирается из Flash-памяти программ согласно адресу, установленному счетчиком команд. При работе с памятью программ используется одноуровневый конвейер. Это означает, что в то время, как одна команда выполняется, следующая команда выбирается из памяти программ. Такой прием позволяет выполнять большинство команд за один тактовый цикл. За малым исключением команды имеют формат одного 16-разрядного слова, в связи с чем каждый адрес памяти программ содержит одну 16-разрядную инструкцию. Все данные могут храниться в энергозависимом статическом ОЗУ (SRAM объемом 4 Кбайт) либо в энергонезависимой электрически стираемой перепрограммируемой памяти (EEPROM).

Для ветвления программы поддерживаются команды условных и безусловных переходов и вызовов процедур, позволяющих непосредственно адресоваться в пределах адресного пространства. В процессе обработки прерываний и вызовов подпрограмм адрес возврата сохраняется в стеке. Стек размещается в SRAM данных и, следовательно, размер стека ограничен только общим размером SRAM и уровнем ее использования. В любой программе перед выполнением процедур обработки прерываний или вызовом подпрограмм должна быть выполнена инициализация указателя стека SP. Указатель стека доступен на чтение и запись в пространстве адресов памяти I/O.

Важной составной частью любого микроконтроллера являются периферийные устройства. Для ATmega128 в состав периферийный устройств входят четыре таймера/счетчика, АЦП, аналоговый компаратор, сторожевой таймер. К периферийным устройствам относятся также средства последовательного интерфейса, такие как два универсальных синхронно-асинхронных приемопередатчика USART, последовательный периферийный интерфейс SPI, JTAG-интерфейс и двухпроводный последовательный интерфейс TWI.

Взаимодействие микроконтроллера с внешними устройствами осуществляется посредством 7 двунаправленных параллельных портов. Все порты кроме общего назначения имеют дополнительные функции. Например, линии порта F

обеспечивают подачу аналогового напряжения на вход встроенного АЦП, а две линии порта E позволяют подать напряжения на входы аналогового компаратора. Назначение выводов показано на рис. 1.2.

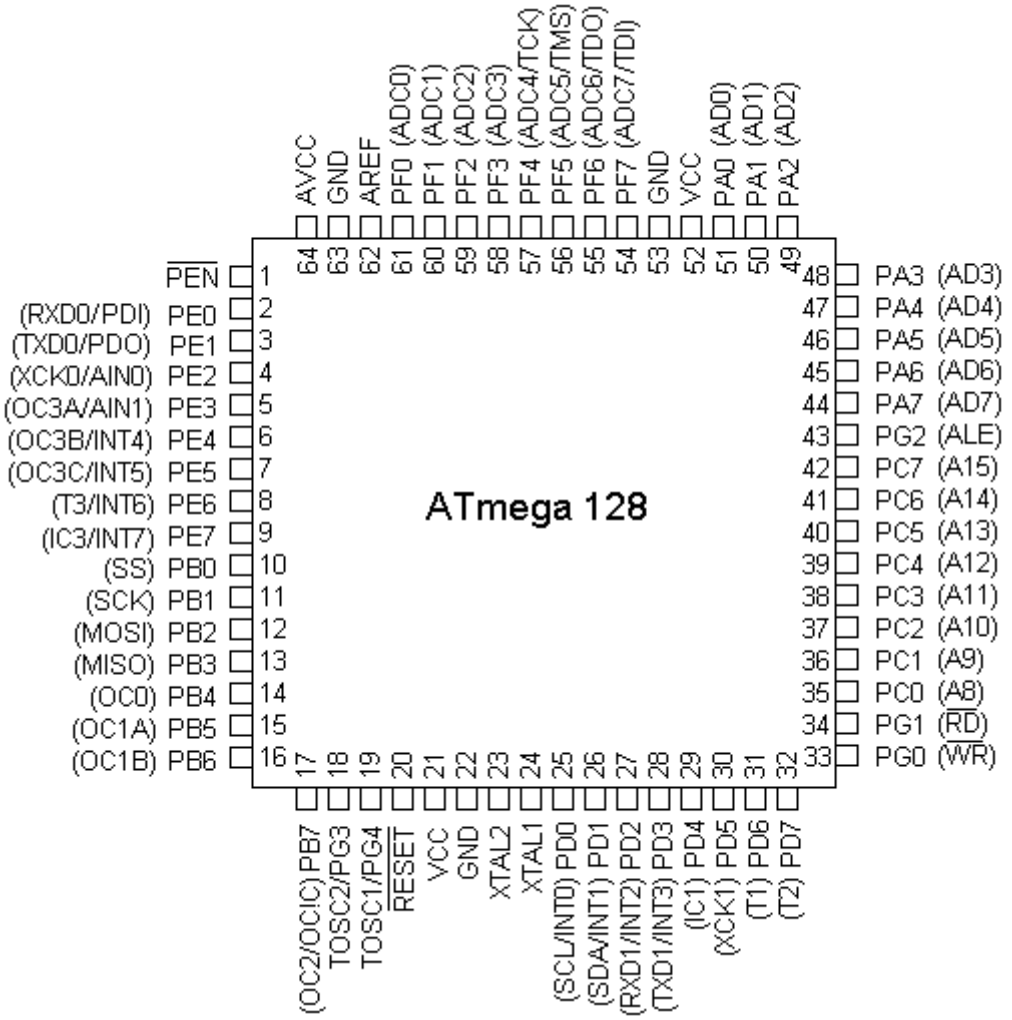

Рис. 1.2. Назначение выводов микроконтроллера ATmega128

Выводы микроконтроллера предназначены для ввода/вывода следующих сигналов:

VCC – вывод для подключения напряжения питания.

GND – вывод для подключения к общей шине.

RESET – вывод управляющего сигнала сброса (активный уровень – низкий). Минимальная длительность импульса сброса составляет 50 нс. Действие импульса меньшей продолжительности не гарантирует сброс.

XTAL1 и **XTAL2** – вход и выход инвертирующего усилителя, который в случае использования кварцевого или керамического резонаторов работает как встроенный генератор. При использовании внешнего генератора тактовой частоты выход генератора подключается к выводу XTAL1, а вывод XTAL2 должен оставаться свободным.

AVCC – вывод для напряжения питания АЦП. Он должен быть внешне связан с VCC, даже если АЦП не используется. При использовании АЦП этот вывод связан с VCC через фильтр низких частот.

AREF – вход подключения источника опорного напряжения для АЦП. На этот вывод для обеспечения работы АЦП подается напряжение в диапазоне между 0 и AVCC.

AGND – вывод для подключения к аналоговой «земле», если таковая имеется в устройстве. В ином случае вывод подсоединяется к общей шине.

PEN – вход разрешения программирования для режима последовательного программирования через интерфейс SPI. Если во время действия сброса при подаче питания на этот вход подать низкий уровень, то микроконтроллер переходит в режим последовательного программирования через SPI. В рабочем режиме PEN не выполняет никаких функций.

PA0…PA7 – 8-разрядный двунаправленный порт ввода/вывода A.

PB0…PB7 – 8-разрядный двунаправленный порт ввода/вывода B.

PC0…PC7 – 8-разрядный двунаправленный порт ввода/вывода C.

PD0…PD7 – 8-разрядный двунаправленный порт ввода/вывода D.

PE0…PE7 – 8-разрядный двунаправленный порт ввода/вывода E.

PF0…PF7 – 8-разрядный двунаправленный порт ввода/вывода F.

PG0…PG4 – 5-разрядный двунаправленный порт ввода/выводаG.

Как уже отмечалось, все выводы портов кроме общего назначения имеют альтернативные дополнительные функции. Эти дополнительные функции, отмеченные на рис. 1.2 в скобках, будут описаны ниже.

Следует отметить, что по количеству выводов и их расположению микроконтроллер ATmega128 практически полностью совпадает с ATmega103, который в настоящее время снят с производства. Это позволяет использовать его в режиме совместимости с ATmega103 с помощью программирования специального конфигурационного бита M103C в расширенном конфигурационном байте. При этом у ATmega128 теряется часть функций, а именно, не функционируют один из четырех таймеров/счетчиков, один USART, JTAG-интерфейс, порт G, уменьшено количество векторов прерываний и т. д.

1.3.2. Организация памяти

В соответствии с Гарвардской архитектурой память всех AVR-микроконтроллеров разделена на две области: память данных и память программ. Кроме того, ATmega128 содержит память ЭСППЗУ для энергонезависимого хранения данных.

Объем памяти программ ATmega128 составляет 128 Кбайт Flash-памяти, которая может быть запрограммирована как параллельно с помощью специальных программаторов, так и последовательно с помощью интерфейсов SPI или JTAG. Она разделена на два сектора: сектор прикладной программы и сектор программы начальной загрузки.

В микроконтроллере ATmega128 поддерживается две конфигурации памяти данных, соответствующие нормальному режиму (конфигурация A) и режиму совместимости с ATmega103 (конфигурация B). Объем внешней и внутренней (встроенной) памяти данных SRAM для конфигураций A и B представлен в таблице 1.1. Рис. 1.3 иллюстрирует организацию памяти программ и данных у ATmega128.

Таблица 1.1

Конфигурация памяти данных

Конфигурация	Встроенная SRAM данных	Внешняя SRAM данных
A – нормальный режим	4096 байт	До 64 Кбайт
B – режим совместимости с ATmega103	4000 байт	До 64 Кбайт

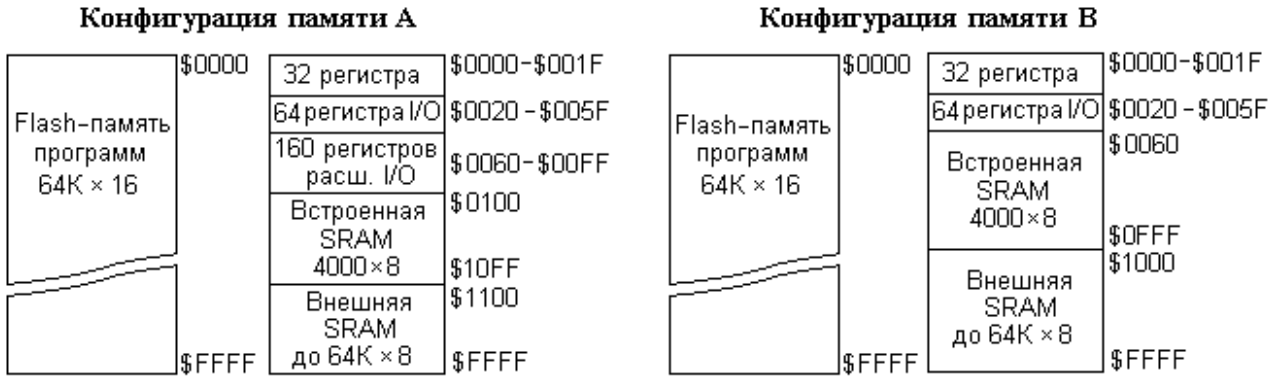

Рис. 1.3. Конфигурация памяти микроконтроллера ATmega128

В нормальном режиме первые 4352 ячейки памяти данных относятся к регистровому файлу, памяти ввода/вывода, расширенной памяти ввода/вывода и встроенной статической SRAM данных. В первых 32 ячейках расположен файл регистров, следующие 64 ячейки занимает стандартная память ввода/вывода, а за ними следуют 160 ячеек расширенной памяти ввода/вывода. Замыкают внутреннюю память данных 4096 ячеек внутренней статической SRAM данных.

В режиме совместимости с ATmega103 первые 4096 ячеек памяти данных относятся к файлу регистров, памяти ввода/вывода и внутренней статической SRAM данных. В первых 32 ячейках расположен файл регистров, затем в 64 ячейках расположена стандартная память ввода/вывода и следующие 4000 ячейки занимает внутренняя SRAM данных.

Кроме показанных на рис. 1.3 Flash-памяти программ и памяти данных статического типа в микроконтроллере имеется встроенная электрически стираемая перепрограммируемая память данных (EEPROM). Память данных EEPROM организована как отдельное пространство данных с возможностью

записи и чтения отдельного байта. EEPROM обеспечивает 100000 циклов стирания/записи. Взаимодействие между EEPROM и центральным процессорным модулем (CPU) определяется регистром адреса EEPROM, регистром данных EEPROM и регистром управления EEPROM.

Для многих задач 4 Кбайта встроенной оперативной памяти данных SRAM является вполне достаточным. В случае необходимости ее объем можно увеличить до 64 Кбайт. Схема подключения внешней SRAM к микроконтроллеру показана на рис. 1.4. Интерфейс взаимодействия включает в себя:

AD0..AD7 – мультиплексированную младшую шину адреса/шину данных;

A0..A7 – старшую шину адреса;

ALE – строб адреса внешней памяти;

RD – строб чтения из внешней памяти;

WR – строб записи во внешнюю память.

Поскольку взаимодействие ATmega128 с внешними устройствами осуществляется посредством 8-разрядных портов, то для формирования 16-разрядного адреса ячейки памяти внешней SRAM используется внешняя микросхема – 8-разрядный регистр, в который на первом такте операции чтения/записи данных записывается младший байт адреса ячейки внешней памяти, к которой происходит обращение. На втором такте из ATmega128 в SRAM напрямую передается старший байт адреса и затем по установленному таким образом 16-разрядному адресу ячейки памяти происходит чтение или запись байта данных. Более детальную информацию о подключении внешней SRAM можно найти в технической документации на ATmega128.

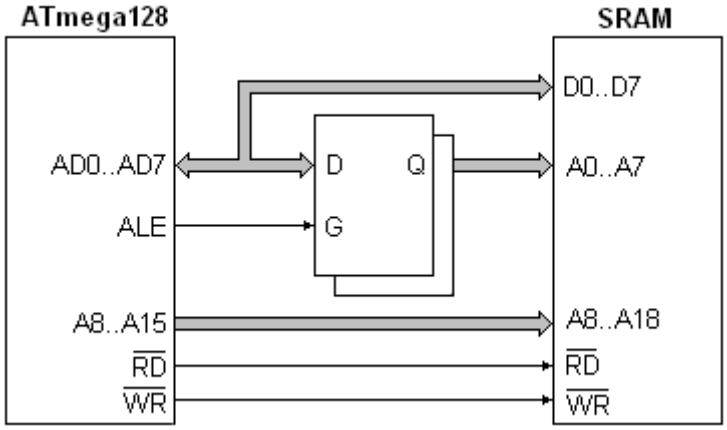

Рис. 1.4. Схема подключения внешней SRAM к ATmega128

Доступ к внешней статической SRAM осуществляется автоматически с помощью тех же команд, что и для внутренней SRAM, если указанное значение адреса находится за пределами внутренней памяти данных. При адресации внутренней памяти сигналы чтения и записи внешней памяти (сигналы RD и WR на выводах PG0 и PG1) неактивны в процессе всего цикла доступа. Работа внешней статической памяти данных SRAM разрешается установкой бита SRE в регистре **MCUCR**, который будет рассмотрен ниже.

Примыкающие к регистрам общего назначения 64 регистра образуют память ввода/вывода (память I/O), а следующие за ними 160 регистра – расширенную память I/O. Как следует из названия регистров памяти I/O, большинство из них предназначено для работы с различными функциональными блоками микроконтроллера такими, как порты ввода/вывода, АЦП, аналоговый компаратор, таймеры/счетчики, SPI, USART, EEPROM, сторожевой таймер. Подробная информация об этих регистрах будет представлена ниже при рассмотрении работы соответствующих функциональных блоков. Здесь же будут рассмотрены лишь те регистры, которые влияют на работу памяти программ и памяти данных.

Регистр статуса **SREG** содержит информацию о результате только что выполненной команды. Данная информация может использоваться для ветвления программы по условию. Следует понимать, что регистр статуса обновляется после выполнения всех операций АЛУ, которые модифицируют флаги, входящие в состав регистра статуса. Флаги этого регистра в большинстве случаев позволяют отказаться от использования команд сравнения, делая код программы более компактным и быстрым. Регистр статуса размещен в пространстве SRAM по адресу $5F. Назначение битов и их символические обозначения приведены в таблице 1.2.

Таблица 1.2

Регистр статуса **SREG**

Бит	Символ	Имя и назначение
7	I	Бит разрешения глобального прерывания. Предназначен для разрешения (I=1) или запрета (I=0) всех прерываний. Управление разрешением конкретного прерывания выполняется содержимым регистров **EIMSK** и **TIMSK**. Бит I аппаратно очищается после генерации запроса на прерывания и устанавливается в лог.1 для последующего разрешения глобального прерывания командой RETI.
6	T	Бит сохранения копии. Команды копирования бита BLD (Bit Load) и бит BST (Bit Store) используют бит T как бит-источник и бит-приемник при операциях с битами. Командой BST бит регистра регистрового файла копируется в бит T, командой BLD бит T копируется в регистр регистрового файла.
5	H	Флаг вспомогательного переноса (полупереноса), возникающего при выполнении некоторых арифметических операций. В основном используется в двоично-десятичной арифметике.
4	S	Бит знака, определяемый выражением S=N⊕V, где символ ⊕ обозначает логическую операцию «исключающее ИЛИ».
3	V	Флаг-указатель переполнения результата арифметических и логических операций, выраженного в дополнительном коде.
2	N	Флаг отрицательного значения, устанавливаемый при отрицательном значении результата ряда арифметических и логических операций.
1	Z	Флаг нулевого значения, устанавливаемый при нулевом значении результата ряда арифметических и логических операций.
0	C	Флаг переноса.

Регистр управления **MCUCR** управляет выполнением основных функций MCU. Назначение битов в регистре **MCUCR** и их символические обозначения приведены в таблице 1.3.

Таблица 1.3

Регистр управления **MCUCR**

Бит	Символ	Имя и назначение
7	SRE	Разрешение внешней SRAM. Установленный в состояние лог.1 бит SRE разрешает обращение к внешней SRAM данных и переводит работу выводов AD0…AD7 (порт A) и AD8…AD15 (порт C), а также выводов ALE, WR и RD на выполнение альтернативных функций.
6	SRW10	Бит выбора состояния ожидания внешней SRAM. При работе микроконтроллера не в режиме совместимости с ATmega103 влияние данного бита описывается ниже при рассмотрении регистра **XMCRA**. В режиме совместимости с ATmega103 при SRW10=1 к циклу обращения к внешней SRAM добавляется один цикл ожидания.
5	SE	Разрешение режима Sleep. Установленный в лог.1 бит SE разрешает перевод MCU в «спящий» режим командой SLEEP.
4	SM1	Биты SM1, SM0 и SM2 определяют один из шести возможных режимов Sleep, которые подробно описаны в технической документации на ATmega128.
3	SM0	
2	SM2	
1	IVSEL	Выбор вектора прерывания. Если бит IVSEL сброшен (IVSEL=0), то векторы прерываний размещаются в начале флэш-памяти. Если данный бит установлен в лог.1, то векторы прерываний перемещаются в начало загрузочного сектора флэш-памяти. Фактический адрес начала загрузочного сектора определяется значением конфигурационного бита BOOTSZ. По умолчанию IVSEL=0.
0	IVCE	Разрешение изменения вектора прерывания. Данный бит используется при операции изменения значения бита IVSEL.

1.3.3. Системная синхронизация и источники тактовых импульсов

Работа любого микроконтроллера включает в себя выборку команд из памяти и их дешифрацию, выборку операндов из ОЗУ и выполнение над ними определенных действий, запись и чтение информации из определенных регистров, обмен информацией ядра ЦПУ с периферийными устройствами и т. д. Для выполнения всех этих действий микроконтроллер необходимо тактировать синхроимпульсами определенной частоты (для ATmega128 эта частота не превышает 16 МГц).

В микроконтроллерах семейства AVR реализованы несколько источников тактовых импульсов: генератор с кварцевым или керамическим резонатором (рис. 1.5а), RC-генератор (рис. 1.5б), внешний источник синхроимпульсов (рис. 1.5в). Предусмотрена также возможность синхронизации импульсами от внутреннего калиброванного RC-генератора.

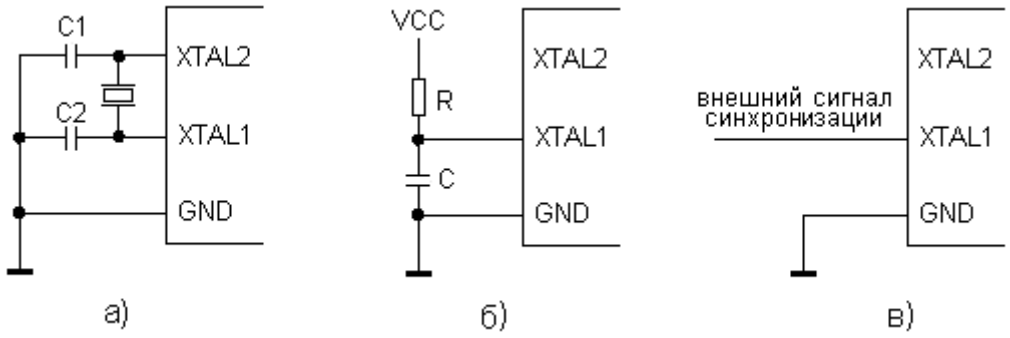

Рис. 1.5. Различные источники тактовых импульсов для AVR-микроконтроллеров

Выбор того или иного источника задается значениями четырех конфигурационных битов CKSEL3…0 согласно таблице 1.4. Для всех конфигурационных битов лог.1 означает незапрограммированное состояние, а лог.0 – запрограммированное.

Таблица 1.4

Выбор источника синхронизации AVR-микроконтроллера

Источники синхронизации	CKSEL3...0
Внешний кварцевый/керамический резонатор	1111 – 1010
Внешний низкочастотный кварцевый резонатор	1001
Внешний RC-генератор	1000 – 0101
Встроенный калиброванный RC-генератор	0100 – 0001
Внешняя синхронизация	0000

В случае выбора внешнего кварцевого/керамического резонатора диапазон рабочих частот зависит от конфигурационного бита CKOPT. Если данный бит запрограммирован (CKOPT=0), то максимальная частота синхроимпульсов для ATmega128 равна 16 МГц, в противном случае – 8 МГц. Подключение резонатора к выводам микроконтроллера осуществляется согласно рис. 1.5а, при этом емкости конденсаторов C1 и C2 выбираются из диапазона от 12 до 22 пФ.

В случае выбора внешнего низкочастотного кварцевого генератора к выводам микроконтроллера XTAL1 и XTAL2 подключается часовой кварцевый резонатор (частота равна 32768 Гц). Если при этом запрограммирован конфигурационный бит CKOPT, то никаких внешних конденсаторов не требуется.

Для приложений, некритичных к стабильности временных характеристик, в качестве источника синхроимпульсов может использоваться внешняя RC-цепь, подключение которой показано на рис. 1.5б. Вывод микроконтроллера XTAL2 остается свободным. Тактовую частоту при этом можно грубо оце-

нить по формуле f = 1/(3RC). Номинал конденсатора C при этом должен быть не менее 22 пФ. Путем программирования конфигурационного бита CKOPT пользователь может разрешить подключение внутреннего конденсатора C = 36 пФ между XTAL1 и GND, тем самым исключая необходимость применения внешнего конденсатора.

Одним из широко используемых вариантов выбора источника синхронизации микроконтроллера является встроенный калиброванный RC-генератор, который формирует фиксированные тактовые частоты 1.0, 2.0, 4.0 или 8.0 МГц. Конкретное значение частоты задается двоичным кодом, образованным конфигурационными битами CKSEL2, CKSEL1 и CKSEL0. Например, если код равен 001, то частота равна 1 МГц, а если код 111, то частота 8.0 МГц. При этом конфигурационный бит CKOPT должен быть незапрограммированным. При напряжении питания 5 В, температуре 25° С и выбранной частоте генератора 1.0 МГц данный метод калибровки обеспечивает погрешность генерации частоты не хуже ± 3 % от номинального значения. Следует отметить, что фирма-производитель поставляет микроконтроллеры с установкой, при которой его синхронизация осуществляется встроенным RC-генератором с частотой 1 МГц.

Микроконтроллер можно синхронизировать и от внешнего источника. В этом случае его необходимо подключить к выводу XTAL1 (рис. 1.5в), а вывод XTAL2 оставить свободным. Значения всех конфигурационных битов CKSEL при этом должны быть равны нулю, а бит CKOPT запрограммирован.

Источником тактовых импульсов в микроконтроллерах семейства AVR может служить также генератор таймера/счетчика. В этом случае к выводам микроконтроллера TOSC1 и TOSC2 (выводы 3 и 4 порта G) необходимо подключить кварцевый резонатор. Генератор оптимизирован для совместной работы с часовым кварцевым резонатором 32768 Гц. Данный тактовый генератор позволяет использовать таймер/счетчик как счетчик реального времени, даже при переводе микроконтроллера в один из режимов энергосбережения.

При использовании микроконтроллера в одном из выбранных режимов синхронизации программным способом можно изменить частоту импульсов, тактирующих все структурные блоки микроконтроллера. Это, в частности, может быть полезно в целях экономии энергопотребления, если быстродействие некритично. Изменение частоты тактовых импульсов осуществляется с помощью регистра управления коэффициентом деления частоты кварцевого генератора **XDIV**. Данный регистр предназначен для установления коэффициента деления частоты в диапазоне от 1 до 129. Назначение битов следующее. Бит 7 регистра XDIV (XDIVEN) разрешает деление частоты. При XDIVEN=1 тактовая частота делится в соответствии с установленными битами XDIV6…XDIV0 коэффициентом деления. Если десятичное значение коэффициента деления, установленное битами XDIV6…XDIV0, обозначить через k, то результирующая тактовая частота микроконтроллера будет определяться по формуле:

$$f = \frac{XTAL}{129 - k},$$

где через XTAL обозначена частота выбранного источника синхронизации. Если значение бита XDIVEN=0, то записанные в биты XDIV6…XDIV0 значения коэффициентов деления игнорируются.

1.3.4. Конфигурационные биты

В микроконтроллерах семейства AVR имеются конфигурационные биты (Fuse Bits), содержимое которых влияет на работу тех или иных структурных блоков. Программирование этих битов позволяет гибко изменять режимы работы микроконтроллера. В ATmega128 конфигурационные биты сгруппированы в три байта: младший, старший и расширенный. Символические обозначения и назначения битов представлены в таблицах 1.5 – 1.7. Следует учесть, что лог.1 соответствует незапрограммированному состоянию, а лог.0 – запрограммированному. Например, если исходное состояние бита M103C = 0, то данный бит запрограммирован и микроконтроллер ATmega128 работает в режиме совместимости с ATmega103.

Таблица 1.5

Младший конфигурационный байт

Обозначение	Разряд	Назначение бита	Исходное значение
BODLEVEL	7	Порог срабатывания супервизора питания	1
BODEN	6	Разрешение супервизора питания	1
SUT1	5	Выбор времени запуска	1
SUT0	4	Выбор времени запуска	0
CKSEL3	3	Выбор тактового источника	0
CKSEL2	2	Выбор тактового источника	0
CKSEL1	1	Выбор тактового источника	0
CKSEL0	0	Выбор тактового источника	1

Таблица 1.6

Старший конфигурационный байт

Обозначение	Разряд	Назначение бита	Исходное значение
OCDEN	7	Включение встроенного блока отладки	1
JTAGEN	6	Включение JTAG-интерфейса	0
SPIEN	5	Разрешение последовательной загрузки программы и данных	0
CKOPT	4	Настройка генератора	1
EESAVE	3	Запрет стирания ЭСППЗУ командой стирание кристалла	1
BOOTSZ1	2	Выбор размера загрузочного сектора	0
BOOTSZ0	1	Выбор размера загрузочного сектора	0
BOOTRST	0	Выбор вектора сброса	1

Таблица 1.7

Расширенный конфигурационный байт

Обозначение	Разряд	Назначение бита	Исходное значение
M103C	1	Режим совместимости с ATmega103	0
WDTON	0	Активизация сторожевого таймера	1

Биты CKSEL0… CKSEL3 совместно с битом CKOPT определяют режимы синхронизации микроконтроллера (тип источника тактовых импульсов и их частоту). Биты SUT0 и SUT1 задают двоичный код, значение которого определяет временную задержку при пуске микроконтроллера (или после формирования сигнала RESET). Эта временная задержка необходима для того, чтобы не допустить работу микроконтроллера при еще неустановившейся частоте синхроимпульсов. Исходные значения SUT1 и SUT0 соответствуют выбору максимального времени рестарта. Назначение остальных конфигурационных битов будет понятно после рассмотрения функциональных блоков, на работу которых влияют данные биты.

Конфигурационные биты, также как и прикладные программы, обычно записываются («прошиваются») в память микроконтроллера с помощью последовательного интерфейса SPI. Разрешение или запрет последовательного программирования с помощью SPI осуществляется с помощью конфигурационного бита SPIEN. В исходном состоянии этот бит запрограммирован (SPIEN = 0), то есть последовательное программирование разрешено. Если изменить значение этого бита на лог.1, то последовательное программирование будет запрещено, что серьезно осложнит работу с микроконтроллером. Чтобы такая ситуация не возникала, изменение значения бита SPIEN с помощью последовательного интерфейса SPI сделано невозможным.

Кроме конфигурационных битов в AVR-микроконтроллерах имеются биты защиты (Lock Bits) и биты защиты загрузочного сектора (Boot lock bit), предотвращающие несанкционированный доступ к памяти микроконтроллера. Комбинации этих битов определяют различные режимы защиты флэш-памяти и ЭСППЗУ. В исходном состоянии эти биты не запрограммированы, то есть никакой защиты нет.

1.4. Порты ввода/вывода

Порты ввода/вывода служат для обмена информацией между микроконтроллером и внешними устройствами. В микроконтроллерах ATmega128 имеется шесть 8-разрядных двунаправленных портов ввода/вывода (**PORTA…PORTF**) и один 5-разрядный двунаправленный порт **PORTG**. Взаимодействие с каждым из выводов любого порта осуществляется посредством трех регистров: регистром данных (регистры **PORTA…PORTG**), регистром направления данных (регистры **DDRA…DDRG**) и регистром состояния входов порта (регистры **PINA…PING**).

Структурная схема отдельного вывода порта Pxn показана на рис. 1.6 (здесь же приведена и расшифровка управляющих сигналов). Через «x» обозначено имя порта (x = A, B…G), а через «n» – номер вывода порта (n = 0, 1…7). Все выводы (разряды) регистров данных **PORTxn** и регистров направления данных **DDxn** программно доступны как для записи, так и для чтения. Разряды порта **PINxn** доступны только для чтения.

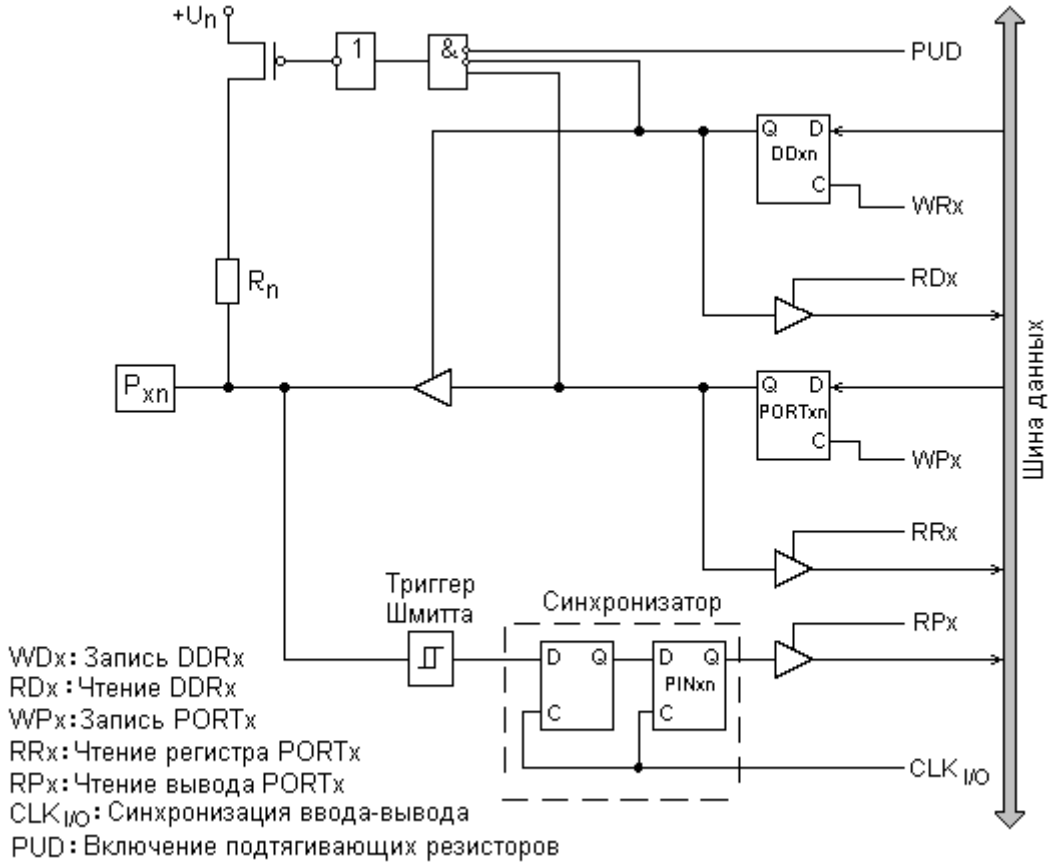

Рис. 1.6. Структурная схема отдельного вывода порта Pxn

Регистры направления данных позволяют настраивать отдельные разряды портов ввода/вывода либо на вход (управляющий бит DDxn в состоянии лог.0), либо на выход (управляющий бит DDxn в состоянии лог.1). Если порт или его отдельные разряды настроены на выход, то запись в какой-либо разряд порта логической единицы или нуля переводит соответствующий вывод порта в высокое или низкое состояние. Эта информация хранится в защелках порта **PORTxn**. Она остается неизменной вне зависимости от изменений физического состояния вывода порта Pxn.

Чтение отдельных разрядов регистров состояния входов порта PINxn позволяет определить физическое состояние отдельных выводов портов Pxn. Выходные буферы портов ввода/вывода обеспечивают втекающий ток до 40 мА, что вполне достаточно для непосредственного управления различного рода внешними устройствами, например, индикаторами. Важно отметить, что все выводы портов имеют встроенные нагрузочные (подтягивающие) резисторы Rn.

Они автоматически подключаются к шине питания в случае, если данный разряд порта настроен на вход (DDxn = 0), а в соответствующий разряд регистра данных записана лог.1 (PORTxn = 1). Если соответствующий вывод порта используется как вход и на нем внешним сигналом удерживается низкий уровень, то подтягивающий резистор обеспечивает необходимый втекающий ток. Отключить одновременно все подтягивающие резисторы у всех портов можно записью бита PUD в регистре SFIOR (по умолчанию PUD = 0).

Все порты кроме общего назначения (ввод/вывод цифровой информации) способны выполнять дополнительные функции. Так выводы порта A обеспечивают взаимодействие с внешней дополнительной памятью данных, а именно, они могут быть использованы в качестве шины адреса/данных для внешней SRAM (младший байт). Дополнительная функция порта A включается установкой бита SRE (разрешение внешней SRAM) в регистре **MCUCR**, при этом установки регистра направления данных игнорируются. Дополнительной функцией порта C является вывод старшего байта адреса внешней SRAM. Активизация этой функции осуществляется управляющим битом SRE.

Дополнительной функцией порта B является обеспечение работы последовательного периферийного интерфейса SPI и таймеров/счетчиков. Включение выводов порта B для выполнения дополнительных функций производится с помощью соответствующей установки битов в регистре данных **PORTB** и регистре направления данных **DDRB**. Дополнительные функции порта B приведены в таблице 1.8. Более подробно они будут описаны ниже при рассмотрении работы интерфейса SPI и таймеров/счетчиков.

Таблица 1.8

Дополнительные функции выводов порта B

Выводы порта	Обозначение	Дополнительные функции
PB0	SS	Вход выбора ведомого
PB1	SCK	Тактовый сигнал последовательной SPI шины
PB2	MOSI	Установка ведущий выход/ведомый вход SPI шины
PB3	MISO	Установка ведущий вход/ведомый выход SPI шины
PB4	OC0	Выход компаратора и ШИМ-сигнал таймера/счетчика 0
PB5	OC1A	Выход A компаратора и ШИМ-сигнал таймера/счетчика 1
PB6	OC1B	Выход B компаратора и ШИМ-сигнал таймера/счетчика 1
PB7	OC2/OC1C	Выход компаратора и ШИМ-сигнал таймера/счетчика 2 или выход C компаратора и ШИМ-сигнал таймера-счетчика 1

Дополнительной функцией порта D является обеспечение работы системы прерываний, таймеров/счетчиков, двухпроводного последовательного интерфейса TWI и универсального синхронно-асинхронного приемопередатчика USART. Включение выводов порта D для выполнения дополнительных функций производится с помощью соответствующей установки битов в регистре данных **PORTD** и регистре направления данных **DDRD**. Дополнительные функ-

ции порта D приведены в таблице 1.9. Более подробно они будут описаны ниже при рассмотрении работы соответствующих блоков и системы прерываний.

Таблица 1.9

Дополнительные функции выводов порта D

Выводы порта	Обозначение	Дополнительные функции
PD0	INT0/SCL	Вход внешнего прерывания 0 или синхронизация последовательного интерфейса TWI
PD1	INT1/SDA	Вход внешнего прерывания 1 или ввод/вывод данных через интерфейс TWI
PD2	INT2/RXD1	Вход внешнего прерывания 2 или вход приемника USART1
PD3	INT3/TXD1	Вход внешнего прерывания 3 или выход передатчика USART1
PD4	IC1	Вход триггера захвата фронта таймера-счетчика 1
PD5	XCK1	Внешняя синхронизация USART1
PD6	T1	Вход синхронизации таймера-счетчика 1
PD7	T2	Вход синхронизации таймера-счетчика 2

Дополнительной функцией порта E является обеспечение работы системы прерываний, а также аналогового компаратора и универсального асинхронного приемопередатчика UART. Включение выводов порта E для выполнения дополнительных функций производится с помощью соответствующей установки битов в регистре данных PORTE и регистре направления данных DDRE. Дополнительные функции порта E приведены в таблице 1.10. Более подробно они будут описаны ниже при рассмотрении работы компаратора и последовательного порта.

Таблица 1.10

Дополнительные функции выводов порта E

Выводы порта	Обозначение	Дополнительные функции
PE0	PDI/RXD0	Ввод программируемых данных или вывод приема USART0
PE1	PDO/TXD0	Вывод программируемых данных или вывод передатчика USART0
PE2	AIN0/XCK0	Неинвертирующий вход аналогового компаратора или вход/выход внешний синхронизации USART0
PE3	AIN1/OC3A	Инвертирующий вход аналогового компаратора или выход A компаратора и ШИМ-сигнал таймера-счетчика 3
PE4	INT4/OC3B	Вход внешнего прерывания 4 или выход B компаратора и ШИМ-сигнал таймера-счетчика 3
PE5	INT5/OC3C	Вход внешнего прерывания 5 или выход C компаратора и ШИМ-сигнал таймера-счетчика 3
PE6	INT6/T3	Вход внешнего прерывания 6 или вход синхронизации таймера-счетчика 3
PE7	INT7/IC3	Вход внешнего прерывания 7 или вход триггера захвата фронта таймера-счетчика 3

Дополнительной функцией выводов порта F (таблица 1.11) является использование их в качестве входов аналогового мультиплексора, подключенного, в свою очередь, к АЦП. Это позволяет осуществлять аналого-цифровое преобразование одновременно восьми аналоговых сигналов. Кроме этого, часть выводов порта используется для работы JTAG-интерфейса.

Таблица 1.11

Дополнительные функции выводов порта F

Выводы порта	Обозначение	Дополнительные функции
PF0	ADC0	Вход канала 0 АЦП
PF1	ADC1	Вход канала 1 АЦП
PF2	ADC2	Вход канала 2 АЦП
PF3	ADC3	Вход канала 3 АЦП
PF4	ADC4/TCK	Вход канала 4 АЦП или синхронизация JTAG тестирования
PF5	ADC5/TMS	Вход канала 5 АЦП или выбор режима JTAG тестирования
PF6	ADC6/TDO	Вход канала 6 АЦП или вывод данных при JTAG тестировании
PF7	ADC7/TDI	Вход канала 7 АЦП или ввод данных при JTAG тестировании

Дополнительные функции выводов порта G представлены в таблице 1.12. Они используются для организации чтения/записи во внешнюю память данных, а также служат для подключения часового кварцевого резонатора, что позволяет реализовать в микроконтроллере часы реального времени.

Таблица 1.12

Дополнительные функции выводов порта G

Выводы порта	Обозначение	Дополнительные функции
PG0	WR	Строб записи внешней памяти
PG1	RD	Строб чтения внешней памяти
PG2	ALE	Разрешение фиксации адреса внешней памяти
PG3	TOSC2	Генератор часов реального времени таймера-счетчика 0
PG4	TOSC1	Генератор часов реального времени таймера-счетчика 0

1.5. Система прерываний и сброса

Прерыванием (Interrupt) применительно к микроконтроллерам называется прекращение выполнения обычной программы (фоновой программы) по запросу от внутреннего периферийного или от внешнего устройства с одновремен-

ным переходом к процедуре обслуживания прерывания. Получив запрос прерывания от какого-нибудь источника (внешнего или внутреннего) микроконтроллер должен закончить выполнение текущей команды, расшифровать, что за источник вызвал этот запрос, а затем перейти к соответствующей процедуре обработки прерывания, предварительно сохранив в стеке адрес возврата в фоновую программу. Переход к процедуре обработки прерывания осуществляется с помощью так называемых векторов прерывания, которые представляют собой заранее определенные ячейки памяти с записанными в них командами безусловного перехода по адресам, где хранятся соответствующие процедуры обработки прерывания.

Запрос прерывания сопровождается установкой соответствующего флага, который должен сбрасываться аппаратно или программно после обработки данного прерывания. При выполнении процедуры обработки прерывания может поступить новый запрос от другого источника прерываний. В этом случае перед контроллером встает проблема выбора: проигнорировать вновь поступивший запрос, или прервать обработку ранее поступившего прерывания и перейти к обработке нового, а затем завершить обработку первого по времени прерывания. Выбор действия осуществляется на основе анализа приоритета поступившего прерывания – если он выше у вновь поступившего прерывания, то его обработка и осуществляется в первую очередь. Таким образом, уровни приоритетов, установленные для каждого типа прерывания, предотвращают возникновение возможных конфликтных ситуаций.

Микроконтроллеры ATmega128 имеют 35 источников прерываний и сброса. Каждому из них соответствует свой вектор в пространстве памяти программ. Для разрешения конкретного прерывания должен быть установлен соответствующий бит разрешения, а также бит I в регистре статуса **SREG** (бит разрешения глобального прерывания). Полный перечень векторов прерываний, их адреса в памяти программ и наименования приведены в таблице 1.13. Расположение векторов в таблице соответствует их уровням приоритета. Прерывания с младшими адресами имеют больший уровень приоритета. Например, RESET имеет наивысший уровень приоритета.

Таблица 1.13

Векторы прерываний и сброса

Номер вектора	Адрес	Источник	Наименование
1	$0000	RESET	Внешний сброс, сброс при подаче питания, сброс при недопустимом снижении питания, сброс сторожевым таймером и сброс от JTAG
2	$0002	INT0	Запрос внешнего прерывания 0
3	$0004	INT1	Запрос внешнего прерывания 1
4	$0006	INT2	Запрос внешнего прерывания 2
5	$0008	INT3	Запрос внешнего прерывания 3
6	$000A	INT4	Запрос внешнего прерывания 4
7	$000C	INT5	Запрос внешнего прерывания 5

Окончание таблицы 1.13

Номер вектора	Адрес	Источник	Наименование
8	$000E	INT6	Запрос внешнего прерывания 6
9	$0010	INT 7	Запрос внешнего прерывания 7
10	$0012	TIMER2 COMP	Совпадение OCR2 с TCNT2 в T/C 2
11	$0014	TIMER2 OVF	Переполнение T/C 2
12	$0016	TIMER1 CAPT	Срабатывание блока захвата в T/C 1
13	$0018	TIMER1 COMPA	Совпадение OCR1A с TCNT1 в T/C 1
14	$001A	TIMER1 COMPB	Совпадение OCR1B с TCNT1 в T/C 1
15	$001C	TIMER1 OVF	Переполнение T/C 1
16	$001E	TIMER0 COMP	Совпадение OCR0 с TCNT0 в T/C 0
17	$0020	TIMER0 OVF	Переполнение T/C 0
18	40022	SPI, STS	Окончание пересылки SPI
19	$0024	USART0, RX	Окончание приема USART0
20	$0026	USART0, UDRE	Регистр данных USART0 пуст
21	$0028	USART0, TX	Окончание передачи USART0
22	$002A	ADC	Окончание операции преобразования АЦП
23	$002C	EE READY	Готовность EEPROM
24	$002E	ANALOG COMP	Срабатывание аналогового компаратора
25	$0030	TIMER1 COMPC	Совпадение OCR1C с TCNT1 в T/C 1
26	$0032	TIMER3 CAPT	Срабатывание блока захвата в T/C 3
27	$0034	TIMER3 COMPA	Совпадение OCR1A с TCNT3 в T/C 3
28	$0036	TIMER3 COMPB	Совпадение OCR1B с TCNT3 в T/C 3
29	$0038	TIMER3 COMPC	Совпадение OCR1C с TCNT3 в T/C 3
30	$003A	TIMER3 OVF	Переполнение T/C 3
31	$003C	USART1, RX	Окончание приема USART1
32	$003E	USART1, UDRE	Регистр данных USART1 пуст
33	$0040	USART1, TX	Окончание передачи USART1
34	$0042	TWI	Запрос прерывания от TWI
35	$0044	SPM READY	Готовность записи в память программ

Сигнал внешнего сброса формируется при поступлении на внешний вывод сигнала RESET (активный уровень – низкий). При этом сигнал RESET должен удерживаться на низком уровне в течение времени, не менее двух тактовых циклов кварцевого резонатора. Сигнал сброса по включению питания формируется при включении питания, как только напряжение Vcc достигнет определенного значения (2,0 ± 0,2 В). Сигнал сброса по сторожевому таймеру формируется, как только истекает период сторожевого таймера при условии, что работа сторожевого таймера разрешена. Два первых источника сброса, вызвав перезапуск микроконтроллера, устанавливают соответствующие флаги в регистре статуса MCUCSR: флаг внешнего сброса EXTRF и флаг включения питания PORF.

Как следует из таблицы 1.14, источники прерывания могут быть разные. Это могут быть периферийные устройства микроконтроллера, а именно, таймеры/счетчики T/C, универсальные синхронно/асинхронные приемопередатчики

USART, последовательный периферийный интерфейс SPI, аналого-цифровой преобразователь, аналоговый компаратор и ряд других. В микроконтроллере имеются 8 внешних прерываний, которые генерируются при определенных изменениях напряжения на выводах INT0...INT7. Запросы внешнего прерывания могут генерироваться по падающему или нарастающему фронту, а также по низкому логическому уровню. При этом они будут генерироваться даже в случае, если линии INT0...INT7 настроены как выходы.

Условия генерации запроса определяются битами в регистрах управления внешними прерываниями **EICRA** и **EICRB** (таблицы 1.14 и 1.15). Если выбрано прерывание по низкому уровню, то для генерации прерывания необходимо, чтобы этот уровень оставался на прежнем низком уровне до момента завершения выполнения текущей инструкции. После разрешения прерывания по уровню оно будет генерироваться непрерывно до тех пор, пока на входе присутствует низкий уровень.

Таблица 1.14

Регистр А управления внешними прерываниями **EICRA**

Бит	Символ	Имя и назначение
7 6 5 4 3 2 1 0	ISC31 ISC30 ISC21 ISC20 ISC11 ISC10 ISC01 ISC00	Биты управления опознаванием внешних прерываний с нулевого по третий. Биты ISCx1 и ISCx0 формируют двоичный код, который определяет тип сигнала, который способен вызвать запрос прерывания. Тип сигнала определяется согласно схеме: ISCx1=0 ISCx0=0 прерывание генерируется низким уровнем; ISCx1=0 ISCx0=1 зарезервировано; ISCx1=1 ISCx0=0 генерируется падающим фронтом; ISCx1=1 ISCx0=1 генерируется нарастающим фронтом. **Примечание.** Символ x принимает значения 3, 2, 1 и 0.

Таблица 1.15

Регистр В управления внешними прерываниями **EICRB**

Бит	Символ	Имя и назначение
7 6 5 4 3 2 1 0	ISC71 ISC70 ISC61 ISC60 ISC51 ISC50 ISC41 ISC40	Биты управления опознаванием внешних прерываний с четвертого по седьмой. Биты ISCx1 и ISCx0 формируют двоичный код, который определяет тип сигнала, который способен вызвать запрос прерывания. Тип сигнала определяется согласно схеме: ISCx1=0 ISCx0=0 прерывание генерируется низким уровнем; ISCx1=0 ISCx0=1 генерируется при любом изменении уровня; ISCx1=1 ISCx0=0 генерируется падающим фронтом; ISCx1=1 ISCx0=1 генерируется нарастающим фронтом. **Примечание.** Символ x принимает значения 7, 6, 5 и 4.

Для определения фронтов на выводах INT4…INT7 осуществляется выборка их состояний. Если выбрано прерывание по фронту или изменению уровня, то прерывание будет сгенерировано, если на входе появляется импульс, длительность которого больше одного периода синхронизации. При действии на входе более коротких импульсов генерация прерывания не гарантируется.

Кроме регистров управления **EICRA** и **EICRB** на работу системы прерываний оказывает влияние регистр масок внешних прерываний **EIMSK** и регистр флагов внешних прерываний **EIFR.** Назначение битов регистра **EIMSK** представлено в таблице 1.16. В битах регистра **EIFR** при возникновении соответствующих запросов на прерывания устанавливаются флаги внешних прерываний INTF0…INTF7. Флаги очищаются аппаратно сразу после выполнения процедуры обработки прерывания. Альтернативно флаг может быть сброшен программно путем записи лог. 1 в соответствующий бит.

Таблица 1.16

Регистр масок внешних прерываний **EIMSK**

Бит	Символ	Имя и назначение
7	INT7	Биты разрешения внешних прерываний. При установке любого их них в состояние лог.1 и установленном бите глобального разрешения прерываний (бит I в регистре статуса **SREG**) прерывания разрешаются. Тип сигналов на выводах микроконтроллера, при которых генерируется запрос на прерывание, определяется содержимым регистра управления.
6	INT6	
5	INT5	
4	INT4	
3	INT3	
2	INT2	
1	INT1	
0	INT0	

Важно подчеркнуть, что при возникновении запроса на прерывание флаг устанавливается всегда, а само прерывание, вызывающее приостановку выполнения фоновой программы и переход на процедуру обработки прерывания, возникает только при условии, что соответствующим битом в регистре **EIMSK** данное прерывание разрешено. Это относится не только к внешним прерываниям, но и к другим прерываниям, связанным с работой периферийных устройств микроконтроллера.

1.6. Периферийные устройства

1.6.1. Таймеры/счетчики

Таймеры/счетчики в микроконтроллерах предназначены для точного задания временных интервалов и измерения временных характеристик импульсных сигналов, поступающих на входы микроконтроллера, а также для генерации прямоугольных импульсов с возможностью их широтно-импульсной модуляции. Микроконтроллеры ATmega128 оснащены четырьмя таймерами/счетчиками общего назначения – двумя 8-разрядными (T/C0 и T/C2) и двумя 16-разрядным (T/C1 и T/C3). Таймер/счетчик T/C0 в дополнение к обычному режиму может тактироваться импульсами с частотой следования, задаваемой внешним резонатором. В этом случае к выводам микроконтроллера TOSC1 и TOSC2 обычно подключают кварцевый резонатор на 32 кГц, что позволяет использовать T/C0 в качестве часов реального времени.

Более широко используются 16-разрядные таймеры/счетчики. Рассмотрим работу таймера/счетчика T/C1 (T/C3 функционирует аналогично). Функциональная схема T/C1 приведена на рис. 1.7. Содержимое регистра TCNT1

увеличивается на единицу (инкрементируется) при каждом поступлении на его вход от блока управления тактирующего импульса. При некоторых режимах работы возможен и обратный отсчет, при котором содержимое регистра **TCNT1** при каждом такте декрементируется на единицу. Частота тактирующих импульсов равна частоте синхронизации микроконтроллера, деленной на программно задаваемый коэффициент (минимальное значение равно 1, максимальное – 1024). Таймер/счетчик может тактироваться и внешними импульсами, поступающими с вывода T1.

Работа таймера/счетчика организована следующим образом. При разрешенной работе T/C1 происходит непрерывное сравнение содержимого регистра **TCNT1** со значением регистров сравнения **OCR1A**, **OCR1B** или **OCR1C**, в которые можно записать любое двухбайтное число. В случае равенства **TCNT1** и какого-либо **OCR1x** (x = A, B, C) в специальном регистре **TIFR** (на схеме не показан) устанавливается соответствующий флаг сравнения OCF1x. Если при этом разрешено прерывание, то параллельно с установкой флага формируется запрос на прерывание, при котором выполнение следующего оператора программы приостанавливается и управление передается программному модулю, осуществляющему обработку данного прерывания. Флаг OCF1x автоматически сбрасывается после начала обработки прерывания. Альтернативно его можно сбросить программно, если записать в него лог. 1.

Таким образом, изменяя содержимое регистров сравнения и частоту тактирования таймера, можно задавать различные временные интервалы, минимальное значение которых равно периоду следования тактирующих импульсов. Можно также варьировать длительность и период следования импульсов, формируемых на соответствующих выводах микроконтроллера, т.е. осуществлять их широтно-импульсную модуляцию. Для T/C1 эти ШИМ-импульсы формируются на выводах OC1x (x = A, B, C).

Таймер-счетчик содержит блок захвата, который запоминает состояние регистра **TCNT1** при возникновении внешнего события, тем самым определяя время его возникновения. В качестве события может выступать изменение логического уровня сигнала от внешнего источника, подключенного к выводу ICx. Для T/C1 альтернативно в качестве источника внешнего события может использоваться аналоговый компаратор (для T/C3 эта возможность отсутствует). При возникновении события 16-разрядное значение содержимого таймера **TCNT1** помещается в регистр захвата **ICR1** и устанавливается флаг захвата ICF1. Если разрешено прерывание по захвату события, то параллельно с установкой флага формируется запрос на прерывание. Флаг ICF1 сбрасывается автоматически при переходе на вектор прерывания. Альтернативно флаг ICFn1 сбрасывается программно, если записать в него лог. 1. Блок захвата может использоваться для вычисления частоты, скважности импульсов и других параметров внешних импульсных сигналов.

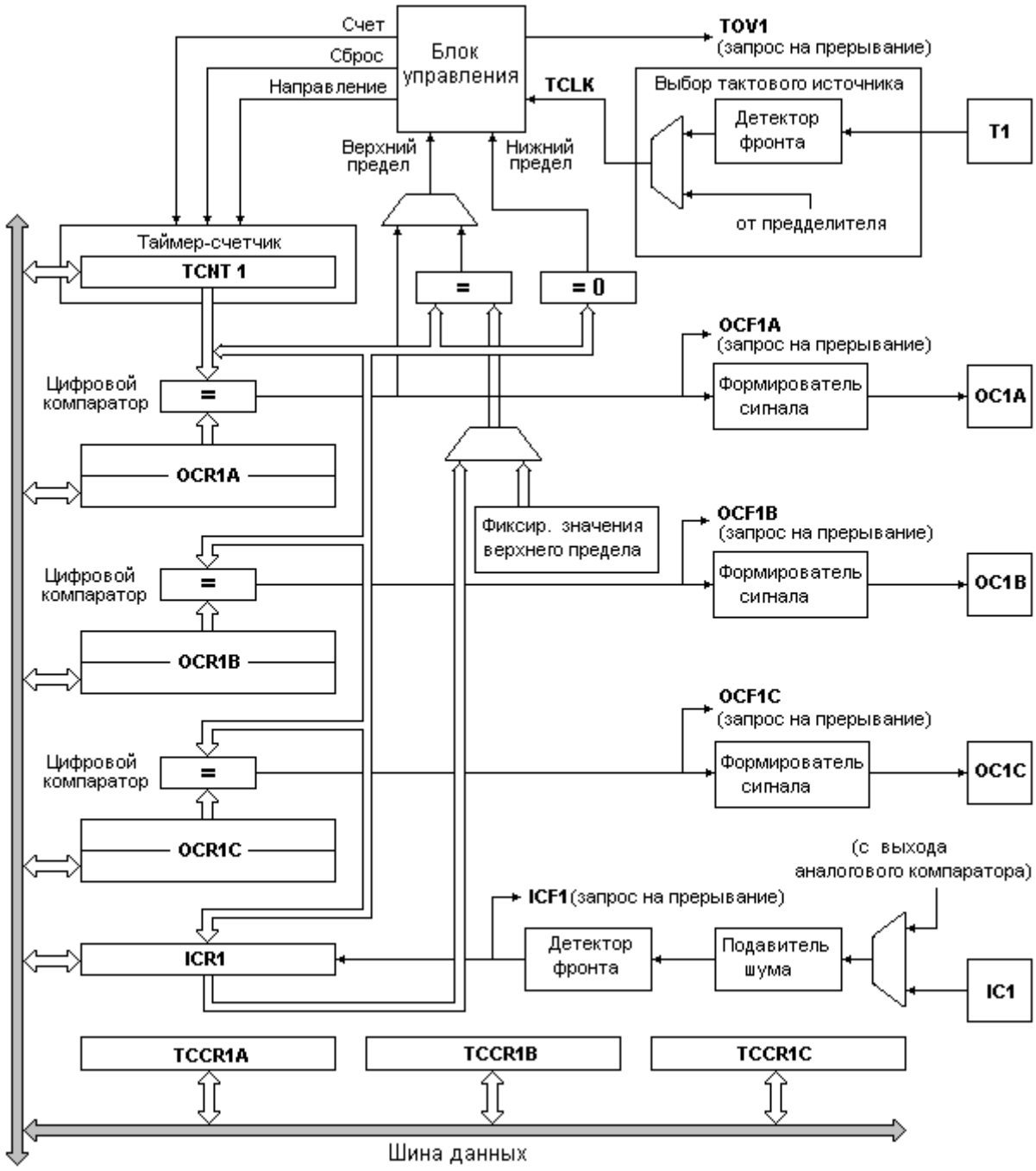

Рис. 1.7. Функциональная схема 16-разрядного таймера счетчика T/C1

Работой T/C1 управляют три регистра управления **TCCR1A**, **TCCR1B** и **TCCR1C**. Они определяют режимы работы генератора, формирующего импульсы с широтно-импульсной модуляцией, задают тактовую частоту таймера/счетчика, вершину счета (верхнее предельное значение регистра **TCNT1**, при достижении которого на следующем такте его значение становится равным нулю) и т.д. В качестве вершины счета могут выступать фиксированные значения 0x00FF, 0x01FF, 0x03FF или содержимое регистров **OCR1A** или **ICR1**.

Символические имена и функциональное назначение управляющих битов этих регистров приведены в таблицах 1.17, 1.18 и 1.19.

Таблица 1.17

Регистр управления **TCCR1A**

Бит	Символ	Имя и назначение
7	COM1A1	Биты COM1x1 и COM1x0 (x = A, B, C) определяют характер сигнала
6	COM1A0	вывода, следующего за совпадением при сравнении T/C1:
5	COM1B1	0 0 - сигналы OC1x отключены;
4	COM1B0	0 1 - переключение;
3	COM1C1	1 0 - сброс в состояние лог.0;
2	COM1C0	1 1 - установка в состояние лог.1.
1	WGM11	Биты WGM11 и WGM10 предназначены для установки режимов ра-
0	WGM10	боты таймера/счетчика T/C1 (см. таблицу 1.20).

Таблица 1.18

Регистр управления **TCCR1B**

Бит	Символ	Имя и назначение
7	ICNC1	Установка режима подавления шума на входе захвата. При ICNC1=0 функция подавления шума запрещена – вход захвата переключается по первому нарастающему/спадающему фронту сигнала, поступившего на вывод IC1. При ICNC1=1 выполняются 4 последовательных опроса состояния вывода IC1 и для реализации захвата все выборки должны иметь одинаковый (высокий/низкий) уровень, определяемый битом ICES1.
6	ICES1	Выбор детектируемого фронта на входе захвата. При ICES1=0 захват производится по падающему фронту сигнала на входе IC1, при ICES1=1 захват по нарастающему фронту сигнала.
5	-	Бит 5 зарезервирован.
4	WGM13	Биты WGM13 и WGM12 предназначены для установки режимов ра-
3	WGM12	боты таймера/счетчика T/C1 (см. таблицу 1.20).
2	CS12	Биты CS12, CS11 и CS10 формируют двоичный код, определяющий
1	CS11	тактовую частоту таймера/счетчика T/C1 (см. таблицу 1.21).
0	CS10	

Таблица 1.19

Регистр управления **TCCR1C**

Бит	Символ	Имя и назначение
7	FOC1A	FOC1A, FOC1B и FOC1C являются битами принудительной установ-
6	FOC1B	ки результата сравнения для канала A, B и C. Становятся активными,
5	FOC1C	когда с помощью бит WGMx3..0 выбран режим без ШИМ.
		Остальные биты зарезервированы

Биты WGM10 и WGM11 в регистре **TCCR1A** совместно с битами WGM12 и WGM13 в регистре **TCCR1B** предназначены для установки режимов работы таймера/счетчика. Под режимом работы понимается алгоритм счета и поведение связанного с ним выхода формирователя импульсов (выводы OC1x). В микроконтроллере реализованы следующие режима работы: нормальный, сброс при совпадении (CTC), быстрой ШИМ, а также ШИМ с фазовой и частотной коррекцией (ШИМ с ФК и ШИМ с ФЧК). Отличительные особенности всех этих режимов представлены в таблице 1.20.

Таблица 1.20

Режимы работы таймера/счетчика Т/С1

Номер режима	WGM13	WGM12	WGM11	WGM10	Режим работы Т/С1	Верхний предел	Обновление OCR1x	Установка флага TOV1 на:
0	0	0	0	0	Нормальный	0xFFFF	сразу после записи	МАКС
1	0	0	0	1	8-разр. ШИМ ФК	0x00FF	на вершине счета	нижнем пределе
2	0	0	1	0	9-разр. ШИМ ФК	0x01FF	на вершине счета	нижнем пределе
3	0	0	1	1	10-разр. ШИМ ФК	0x03FF	на вершине счета	нижнем пределе
4	0	1	0	0	СТС	OCR1A	сразу после записи	МАКС
5	0	1	0	1	8-разр. быстрая ШИМ	0x00FF	на вершине счета	вершине счета
6	0	1	1	0	9-разр. быстрая ШИМ	0x01FF	на вершине счета	вершине счета
7	0	1	1	1	10-разр. быстрая ШИМ	0x03FF	на вершине счета	вершине счета
8	1	0	0	0	ШИМ ФЧК	ICR1	на нижнем пределе	нижнем пределе
9	1	0	0	1	ШИМ ФЧК	OCR1A	на нижнем пределе	нижнем пределе
10	1	0	1	0	ШИМ ФК	ICR1	на вершине счета	нижнем пределе
11	1	0	1	1	ШИМ ФК	OCR1A	на вершине счета	нижнем пределе
12	1	1	0	0	СТС	ICR1	сразу после записи	МАКС
13	1	1	0	1	(резерв)	–	–	–
14	1	1	1	0	Быстрая ШИМ	ICR1	на вершине счета	вершине счета
15	1	1	1	1	Быстрая ШИМ	OCR1A	на вершине счета	вершине счета

В нормальном режиме (WGM1 = 0b0000) счетчик работает как суммирующий (инкрементирующий), при этом сброс счетчика не выполняется. Переполнение счетчика происходит при переходе через максимальное 16-разрядное значение (0xFFFF) к нижнему пределу счета (0x0000). В нормальном режиме работы флаг переполнения таймера/счетчика TOV1 будет установлен на том же такте синхронизации, когда **TCNT1** примет нулевое значение.

В режиме СТС (номера режимов 4 и 12) регистр OCR1x используется для задания разрешающей способности счетчика. Если задан режим СТС и значение счетчика **TCNT1** совпадает со значением регистра **OCR1x**, то счетчик об-

нуляется (**TCNT1**=0). Таким образом, **OCR1x** задает вершину счета счетчика, и его разрешающую способность.

Режим быстрой ШИМ (номера режимов 5, 6, 7, 14 и 15) предназначен для генерации ШИМ-импульсов повышенной частоты. В отличие от других режимов работы в нем используется однонаправленная работа счетчика. Счет выполняется в направлении от нижнего к верхнему пределу счета. Если задан неинвертирующий режим выхода, то при совпадении **TCNT1** и **OCR1x** вывод OC1x устанавливается в лог.1, а на верхнем пределе счета сбрасывается в лог.0. Если задан инвертирующий режим, то выход OC1x сбрасывается при совпадении и устанавливается на верхнем пределе счета. За счет однонаправленности счета, рабочая частота для данного режима в два раза выше по сравнению с режимом ШИМ с фазовой коррекцией, где используется двунаправленный счет.

Режим широтно-импульсной модуляции с фазовой коррекцией (номера режимов 1, 2, 10 и 11) предназначен для генерации ШИМ-сигнала с фазовой коррекцией и высокой разрешающей способностью. Режим ШИМ с ФК основан на двунаправленной работе таймера/счетчика. Счетчик циклически выполняет счет в направлении от нижнего предела (0x0000) к верхнему, а затем обратно от верхнего предела к нижнему. Если задан неинвертирующий режим выхода формирователя импульсов, то выход OC1x сбрасывается (или устанавливается) при совпадении значений **TCNT1** и **OCR1x** во время прямого (или обратного) счета. Если задан инвертирующий режим выхода, то, наоборот, во время прямого счета происходит установка, а во время обратного – сброс выхода OC1x. Режим ШИМ с ФЧК в основном аналогичен режиму ШИМ с ФК. Основное отличие состоит в моменте обновления регистра **OCR1x**.

Установки бит COM1x0 и COM1x1 в регистре **TCCR1A** оказывают различное влияние в зависимости от выбранного режима работы. Общим для всех режимов работы является не выполнение каких-либо действий с выводом OC1x при возникновении совпадения, если оба этих бита равны нулю. В режимах без ШИМ (нормальном или СТС) при COM1x1=0 и COM1x0=1 происходит переключение (инвертирование) вывода OC1x. При COM1x1=1 и COM1x0=0 вывод OC1x сбрасывается в лог.0. При COM1x1=1 и COM1x0=1 вывод OC1x устанавливается в лог.1.

Если установлен режим быстрой ШИМ и COM1x1=0 и COM1x0=1, то для режима с номером 15 происходит инвертирование вывода OC1A при совпадении, а выводы OC1B и OC1C отключены. Для всех других режимов быстрой ШИМ (номера 5, 6, 7 и 14) выводы OC1A, OC1B и OC1C отключены. При COM1x1=1 и COM1x0=0 вывод OC1x сбрасывается в лог.0 при совпадении и устанавливается в лог.1 на вершине счета. При COM1x1=1 и COM1x0=1 вывод OC1x устанавливается в лог.1 при совпадении и сбрасывается в лог.0 на вершине счета.

Если установлены режимы ШИМ с ФК и ШИМ с ФЧК (номера 9 или 11) и COM1x1=0 и COM1x0=1, то происходит инвертирование вывода OC1A при

совпадении, а выводы OC1B и OC1C отключены. Для всех других режимов ШИМ с ФК и ШИМ с ФЧК выводы OC1A, OC1B и OC1C отключены.

Важно отметить, что формирование ШИМ-импульсов на выводах OC1x возможно только при установке соответствующих линий порта (линии PB5, PB6 и PB7) на выход. Это означает, что в регистре направления данных **DDRB** соответствующие биты необходимо установить в состояние лог.1.

Биты CS12, CS11 и CS10 в регистре **TCCR1B** формируют двоичный код, определяющий тактовую частоту таймера/счетчика. При CS=0b110 и 0b111 тактирование Т/С1 осуществляется от внешнего источника (см. таблицу 1.21).

Таблица 1.21

Выбор частоты тактирования таймера/счетчика Т/С1

CS12	CS11	CS10	Описание
0	0	0	Нет синхронизации. Таймер-счетчик остановлен.
0	0	1	clkI/O/1 (без предделения)
0	1	0	clkI/O /8 (с предделением на 8)
0	1	1	clkI/O/64 (с предделением на 64)
1	0	0	clkI/O/256 (с предделением на 256)
1	0	1	clkI/O/1024 (с предделением на 1024)
1	1	0	Тактирование от внешнего источника с вывода Tn. Синхронизация по падающему фронту.
1	1	1	Тактирование от внешнего источника с вывода Tn. Синхронизация по нарастающему фронту.

В режимах генерации импульсов без ШИМ в формирователе импульсов результат сравнения может быть установлен непосредственно через бит принудительной установки результата сравнения FOC1x в регистре управления **TCCR1C**. Принудительная установка результата сравнения не приводит к установке флага OCF1x или сбросу/перезагрузке таймера, но влияет на состояние вывода OC1x, который будет устанавливаться, сбрасываться или переключаться (инвертироваться) в зависимости от выбранной установки бит COM1x.

Как уже отмечалось, при возникновении определенных ситуаций, например, при совпадении содержимого регистра **TCNT1** с содержимым одного из регистров сравнения **OCR1x** (x = A, B, C), генерируются запросы на прерывания и устанавливаются соответствующие флаги. Функционирование системы прерываний от всех таймеров/счетчиков осуществляется с участием двух регистров масок прерываний **TIMSK** и **ETIMSK**, а также двух регистров флагов **TIFR** и **ETIFR**. Имена и назначение битов регистров масок прерываний приведены в таблицах 1.22 и 1.23.

Таблица 1.22

Регистр масок прерываний таймеров/счетчиков **TIMSK**

Бит	Символ	Имя и назначение
7	OCIE2	Разрешение прерывания по совпадению T/C2. При установленном бите OCIE2 и установленном бите I регистра **SREG** разрешается прерывание по совпадению состояния T/C2 и содержимого регистра сравнения.
6	TOIE2	Разрешение прерывания по переполнению T/C2.
5	TICIE1	Разрешение прерывания по захвату вывода T/C1.
4	OCIE1A	Разрешение прерывания по совпадению содержимого регистра сравнения A с состоянием T/C1.
3	OCIE1B	Разрешение прерывания по совпадению содержимого регистра сравнения B с состоянием T/C1.
2	TOIE1	Разрешение прерывания по переполнению T/C1.
1	OCIE0	Разрешение прерывания по совпадению регистра сравнения с T/C0.
0	TOIE0	Разрешение прерывания по переполнению T/C0.

Таблица 1.23

Расширенный регистр масок прерываний таймеров/счетчиков **ETIMSK**

Бит	Символ	Имя и назначение
7, 6	-	Зарезервированы
5	TICIE3	Разрешение прерывания по захвату вывода T/C3.
4	OCIE3A	Разрешение прерывания по совпадению содержимого регистра сравнения A с состоянием T/C3.
3	OCIE3B	Разрешение прерывания по совпадению содержимого регистра сравнения B с состоянием T/C3.
2	TOIE3	Разрешение прерывания по переполнению T/C3.
1	OCIE3C	Разрешение прерывания по совпадению содержимого регистра сравнения C с состоянием T/C3.
0	OCIE1C	Разрешение прерывания по совпадению содержимого регистра сравнения C с состоянием T/C1.

В регистрах флагов прерываний по таймерам/счетчикам **TIFR** и **TIFR** устанавливаются флаги запросов разрешенных прерываний от всех таймеров/счетчиков. Имена и назначение этих битов приведены в таблицах 1.24 и 1.25.

Таблица 1.24

Регистр флагов прерываний таймеров/счетчиков **TIFR**

Бит	Символ	Имя и назначение
7	OCF2	Флаг совпадения T/C2 и регистра OCR2.
6	TOV2	Флаг переполнения T/C2.
5	ICF1	Флаг захвата вывода T/C1. Флаг сигнализирует о том, что содержимое T/C1 переслано во входной регистр захвата ICR1.
4	OCF1A	Флаг 1A совпадения выхода. Флаг сигнализирует о том, что содержимое регистра OCR1A совпало с содержимым T/C1.
3	OCF1B	Флаг 1B совпадения выхода. Флаг сигнализирует о том, что содержимое регистра OCR1B совпало с содержимым T/C1.
2	TOV1	Флаг переполнения T/C1. Флаг сигнализирует о переполнении содержимого T/C1.
1	OCF0	Флаг совпадения T/C0 и регистра OCR0.
0	TOV0	Флаг переполнения T/C0.

Таблица 1.25

Расширенный регистр флагов прерываний таймеров/счетчиков **ETIFR**

Бит	Символ	Имя и назначение
7	-	Зарезервировано
6	-	Зарезервировано
5	ICF3	Флаг захвата вывода T/C3. Флаг сигнализирует о том, что содержимое T/C3 переслано во входной регистр захвата ICR3.
4	OCF3A	Флаг 3A совпадения выхода. Флаг сигнализирует о том, что содержимое регистра OCR3A совпало с содержимым T/C3.
3	OCF3B	Флаг 3B совпадения выхода. Флаг сигнализирует о том, что содержимое регистра OCR3B совпало с содержимым T/C3.
2	TOV3	Флаг переполнения T/C3. Флаг сигнализирует о переполнении содержимого T/C3.
1	OCF3C	Флаг 3C совпадения выхода. Флаг сигнализирует о том, что содержимое регистра OCR3C совпало с содержимым T/C3.
0	OCF1C	Флаг 1C совпадения выхода. Флаг сигнализирует о том, что содержимое регистра OCR1C совпало с содержимым T/C1.

Отклик на выполнение всех разрешенных прерываний составляет минимум четыре тактовых цикла. В течение четырех тактовых циклов после установки флага прерывания выполняется переход по адресу вектора прерывания для выполнения подпрограммы обработки прерывания. В течение этих четырех циклов содержимое счетчика команд (2 байта) опускается в стек и указатель стека декрементируется на 2. Вектор указывает переход в подпрограмму обработки прерывания и этот переход занимает три тактовых цикла. Если прерывание возникает во время выполнения многоцикловой команды, то команда завершается до начала обслуживания прерывания.

Возврат из подпрограммы обработки прерывания (как и вызов самой подпрограммы) занимает четыре тактовых цикла. В течение этих четырех циклов состояние счетчика команд (2 байта) извлекается из стека и указатель стека инкрементируется на 2. Когда микроконтроллер выходит из прерывания, он всегда возвращается в основную программу и выполняет еще одну команду прежде, чем начать обслуживание какого-либо отложенного прерывания.

Для прерываний, запускаемых статическими событиями (например, при совпадении содержимого регистра сравнения 1A с состоянием T/C1) флаг прерывания устанавливается в момент наступления события. Если флаг очищен, но условия возникновения прерывания продолжают существовать, флаг не будет устанавливаться до тех пор, пока это событие не наступит вновь. Все флаги автоматически сбрасываются при переходе на соответствующий вектор прерывания. Альтернативно флаги можно сбросить путем записи в них лог. 1.

1.6.2. Аналого-цифровой преобразователь

Микроконтроллеры ATmega128 оснащены 10-разрядным аналого-цифровым преобразователем последовательного приближения, который имеет следующие технические характеристики:

- разрешение – 10 разрядов;
- абсолютная погрешность – 2 единицы младшего разряда;
- время преобразования – 65…260 мкс;
- 8 мультиплексируемых однополярных каналов входа;
- 7 дифференциальных входных каналов;
- 2 дифференциальных входных канала с усилением на 10 и 200;
- входное сопротивление аналогового входа – 100 МОм;
- наличие внутреннего источника опорного напряжения 2,56 В;
- режимы циклического и однократного преобразования;
- формирование прерывания IRQ по завершению преобразования;
- устройство подавления шумов в режиме Sleep.

АЦП подсоединен к 8-канальному аналоговому мультиплексору, позволяющему использовать любой вывод порта F в качестве аналогового входа АЦП. АЦП содержит устройство выборки и хранения (УВХ), удерживающее входное напряжение АЦП во время преобразования на неизменном уровне. Для питания АЦП используются два отдельных вывода: AVCC и AGND. Напряжение AVCC не должно отличаться от напряжения питания микроконтроллера Vcc более чем на 0,3 В. Вывод AGND может быть подключен к общей шине GND. Внешнее напряжение сравнения подается на вывод AREF и должно быть в диапазоне от 0 до AVCC. В качестве внутреннего опорного напряжения может выступать напряжение от внутреннего ИОН на 2.56 В или напряжение AVCC. Если требуется использование внешнего ИОН, то он должен быть подключен к выводу AREF с подключением к этому выводу блокировочного конденсатора для улучшения шумовых характеристик.

Функциональная схема АЦП представлена на рис. 1.8. Функционирование АЦП обеспечивают три регистра: регистр данных **ADC**, регистр выбора мультиплексора **ADMUX** и регистр А управления и статуса **ADCSRA**. Регистр данных **ADC** по существу состоит из двух регистров **ADCL** и **ADCH**, в которых хранятся младший и старший байты 10-разрядного результата преобразования, причем в **ADCH** используются только два младших разряда. При работе в циклическом режиме чтение результата преобразования необходимо начинать с младшего байта. Регистр выбора мультиплексора **ADMUX** представляет собой 8-разрядный регистр, в трех младших разрядах которого устанавливаются биты MUX2, MUX1 и MUX0, определяющие двоичный код номера используемого канала мультиплексора. В регистре А управления и статуса **ADCSRA** устанавливаются управляющие биты, позволяющие изменять режимы работы АЦП. Имена и назначения битов в регистрах **ADCSRA** и **ADMUX** приведены в таблицах 1.26 и 1.27.

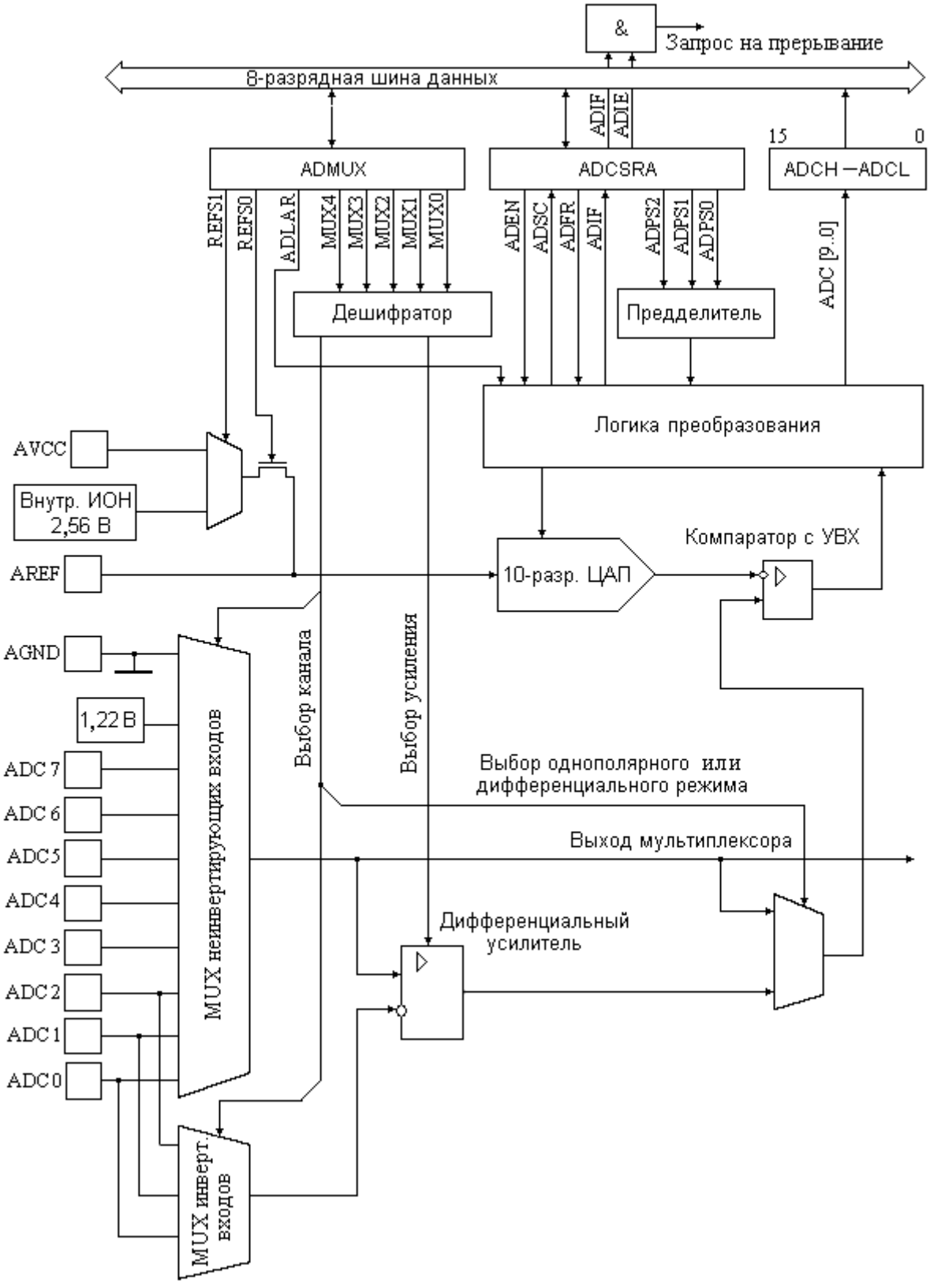

Рис. 1.8. Функциональная схема аналого-цифрового преобразователя

Аналого-цифровой преобразователь может работать в двух режимах: режиме однократного преобразования и режиме автоматического перезапуска (режиме циклического преобразования). В первом случае каждое преобразова-

ние инициируется пользователем, во втором – АЦП осуществляет выборку и обновление содержимого регистра данных непрерывно. Выбор режима производится битом ADFR регистра **ADCSRA**. Работа АЦП разрешается установкой бита ADEN в состояние лог.1. Первому преобразованию, начинающемуся после разрешения АЦП, предшествует пустое инициализирующее преобразование. Поэтому первое преобразование будет занимать 27 тактовых циклов вместо обычных 14.

Таблица 1.26

Регистр управления и состояния **ADCSRA**

Бит	Символ	Имя и назначение
7	ADEN	Разрешение АЦП. При установке бита в состояние лог.1 разрешается работа АЦП.
6	ADSC	Запуск преобразования АЦП. Установка бита в лог.1 инициирует процесс аналого-цифрового преобразования. По завершении преобразования бит ADSC сбрасывается в лог.0. В режиме одиночного преобразования установка бита в лог.1 инициирует старт каждого преобразования. По завершении преобразования бит ADSC сбрасывается в лог.0. В режиме автоматического перезапуска установкой этого бита инициируется только первое преобразование, а все остальные выполняются автоматически.
5	ADFR	Выбор режима автоматического перезапуска АЦП. При ADFR =1 АЦП будет работать в циклическом режиме, т.е. выборки и обращения к регистрам данных происходят одно за другим. Очистка бита прекращает циклический режим.
4	ADIF	Флаг прерывания АЦП. Устанавливается в лог.1 по завершении преобразования при условии, что данное прерывание разрешено. Флаг ADIF сбрасывается аппаратно при переходе на соответствующий вектор прерывания. Альтернативно флаг ADIF сбрасывается путем записи в него лог.1.
3	ADIE	Разрешение прерывания, если ADIE = 1 (при условии, что установлен бит I в регистре **SREG**). Запрет прерывания, если ADIE = 0.
2 1 0	ADPS2 ADPS1 ADPS0	Выбор коэффициентов предварительного деления. Биты ADPS2, ADPS1 и ADPS0 определяют 8 ступеней деления частоты XTAL (2, 2, 4, 8, 16, 32, 64, 128)

Таблица 1.27

Регистр управления мультиплексором **ADMUX**

Бит	Символ	Имя и назначение
7 6	REFS1 REFS0	Биты REFS1 и REFS0 определяют выбор источника опорного напряжения согласно схеме: REFS1=0 REFS0=0 AREF, внутренний ИОН отключен REFS1=0 REFS0=1 AVCC с внешним конденсатором на выводе AREF REFS1=1 REFS0=0 Зарезервировано REFS1=1 REFS0=1 Внутренний ИОН с напряжением 2.56 В с внешним конденсатором на выводе AREF

Окончание таблицы 1.27

Бит	Символ	Имя и назначение
5	ADLAR	Бит управления представлением результата преобразования. Если ADLAR = 1, то результат преобразования будет иметь левосторонний формат (выравнивание слева), при ADLAR = 0 формат правосторонний (выравнивание справа).
4	MUX4	Биты MUX4… MUX0 формируют двоичный код, определяющий выбор канала мультиплексора и коэффициента усиления. Данные биты определяют, какие из аналоговых входов подключаются к АЦП. Кроме того, с их помощью можно выбрать коэффициент усиления для дифференциальных каналов
3	MUX3	
2	MUX2	
1	MUX1	
0	MUX0	

Канал аналогового ввода и каскад дифференциального усиления выбираются путем записи бит MUX0...MUX4 в регистре **ADMUX**. В качестве однополярного аналогового входа АЦП может быть выбран один из входов ADC0…ADC7, а также GND и выход фиксированного источника опорного напряжения 1,22 В. В режиме дифференциального ввода предусмотрена возможность выбора инвертирующих и неинвертирующих входов к дифференциальному усилителю. Если выбран дифференциальный режим аналогового ввода, то дифференциальный усилитель будет усиливать разность напряжений между выбранной парой входов на заданный коэффициент усиления. Усиленное таким образом значение поступает на аналоговый вход АЦП. Если выбирается однополярный режим аналогового ввода, то каскад усиления пропускается.

Непосредственно сам процесс преобразования начинается с установки в состояние лог.1 бита начала преобразования ADSC. Этот бит находится в состоянии лог.1 в течение всего цикла преобразования и сбрасывается аппаратно по завершении преобразования. Если в процессе выполнения преобразования выполняется смена канала мультиплексора, то АЦП вначале закончит текущее преобразование и лишь потом выполнит переход к другому каналу.

Аналого-цифровой преобразователь относится к АЦП последовательного приближения и работает с тактовой частотой от 50 до 200 кГц. В режиме циклического преобразования для преобразования необходимо 14 тактовых циклов, т.е. преобразование осуществляется за время от 70 до 280 мкс. В режиме однократного преобразования преобразование выполняется за 15 тактовых циклов. Если тактовая частота выйдет за указанные пределы, то правильность результата не гарантируется. Биты ADPS2, ADPS1 и ADPS0 используются для задания необходимой тактовой частоты АЦП в случае, если частота микроконтроллера XTAL превышает 100 кГц.

1.6.3. Аналоговый компаратор

Аналоговый компаратор сравнивает уровни напряжений на неинвертирующем входе AIN0 (вывод 2 порта E) и инвертирующем входе AIN1 (вывод 3 порта E). Если напряжение на неинвертирующем входе AIN0 превышает напряжение на инвертирующем входе AIN1, выход аналогового компаратора ус-

танавливается в состояние лог.1, при обратном соотношении сравниваемых напряжений выход компаратора сбрасывается в состояние лог.0. Функциональная схема компаратора и связанной с ним логики представлена на рис. 1.9.

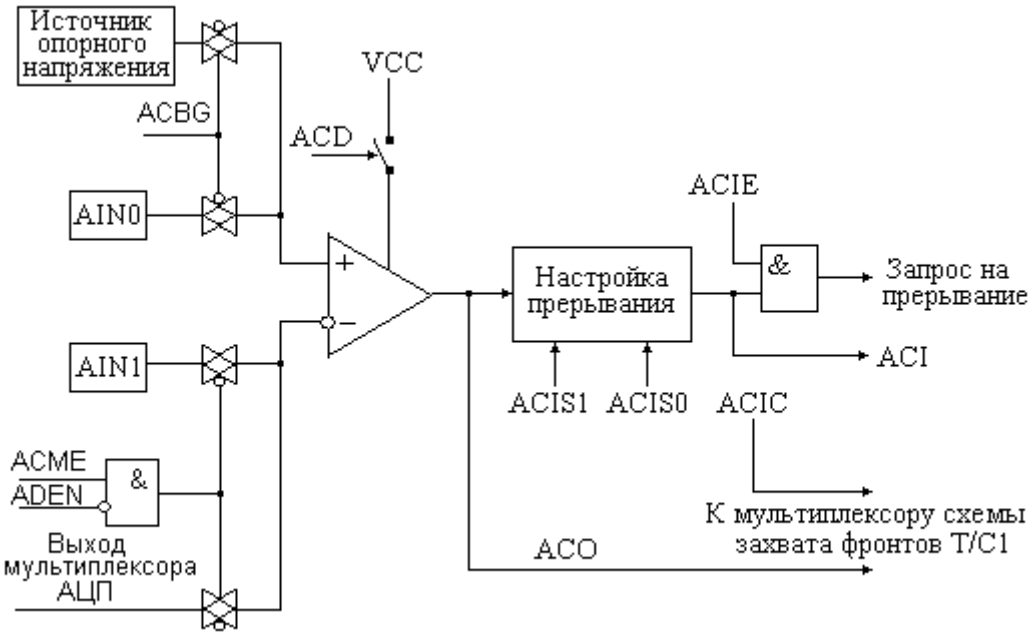

Рис. 1.9. Функциональная схема аналогового компаратора

Переключение выхода компаратора может вызвать запрос прерывания, если данное прерывание разрешено. Тип изменения сигнала на выходе компаратора, вызывающего прерывания устанавливается специальными управляющими битами ACIS1 и ACIS0 в регистре статуса и управления аналогового компаратора **ACSR**. Можно задать формирование запроса прерывания по нарастающему или падающему фронту, а также по переключению выхода компаратора.

Сигнал с выхода компаратора ACO может быть использован для реализации функции захвата входа таймера/счетчика 1. В этом случае выход аналогового компаратора подсоединяется непосредственно к входной цепи логики захвата входа T/C1.

Работой аналогового компаратора управляет регистр статуса и управления **ACSR**, в котором устанавливаются биты управления компаратором, а также аппаратно устанавливаются биты, характеризующие состояние компаратора. Имена и назначения этих битов приведены в таблице 1.28.

Таблица 1.28

Регистр статуса и управления компаратора **ACSR**

Бит	Символ	Имя и назначение
7	ACD	Отключение аналогового компаратора. При установленном в состояние лог.1 бите ACD работа аналогового компаратора запрещена.
6	ACBG	Бит подключения источника опорного напряжения к аналоговому компаратору. При ACBG = 1 к неинвертирующему входу компаратора подключается источник опорного напряжения. При ACBG = 0 неинвертирующий вход связан с выводом AIN0.

Окончание таблицы 1.28

Бит	Символ	Имя и назначение
5	ACO	Выход аналогового компаратора. Бит ACO связан непосредственно с выходом компаратора.
4	ACI	Флаг прерывания по аналоговому компаратору. Флаг устанавливается, если данное прерывание разрешено (биты разрешения ACIE и глобального разрешения I в регистре **SREG** установлены в состояние лог.1). Флаг сбрасывается аппаратно после выполнения процедуры обработки прерывания.
3	ACIE	Разрешение прерывания по аналоговому компаратору. При ACIE = 1 прерывание разрешено, при ACIE = 0 – запрещено.
2	ACIC	Разрешение входа захвата аналогового компаратора. Установленный в лог.1 бит ACIC разрешает срабатывание функции захвата входа T/C1 по переключению аналогового компаратора.
1 0	ACIS1 ACIS0	Выбор режима прерывания по аналоговому компаратору: ACIS1=0 ACIS0=0 Прерывание по переключению выхода ACIS1=0 ACIS0=1 Не используется ACIS1=1 ACIS0=0 Прерывание по падающему фронту на выходе ACIS1=1 ACIS0=1 Прерывание по нарастающему фронту на выходе

Примечание. При изменении состояния битов ACIS1 и ACIS0 прерывание по аналоговому компаратору должно быть запрещено очисткой бита разрешения прерывания в регистре **ACSR**. Иначе при изменении состояния битов может произойти прерывание.

В микроконтроллере имеется возможность использовать в качестве неинвертирующих входов выводы ADC0… ADC7 (выводы порта F). Для реализации такой возможности используется мультиплексор АЦП и, следовательно, в этом случае сам АЦП должен быть отключен. Если установлен бит разрешения подключения мультиплексора к аналоговому компаратору (бит ACME в регистре **SFIOR**) и выключен АЦП (ADEN = 0 в регистре **ADCSRA**), то двоичный код MUX2..0 регистра **ADMUX** определяет, какой вывод микроконтроллера подключен к неинвертирующему входу аналогового компаратора. Если ACME сброшен или установлен ADEN, то в качестве неинвертирующего входа аналогового компаратора используется только вывод AIN1.

1.6.4. Последовательный периферийный интерфейс

Последовательный интерфейс SPI (Serial Peripheral Interface) обеспечивает высокоскоростной синхронный обмен данными между микроконтроллером ATmega128 и периферийными устройствами или между несколькими микроконтроллерами семейства AVR. Основные технические характеристики SPI интерфейса следующие:

- Полнодуплексный 3-проводной синхронный обмен данными.
- Режим работы ведущий или ведомый.
- Передача первым младшего или старшего бита.
- Четыре программируемые скорости обмена данными.

• Флаг прерывания по окончании передачи.

• Активация из Idle режима (только в режиме ведомого).

Соединение между ведущим (Master) и ведомым (Slave) микроконтроллерами, использующими SPI интерфейс, показано на рис. 1.10. Система состоит из двух сдвиговых регистров и генератора ведущей синхронизации. Ведущий МК инициирует сеанс связи подачей низкого уровня на вход SS. Оба МК (ведущий и ведомый) подготавливают данные к передаче в своем сдвиговом регистре, при этом на стороне ведущего генерируются также импульсы синхронизации на линии SCK. По линии MOSI всегда осуществляется передача данных от ведущего к ведомому, а по MISO, наоборот, от ведомого к ведущему. После сдвига одного байта тактовый генератор SPI останавливается, устанавливая флаг окончания передачи SPIF в регистре статуса SPI. Если в регистре управления SPI был установлен бит разрешения прерывания SPI (бит SPIE), то произойдет запрос прерывания. По окончании передачи каждого пакета данных ведущее SPI-устройство должно засинхронизировать ведомое путем подачи высокого уровня на линию SS (выбор подчиненного интерфейса).

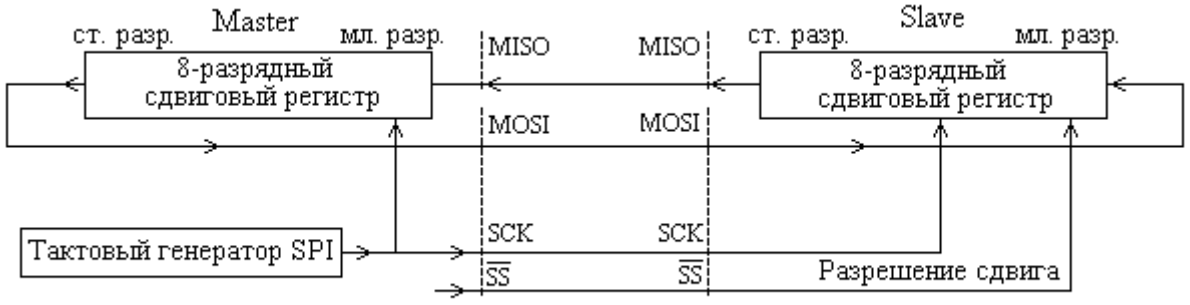

Рис. 1.10. Межсоединения ведущего (Master) и ведомого (Slave) микроконтроллерами, использующими SPI интерфейс

Если МК настроен как ведущий (Master), то управление линией SS происходит не автоматически. Данная операция должна быть выполнена программно перед началом сеанса связи. После этого, запись в регистр данных SPI инициирует генерацию синхронизации и аппаратный сдвиг восьми разрядов в ведомое устройство. По окончании сдвига одного байта генератор синхронизации SPI останавливается, при этом устанавливая флаг окончания передачи SPIF. Если установлен бит SPIE в регистре **SPCR**, то разрешается прерывание SPI и по окончании передачи байта будет генерироваться запрос на прерывание. Ведущий МК может продолжить сдвигать следующий байт, если записать его в регистр **SPDR**, или подать сигнал окончания пакета путем установки низкого уровня на линии SS. Последний принятый байт сохраняется в буферном регистре.

В режиме ведомого интерфейс SPI находится в состоянии ожидания, в котором MISO переводится в третье состояние, до тех пор, пока на выводе SS присутствует высокий уровень. В этом состоянии программа может обновлять содержимое регистра данных **SPDR**, но при этом входящие импульсы синхронизации не сдвигают данные до подачи низкого уровня на вывод SS. После того

как один байт был полностью сдвинут, устанавливается флаг окончания передачи SPIF. Если установлен бит разрешения прерывания SPIE в регистре **SPCR**, то установка флага SPIF приводит к генерации запроса на прерывание. Ведомый МК может продолжать размещать новые данные для передачи в регистр **SPDR** перед чтением входящих данных. Последний принятый байт хранится в буферном регистре.

Работой последовательного периферийного интерфейса управляют три регистра: регистр данных **SPDR** регистр управления **SPCR** и регистр статуса **SPSR**. Регистр данных **SPDR** представляет собой 8-разрядный регистр с возможностью чтения/записи и предназначен для пересылки данных между регистровым файлом и сдвиговым регистром последовательного периферийного интерфейса SPI. Запись байта в регистр **SPDR** инициирует передачу данных, считывание регистра приводит к чтению сдвигового регистра приема.

Регистр управления **SPCR** предназначен для хранения программно доступных битов, управляющих работой SPI. Символические имена и назначение битов регистра **SPCR** приведены в 1.29.

Таблица 1.29

Регистр управления **SPCR**

Бит	Символ	Имя и назначение
7	SPIE	Разрешение прерывания SPI.
6	SPE	Разрешение SPI. Установка бита SPE в состояние лог.1 разрешает работу SPI независимо от того, в каком режиме он будет работать.
5	DORD	Порядок передачи данных. При DORD = 1 передача слова данных происходит младшим знаковым разрядом (LSB) вперед. При DORD = 0 первым передается старший значащий разряд (MSB).
4	MSTR	Выбор режима ведущий/ведомый. При MSTR = 1 МК работает в режиме ведущего. Если SS настроен как вход и к нему приложен низкий уровень, когда MSTR = 1, то бит MSTR автоматически сбрасывается и устанавливается флаг SPIF в регистре **SPSR**.
3	CROL	Полярность тактового сигнала. При CPOL = 1 вывод SCK находится на высоком уровне, при CPOL = 0 – на низком.
2	CPHA	Фаза тактового сигнала. Данный бит определяет, по какому фронту SCK происходит выборка данных: по переднему или заднему.
1 0	SPR1 SPR0	Выбор частоты тактового сигнала. Биты SPR1 и SPR0 определяют двоичный код выбора частоты SCK согласно схеме: SPR1=0 SPR0=0 частота равна $f_0/4$ SPR1=0 SPR0=1 частота равна $f_0/16$ SPR1=1 SPR0=0 частота равна $f_0/64$ SPR1=1 SPR0=1 частота равна $f_0/128$, где f_0 – частота генератора микроконтроллера.

Частоту тактового сигнала SCK можно увеличить в 2 раза, установив в регистре статуса **SPSR** бит удвоения скорости SPI2X. При SPI2X = 0 частота определяется только битами SPR1 и SPR0.

В регистре статуса **SPSR**, кроме бита удвоения скорости SPI2X (нулевой бит), используются еще два бита: бит 7 – флаг прерывания SPIF и бит 6 – флаг

ошибки при записи WCOL. Флаг прерывания SPIF устанавливается в лог.1 по завершении обмена последовательными данными. Если при этом установлен бит SPIE в регистре **SPCR** и разрешено глобальное прерывание, то генерируется сигнал прерывания. SPIF сбрасывается аппаратно при переходе на соответствующий вектор прерывания. Альтернативно, бит SPIF сбрасывается при первом чтении регистра статуса SPI с установленным флагом SPIF, а также во время доступа к регистру данных SPI (регистру **SPDR**).

Бит WCOL устанавливается в состояние лог.1, если в процессе передачи данных выполнялась запись в регистр данных. Запись в регистр данных (как и чтение его содержимого), выполненная во время пересылки данных, могут привести к неверному результату. Бит WCOL аппаратно сбрасывается в состояние лог.0 при первом считывании состояния регистра статуса SPSR (с установленным битом WCOL) с последующим обращением к регистру **SPDR**.

1.6.5. Универсальный синхронный и асинхронный приемопередатчик

Универсальный синхронный и асинхронный приемопередатчик (USART – Universal Synchronous and Asynchronous Receiver/Transmitter) предназначен для реализации обмена данными в последовательном коде с микроконтроллерами или иными микропроцессорными устройствами. ATmega128 содержит два совершенно одинаковых по функциям приемопередатчика: USART0 и USART1. Отличительными особенностями USART являются:

- полнодуплексная работа (раздельные регистры приема и передачи);
- асинхронная или синхронная работа;
- ведущее или ведомое тактирование связи в синхронном режиме работы;
- поддержка формата передаваемых данных с 5, 6, 7, 8 или 9 битами данных и 1 или 2 стоп-битами;
- аппаратная генерация и проверка бита паритета (четность/нечетность);
- три раздельных прерывания: по завершении передачи, освобождении регистра передаваемых данных и завершении приема;
- режим удвоения скорости связи в асинхронном режиме;
- режим многопроцессорной связи.

Каждый из USART состоит из трех блоков: тактового генератора, передатчика и приемника. Логика тактового генератора состоит из логики синхронизации, связанной с внешним тактовым входом (используется в режиме ведомого) и генератора скорости связи. Вывод XCK (синхронизация передачи) используется только в режиме синхронной передачи. Передатчик состоит из одного буфера записи, последовательного сдвигового регистра, генератора паритета и управляющей логики, которая поддерживает различные форматы последовательной посылки. Буфер записи позволяет непрерывно передавать данные без каких-либо задержек между передачей посылок. Приемник является более сложным блоком USART, в его состав входят модули обнаружения данных и синхронизации. Модули обнаружения необходимы для асинхронного приема данных. Помимо модулей обнаружения в приемник входит устройство провер-

ки паритета, сдвиговый регистр и двухуровневый приемный буфер. Приемник поддерживает те же последовательные форматы, что и передатчик, и может определить ошибку в посылке (кадре), переполнение данных и ошибку паритета.

Работой USART0 и USART1 управляют три регистра статуса и управления **UCSRxA**, **UCSRxB** и **UCSRxC** (x = 0, 1), имена и назначения битов которых приведены в таблицах 1.30 – 1.32. Кроме этого имеется регистр данных **UDRx**, в который записываются данные при передаче и считываются при приеме, а также два регистра **UBRRxL** и **UBRRxH**, задающие скорость передачи USART.

Таблица 1.30

Регистр А управления и статуса **UCSRxA** (x = 0, 1)

Бит	Символ	Имя и назначение
7	RXCx	Флаг завершения приема. Бит RXCx устанавливается, если в приемном буфере содержатся несчитанные данные, и сбрасывается, когда приемный буфер свободен (не содержит несчитанных данных). Если приемник отключается, то приемный буфер сбрасывается и, следовательно, флаг RXCx принимает нулевое значение. Флаг RXCx может использоваться для генерации прерывания по завершению приема (см. описание бита RXCIEx).
6	TXCx	Флаг завершения передачи. Бит TXCx устанавливается, если вся посылка из сдвигового регистра передатчика полностью передана и в передающем буфере UDRx нет новых данных для передачи. Флаг TXCx автоматически сбрасывается при переходе на вектор прерывания по завершению передачи или сбрасывается программно путем записи в него лог.1. Флаг TXCx может служить источником для генерации прерывания по завершению передачи (см. описание бита TXCIEx).
5	UDREx	Флаг освобождения регистра данных. Бит UDREx индицирует о готовности приемного буфера UDRx к приему новых данных. Если UDREx=1, то буфер свободен и, следовательно, готов к записи. Флаг UDREx может служить источником для генерации прерывания по освобождению регистра данных. Бит UDREx устанавливается после сброса, индицируя о готовности передатчика.
4	FEx	Флаг ошибки посылки. Бит FEx устанавливается в лог.1, если при приеме посылки была определена ошибка в структуре посылки. При записи в регистр **UCSRxA** в позиции данного бита необходимо записать лог. 0.
3	DORx	Флаг переполнения данных. Бит DORx устанавливается в лог.1 при обнаружении условий переполнения, то есть когда символ, уже находящийся в регистре **UDRx**, не считан перед пересылкой нового символа из сдвигового регистра приема. При записи в регистр **UCSRxA** в позиции данного бита необходимо записать лог.0.
2	UPEx	Флаг ошибки паритета. Бит UPEx устанавливается в лог.1, если следующая посылка в приемном буфере характеризуется ошибкой паритета при условии, что во время приема этой посылки был разрешен контроль паритета (UPMx1 = 1). При записи в регистр **UCSRxA** в позиции данного бита необходимо записать лог. 0.
1	U2Xx	Бит удвоения скорости связи. Оказывает влияние только в асинхронном режиме связи. В синхронном режиме в данный бит необходимо записать лог. 0. При U2Xx = 1 скорость связи удваивается.
0	MPCMx	Бит установки режима многопроцессорной связи.

Таблица 1.31

Регистр B управления и статуса **UCSRxB** (x = 0, 1)

Бит	Символ	Имя и назначение
7	RXCIEx	Разрешение прерывания по завершению приема. При RXCIEx = 1 разрешено прерывание по флагу RXCx (при условии, что в регистре **SREG** установлен флаг I общего разрешения прерываний).
6	TXCIEx	Разрешение прерывания по завершению передачи. При TXCIEx = 1 разрешено прерывание по флагу TXCx.
5	UDRIEx	Разрешение прерывания по освобождению регистра данных. При UDRIEx = 1 разрешено прерывание по флагу UDREn.
4	RXENx	Бит разрешения работы приемника.
3	TXENx	Бит разрешения работы передатчика.
2	UCSZx2	Бит установки формата данных. Бит UCSZx2 совместно с битами UCSZx1..0 в регистре **UCSRxC** задают количество бит данных в посылке.
1	RXB8x	RXB8x содержит значение 9-го бита принятой посылки с 9-битным форматом. Данный бит необходимо считать прежде, чем будут считаны младшие 8 бит из регистра **UDRx**.
0	TXB8x	TXB8n содержит значение 9-го бита данных для передачи посылки с 9-битным форматом. Данный бит необходимо записать перед тем, как будут записаны младшие разряды данных в **UDRx**.

Таблица 1.32

Регистр C управления и статуса **UCSRxC** (x = 0, 1)

<table>
<tr><th>Бит</th><th>Символ</th><th>Имя и назначение</th></tr>
<tr><td>7</td><td>-</td><td>Резервный бит.</td></tr>
<tr><td>6</td><td>UMSELx</td><td>Выбор режима связи: синхронный (UMSELx = 1) или асинхронный (UMSELx = 0).</td></tr>
<tr><td>5
4</td><td>UPMx1
UPMx0</td><td>Биты UPMx1 и UPMx0 разрешают и устанавливают тип генерируемого и контролируемого паритета. Бит паритета в посылке устанавливается согласно схеме:
UPMx1=0 UPMx0=0 режим паритета отключен
UPMx1=0 UPMx0=1 зарезервировано
UPMx1=1 UPMx0=0 контроль по четности
UPMx1=1 UPMx0=1 контроль по нечетности</td></tr>
<tr><td>3</td><td>USBSx</td><td>Бит выбора числа стоп-бит. Бит USBSx определяет количество стоповых бит, которое вставляет передатчик при генерации посылки: 1 (при USBSx = 0) или 2 (при USBSx = 1)</td></tr>
<tr><td>2
1</td><td>UCSZx1
UCSZx0</td><td>Биты UCSZx1 и UCSZx0 вместе с UCSZx2 в регистре UCSRxB задают количество бит данных в посылке, как для приемника, так и для передатчика согласно схеме:
<table>
<tr><th>UCSZx2</th><th>UCSZx1</th><th>UCSZx0</th><th>Формат данных</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>5 бит</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>6 бит</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>7 бит</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>8 бит</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Зарезервировано</td></tr>
</table></td></tr>
</table>

Окончание таблицы 1.32

<table>
<tr><td>Бит</td><td>Символ</td><td>Имя и назначение</td></tr>
<tr><td>0</td><td>UCPOLx</td><td>
<table>
<tr><td>1</td><td>0</td><td>1</td><td>Зарезервировано</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Зарезервировано</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>9 бит</td></tr>
</table>
Полярность синхронизации. Бит UCPOLx используется только в синхронном режиме. Если используется асинхронный режим, то в данный бит необходимо записать лог. 0.
</td></tr>
</table>

USART поддерживает четыре режима работы синхронизации: нормальная асинхронная, асинхронная с удвоением скорости, ведущая синхронная и ведомая синхронная. Бит UMSELx в регистре С управления и статуса **UCSRxC** позволяют выбрать асинхронную или синхронную работу. Удвоение скорости (только в асинхронном режиме) управляется битом U2Xx в регистре **UCSRxA**. При использовании синхронного режима (UMSELx = 1) соответствующий бит в регистре направления данных для вывода XCK задает, будет ли синхронизация внутренней (ведущий режим) или внешней (ведомый режим). Вывод XCK активен только при использовании синхронного режима.

Последовательная посылка состоит из бит данных, бит синхронизации (старт и стоп-биты), а также опционального бита паритета для поиска ошибок. USART поддерживает следующие форматы посылок:

1 старт-бит (лог.0);

5, 6, 7, 8 или 9 бит данных (без паритета, с битом четности или с битом нечетности);

1 или 2 стоп-бита (лог.1).

Формат посылки, который используется USART, задается битами UCSZx2..0, UPMx1..0 и USBSx в регистрах **UCSRxB** и **UCSRxC**. Посылка начинается со старт-бита, а за ним следует передача бит данных, начиная с самого младшего разряда. Если разрешена функция контроля паритета, то сразу после бит данных передается бит паритета. Бит паритета позволяет проконтролировать правильность передачи данных по линии связи. Он вычисляется путем выполнения логической операции исключающего ИЛИ над всеми битами данных в посылке. Если используется нечетность, то результат этой операции инвертируется. Последними передаются стоп-биты. После завершения передачи посылки имеется возможность либо передавать следующую посылку, либо перевести линию связи в состояние ожидания (высокий уровень).

Передача данных по последовательному каналу инициируется записью передаваемых данных в регистр данных **UDRx**. После этого данные из регистра **UDRx** пересылаются в сдвиговый регистр передачи. Если новый символ записывается в **UDRx** после того, как из сдвигового регистра был выведен стоповый бит предшествующего символа, то сдвиговый регистр загружается немедленно. Если же новый символ записан до того, как из сдвигового регистра был выведен стоповый бит предшествующего символа, то загрузка сдвигового регистра

осуществляется лишь после вывода стопового бита предшествующего символа. В действительности регистр **UDRx** является двумя физически разделенными регистрами: регистром передачи данных и регистром приема данных, использующими один и тот же адрес в пространстве памяти I/O. При записи данных в регистр происходит обращение к регистру передачи данных USART, а при чтении – обращение к регистру приема данных USART.

После пересылки данных в сдвиговый регистр тактом генератора они сдвигаются на вывод TXDx микроконтроллера. Скорость передачи посылки определяется битами в регистре скорости связи **UBRRx**, а также битом U2Xx в регистре **UCSRxA**. Ход процесса передачи данных можно проконтролировать, анализируя состояние флага UDREx (регистр **UDRx** пуст) и флага TXCx (флаг завершения передачи).

При поступлении потока данных на вход RXDx микроконтроллера специальная логика восстановления данных анализирует поступающие данные на предмет обнаружения старт-бита. После его обнаружения следующие за ним биты считаются информационными и, если значение стопового бита равно лог.1, то из сдвигового регистра данные пересылаются в регистр **UDRx**, откуда они могут быть программно считаны. Если же значение стоп-бита оказалось равным лог.0 (в результате ошибочного детектирования старт-бита), то данные из сдвигового регистра все равно будут переданы в регистр **UDRx**, но при этом установится флаг FEx (ошибка кадра). Для обнаружения ошибки кадра пользователь перед чтением регистра **UDRx** должен проверить состояние флага FEx. Флаг FEx очищается при считывании данных из регистра **UDRx**.

2. ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

На начальном этапе своего развития микроконтроллеры программировались исключительно на том или ином языке ассемблера, ориентированного на конкретное устройство. По сути, такие языки представляли собой символьные мнемоники соответствующих машинных кодов, а перевод мнемоники в машинный код выполнялся транслятором. Это позволяло создавать оптимальные по объему и быстродействию программы, но требовало знаний всех особенностей архитектуры конкретного типа микроконтроллера и логики его работы. В процессе развития микроконтроллеров их объем памяти и быстродействие резко увеличились, а решаемые задачи значительно усложнились. Это привело к тому, что при разработке программного обеспечения для микроконтроллеров стали в основном использовать языки высокого уровня, в частности, язык Си.

В данном разделе приводятся краткие сведения о языке Си и компиляторе Image Craft, достаточные для того, чтобы приобрести навыки программирования микроконтроллеров и разрабатывать микропроцессорные устройства, требующие несложного программного обеспечения. Для более углубленного изучения языка Си необходимо обратиться к специальной литературе.

2.1. Краткие сведения из языка программирования Си

2.1.1. Элементы языка Си

К основным элементам языка Си относятся идентификаторы, ключевые слова, операторы и комментарии. *Идентификатором* называется последовательность букв и цифр, а также специального символа подчеркивания «_». Буквы в верхнем и нижнем регистрах считаются различными. Идентификатор не может начинаться с цифры и не может иметь пробелов. Идентификатор используется для именования переменных, констант, функций, типов данных и т.д., например, переменной X, которой присвоено значение 2:

```
X = 2;
```

Ключевые слова – это зарезервированное слово, которое наделено четко определенным смыслом. Ключевые слова не могут быть использованы в качестве идентификаторов. Их можно использовать только в соответствии со значением, известным компилятору языка Си. Список ключевых слов:

asm	auto	bit	bool	break	case	char	const	continue	default
defined	do	double	else	enum	extern	false	float	for	goto
if	inline	int	long	register	return	short	signed	sizeof	static
struct	switch	true	tupedef	union	unsigned	void	volatile	while	

Оператор – это символ или текстовая команда, которые указывают компилятору, какие действия (операции) надо выполнить над операндами. В качестве операндов выступают константы, переменные, символы, строки, функции и т.д. Комбинация символов операций и операндов, результатом которой явля-

ется определенное значение, называется выражением. Символы операций определяют действия, которые должны быть выполнены над операндами. Значение выражения зависит от расположения знаков операций и круглых скобок в выражении, а также от приоритета выполнения операций. Например, переменная X равна удвоенной сумме переменных a и b:

```
X = (a + b)*2;
```

Комментарий – это некоторый поясняющий текст, который при компиляции программы не учитывается. Комментарии бывают многострочными (начинаются с символов /* и заканчиваются символами */) и однострочными (начинаются с символов //). В последнем случае комментарием считается вся часть строки, расположенная справа от символов //.

2.1.2. Типы данных

Тип данных определяет диапазон допустимых значений и пространство, отводимое в памяти для переменных, констант и результатов, возвращаемых функциями. Основные стандартные типы данных приведены в таблице 2.1.

Таблица 2.1

Основные стандартные типы данных языка Си

Тип	Название	Размер в битах	Диапазон значений
bit	Бит	1	0, 1
char	Символ	8	-128...127
int	Целое число (со знаком)	16	-32 768…32 767
float	Вещественное число	32	$\pm1{,}175\cdot10^{-38}\ldots\pm3{,}402\cdot10^{38}$
long int	Длинное целое	32	-2 147 483 648…2 147 483 647
unsigned char	Беззнаковый символ или логическое значение	8	0…255 TRUE, FALSE
unsigned int	Беззнаковое целое	16	0…65 535
unsigned long int	Беззнаковое длинное целое	32	0…4 294 967 295

Чтобы использовать какую-нибудь переменную в программе, ее предварительно необходимо объявить, например,

```
int i, j;                    // объявление двух переменных типа int
unsigned char flag = 0;      //объявление вместе с начальной инициализацией
```

Все переменные могут быть глобальными или локальными. К глобальным переменным имеют доступ все функции, имеющиеся в программе. Такие переменные объявляются в программе перед объявлением всех функций. Локальные переменные объявляются внутри функции и к ним имеет доступ только эта функция.

Любой переменной в программе может быть присвоено числовое значение, причем для этого используют различные формы представления числа:

```
i = 10;          10 в десятичной форме;
i = 0xA;         10 в шестнадцатеричной форме (префикс 0x);
i = 0b1010;      10 в бинарной форме (префикс 0b);
i = 012;         10 в восьмеричной форме (префикс 0);
i = 10.5;        число 10,5 с плавающей точкой;
```

Переменная любого типа может быть объявлена как немодифицируемая. Это достигается добавлением ключевого слова **const** при объявлении переменной, после чего она становится константой:

```
const int b = 286;
```

Следует отметить, что величина, объявленная как константа, будет размещена компилятором в памяти программ, а не в памяти данных (которая у многих микроконтроллеров относительно невелика).

Важной операцией является приведение типов, под которой понимается принудительное преобразование значения одного типа к другому, совместимому с исходным. Это используется в арифметических операциях, когда полученные значения могут, например, выходить за допустимые пределы. Приведение типов бывает явным и неявным. Неявное приведение типов используется в операторах присваивания, когда компилятор сам выполняет необходимые преобразования без явного на то указания. Для явного приведения типа некоторой переменной перед ней следует указать в круглых скобках имя нового типа.

Пример:

```
int  X;
int  Y = 200;
char Z = 30;
X = (int)Z*10+Y;      // при вычислении X переменная Z приведена к типу int
```

Если бы в этом примере не было выполнено явное приведение типов, то компилятор предположил бы, что выражение Z*10 – это восьмиразрядное умножение (разрядности типа **char**) и вместо корректного значения 300 (0x12C) в память данных было бы помещено урезанное значение 44 (0x2C). Таким образом, переменной X было бы присвоено некорректное значение 244 вместо правильного значения 500. В результате приведения типа переменная Z распознается компилятором как 16-разрядная и ошибки в вычислении X не возникает.

При написании программного кода к переменным или другим типам данных можно обратиться через идентификатор (по имени переменной), а можно использовать указатель (по адресу в памяти). Указатель – это переменная, содержащая адрес некоторого элемента данных (переменной, константы, массива, функции и т.д.). При объявлении переменной типа указатель, необходимо определить тип объекта данных, адрес которых будет содержать переменная, и имя указателя с предшествующей звездочкой:

```
int *p;          // p – указатель на целое число
```

Часто в программе совместно с указателями используются оператор **&** (оператор адреса) и оператор * (оператор разадресации, осуществляющий доступ к адресуемой величине через указатель). Пример такого использования:

```
char *p;        // объявляется указатель на символьную переменную
char x, y;      // объявляются символьные переменные x  и y
x = 'A';        // переменной x присваивается символ A
p = &x;         // p присваивается адрес переменной x (указывает на символ A)
y = *p;         // переменной y присвоен символ A
```

Применительно к программированию микроконтроллеров, указатели можно использовать, например, для записи данных в порт ввода/вывода. Пусть регистр данных порта расположен в памяти по адресу 0x16. Для записи в этот регистр данных порта значения 0xFF можно воспользоваться следующим фрагментом программного кода:

```
unsigned char  *Port;
Port = 0x16;
*Port = 0xFF;
```

Эту же операцию можно выполнить и непосредственным присвоением числа 0xFF переменной PORTx, где через x обозначены порты A, B…G (см. примеры в пункте 2.3).

В языке Си любую переменную можно определить, задав с помощью перечисления список ее возможных значений. Такая переменная называется переменной перечислимого типа или просто перечислением. Для этого используется ключевое слово **enum**. Пример:

```
enum week { SUBB = 0, VOSK = 0, POND, VTOR, SRED, CHET, PJAT) rab_ned;
```

В данном примере объявлена переменная rab_ned, имеющая перечисляемый тип с именем week. Первым двум перечисляемым значениям SUBB и VOSK присваиваются значения 0, всем последующим – 1, 2, 3 и т.д. Если никакой явной инициализации перечисляемых значений нет, то по умолчанию они принимают последовательный ряд целочисленных значений, начиная с нуля. В дальнейшем можно объявить другую перечисляемую переменную rab_ned_1 типа week, принимающую такой же ряд значений, что и переменная rab_ned:

```
enum week rab_ned_1;
```

Кроме объявлений переменных различных типов, язык Си позволяет объявлять свои нестандартные (пользовательские) типы данных. Для этого используется ключевое слово **typedef**, например:

```
typedef  unsigned int word                  // объявляется новый тип данных word
```

В этом примере объявляется новый тип данных **word**, который по сути является стандартным типом **unsigned int**, но в некоторых случаях является более наглядным (например, если такой тип имеет двухбайтная переменная, хранящаяся в оперативной памяти микроконтроллера как слово).

2.1.3. Массивы и структуры

Массивы – это группа элементов одинакового типа (int, float, char и т.п.). Массивы объявляются подобно обычным переменным с указанием в квадратных скобках их размерности (количества элементов в массиве):

```
int x[10];                  // массив из 10 элементов типа int
char str[10];               // массив из 10 символов (строка)
```

Доступ к элементам массива осуществляется с помощью индекса (порядкового номера элемента, начиная с нулевого), например:

```
x[5] = 1;
str[0] = 'A';
```

В языке СИ определены только одномерные массивы, но поскольку элементом массива может быть массив, можно определить и многомерные массивы. Примеры объявления таких массивов:

```
int a[2][3];                // двумерный массив из 2 строк и 3 столбцов
int b[3][2][5];             // трехмерный массив из 3 х 2 х 5 элементов
```

Инициализацию массива осуществляют обычно с помощью фигурных скобок, например:

```
int x[10] = { 0, 5, 1, 7, 2, 3, 4, 9, 5, 1 };     // инициализация одномерного массива
int a[2][3] = { 2, 3, 4, 1, 0, 9 };               // инициализация двумерного массива
```

Во втором примере учтено, что фактически все элементы многомерного массива хранятся в памяти последовательно, поэтому инициализацию можно осуществить в одну строку.

Особой разновидностью массива из элементов типа **char** является строка:

```
char str[10] = {'t', 'h', 'e', ' ', '',  'l', 'i', 'n', 'e' };          // строка символов
```

Объявление и инициализацию строки можно также выполнить с помощью строкового литерала:

```
char str[] = "the line";
```

При этом необходимо помнить, что компилятор неявно завершает строковые литералы символом конца строки '\0'. Это следует учитывать, чтобы при инициализации массива не выйти за его пределы. Следует отметить, что при инициализации строк размерность массива можно явно не указывать. В этом случае компилятор определяет размерность самостоятельно.

Структура – это составной объект, в который входят элементы любых типов, за исключением функций. В отличие от массива, который является однородным объектом, структура может состоять из нескольких разнотипных переменных (полей). В общем случае объявление структуры имеет вид:

```
struct  имя_структуры {
    тип   поле_1;
    . . .
    тип   поле_N;          }
```

В качестве примера можно привести структуру, определяющую почтовый адрес получателя корреспонденции, которая включает в себя поля с информацией об имени, почтовом индексе, городе, улице, доме, квартире:

```
struct  address {
    char name [30];                 // имя получателя из 30 символов
    long int  index;                // почтовый индекс
    char town [15];                 // город
    char street [15];               // улица
    int number_ house;              // номер дома
    int number_ apartment;          // номер квартиры
}
```

Для доступа к полям структуры можно использовать запись вида имя_структуры. поле, например:

```
address.name = “V. Ivanov”;
```

Отметим, что структуры при программировании микроконтроллеров используются довольно редко.

2.1.4. Операторы

Как уже отмечалось, операторы указывают компилятору, какие действия (операции) надо выполнить над операндами. По количеству операндов, участвующих в операции, они подразделяются на унарные, бинарные и тернарные. В языке Си имеются следующие унарные операции:

- `-` арифметическое отрицание (отрицание и дополнение);
- `~` побитовое логическое отрицание (дополнение);
- `!` логическое отрицание;
- `*` разадресация (косвенная адресация);
- `&` вычисление адреса;
- `+` унарный плюс;
- `++` увеличение на единицу (инкремент);
- `--` уменьшение на единицу (декремент);
- `sizeof` определение размера памяти, которая соответствует идентификатору.

Унарные операции выполняются справа налево. Операции инкремента и декремента увеличивают или уменьшают значение операнда на единицу и могут быть записаны как справа так и слева от операнда. Если знак операции записан перед операндом (префиксная форма), то изменение операнда происходит до его использования в выражении. Если знак операции записан после операнда (постфиксная форма), то операнд вначале используется в выражении, а затем происходит его изменение. Например:

```
 i ++ ;                 // переменная i инкрементируется на единицу
m = sizeof(f)           // определение области памяти переменной f
```

Список бинарных операций приведен в таблице 2.2. В отличие от унарных операций бинарные выполняются слева направо. При записи выражений

следует помнить, что ряд символов (*, &, - и +) может обозначать унарную или бинарную операцию. Примеры бинарных операций:

```
T0 = (int) T0_hi << 8;          // сдвиг T0_hi на 8 разрядов влево
T0 += ( int) T0_low;            // сложение T0 с T0_low, результат присвоить T0
PORTB & = ~(1 << PB6);          // установка 6-го вывода порта B в состояние лог.0
```

Первые две операции позволяют записать в переменную T0 (целого типа **int**) старший байт T0_hi и младший байт T0_low, имеющие тип **char**. Для этого предварительно произведено преобразование типов.

Таблица 2.2

Бинарные операции языка Си

Знак операции	Операция	Знак операции	Операция
*	Умножение	&&	Логическое И
/	Деление	||	Логическое ИЛИ
%	Остаток от деления	,	Последовательное вычисление
+	Сложение	=	Присваивание
-	Вычитание	*=	Умножение с присваиванием
<<	Сдвиг влево	/=	Деление с присваиванием
>>	Сдвиг вправо	%=	Остаток от деления с присваиванием
<	Меньше	-=	Вычитание с присваиванием
<=	Меньше или равно	+=	Сложение с присваиванием
>=	Больше или равно	<<=	Сдвиг влево с присваиванием
= =	Равно	>>=	Сдвиг вправо с присваиванием
!=	Не равно	&=	Поразрядное И с присваиванием
&	Поразрядное И	|=	Поразрядное ИЛИ с присваиванием
|	Поразрядное ИЛИ	^=	Поразрядное исключающее ИЛИ с присваиванием
^	Поразрядное исключающее ИЛИ		

В языке СИ имеется одна тернарная операция – условная операция, которая имеет следующий формат:

операнд 1 **?** операнд 2 **:** операнд 3

Операнд 1 оценивается с точки зрения его эквивалентности 0. Если операнд 1 не равен 0, то вычисляется операнд 2 и его значение является результатом операции. Если операнд 1 равен 0, то вычисляется операнд 3 и его значение является результатом операции. Следует отметить, что вычисляется либо операнд 2, либо операнд 3, но не оба.

Все операторы могут быть условно разделены на следующие категории:

- условные операторы, к которым относятся оператор условия **if-else** и оператор выбора **switch-case**;

- операторы цикла (**for**, **while**, **do-while**);
- операторы перехода (**break**, **continue**, **return**, **goto**);
- другие операторы (оператор «выражение», пустой оператор).

Операторы в программе могут объединяться в составные операторы с помощью фигурных скобок. Любой оператор в программе может быть помечен меткой, состоящей из имени и следующего за ним двоеточия. Все операторы языка Си, кроме составных операторов, заканчиваются точкой с запятой.

Выполнение оператора «выражение» заключается в вычислении выражения. Примеры таких операторов были рассмотрены выше. Пустой оператор состоит только из точки с запятой. При выполнении этого оператора ничего не происходит. Он обычно ставится в строках программы при использовании операторов **if-else, for**, **do-while** в тех случаях, когда по логике работы программы никакого оператора не требуется, но по синтаксису языка Си необходимо, чтобы в данном месте стоял хотя бы один оператор.

Оператор **if-else** используется для выполнения того или иного блока программы в зависимости от некоторого условия. Формат оператора:

```
if (выражение)   {блок операторов 1;}
[else {блок операторов 2;}]
```

Выполнение оператора начинается с вычисления выражения. Если выражение истинно, то есть отлично от 0, то выполняется блок операторов 1, который заключаются в фигурные скобки. Если выражение ложно, то есть равно 0, то выполняется блок операторов 2. Если выражение ложно и отсутствует **else** (в квадратные скобки заключена необязательная конструкция), то выполняется следующий за **if** оператор. Пример:

```
if (i < j)   i++;                  // если i меньше j, то i инкрементируется на единицу
else { j = i-3;   i++; }           // если иначе, то выполняются другие операции
```

Оператор **switch-case** предназначен для организации выбора из множества различных вариантов. Формат оператора следующий:

```
switch ( выражение )
{
   case  константное выражение1: блок операторов1;}
   case  константное выражение2: {блок операторов2;}
   ……
   default:  {блок операторов;}
}
```

Выражение, следующее за ключевым словом **switch** в круглых скобках, может принимать только целые значения. В зависимости от конкретного значения этого выражения выполняется тот или иной блок операторов. Блок операторов может быть пустым, содержать один оператор или более. Схема выполнения оператора **switch-case** следующая:

- вычисляется выражение в круглых скобках;

- вычисленные значения последовательно сравниваются с константными выражениями, следующими за ключевыми словами **case**;

- если одно из константных выражений совпадает со значением выражения, то управление передается на оператор, помеченный соответствующим ключевым словом **case**;

- если ни одно из константных выражений не равно выражению, то управление передается оператору, помеченному ключевым словом **default**, а в случае его отсутствия – оператору, следующему после **switch-case**.

Пример использования оператора **switch-case**:

```
switch (Num_Bite_Com)
{
        case 0: { I_heat=Receive_Byte_UART1();          // приём 1-го байта
                  Num_Bite_Com++;                       // инкремент счётчика байтов
                 }
                break;
        case 1: { Tau_min=Receive_Byte_UART1();         // приём 2-го байта
                  Num_Bite_Com++                        // инкремент счётчика байтов
                 }
                break;
        ……..
        case 7: { RUN_Meas = Receive_Byte_UART1();      // приём последнего байта
                  Num_Bite_Com=0;                       // обнуление счётчика байтов
         }
                break;
}
```

В данном примере производится прием посылки данных из восьми байтов, полученных из компьютера по последовательному интерфейсу USART, и присвоение полученных значений (в зависимости от их номера в посылке) соответствующим переменным. Оператор **break**, находящийся внутри тела оператора **switch-case**, прерывает его выполнение и осуществляет переход на оператор, следующий за **switch-case**.

Операторы цикла (**for**, **while**, **do-while**) предназначены для повторного выполнения некоторого блока программы. Количество повторов либо заранее задано, либо определяется каким-либо условием.

Оператор **for** имеет следующий формат:

```
for ( выражение 1 ; выражение 2 ; выражение 3 )
{тело цикла из блока операторов;}
```

Выражение 1 обычно используется для установки начального значения переменных, управляющих циклом. Выражение 2 определяет условие, при котором будет выполняться тело цикла. Выражение 3 определяет изменение переменных, управляющих циклом после каждого выполнения тела цикла. Пример использования оператора **for**:

```
summa = 0;
for (i = 1; i < 10; i++)
   {summa += i; }
```

В этом примере вычисляется сумма чисел от 1 до 9.

Оператор цикла **while** называется циклом с предусловием и имеет следующий формат:

```
while (выражение)
{тело цикла из блока операторов;}
```

Схема выполнения оператора **while** следующая:

- вычисляется выражение;
- если выражение ложно, то выполнение оператора **while** заканчивается и выполняется следующий по порядку оператор; если выражение истинно, то выполняется тело цикла;
- затем снова вычисляется выражение и процесс повторяется сначала.

Пример использования оператора **while**:

```
summa = 0;
i = 1;
while (i < 10)
   {summa += i;
    i++;}
```

В этом примере, как и в предыдущем с оператором **for**, вычисляется сумма чисел от 1 до 9. Часто оператор **while** используется для организации бесконечных замкнутых циклов:

```
while (1)                                  // условное выражение всегда истинно
{тело цикла из блока операторов;}          // бесконечный цикл
```

Оператор **do-while** называется оператором цикла с постусловием и используется в тех случаях, когда необходимо выполнить тело цикла хотя бы один раз. Формат оператора имеет следующий вид:

do {тело цикла} **while** (выражение);

Схема выполнения оператора **do while**:

- выполняется тело цикла (которое может быть составным оператором);
- вычисляется выражение;
- если выражение ложно, то выполнение оператора **do-while** заканчивается и выполняется следующий по порядку оператор; если выражение истинно, то выполнение оператора повторяется сначала.

Если в теле любого цикла встречается оператор **break**, то управление передается оператору, следующему за оператором цикла, вне зависимости от истинности или неистинности условного выражения. При этом во вложенных циклах выход осуществляется не на самый верхний уровень вложенности, а лишь на один уровень вверх. Это, в частности, позволяет осуществить выход из бесконечных циклов при выполнении определенных условий.

Оператор **continue**, как и оператор **break**, используется только внутри операторов цикла, но в отличие от него выполнение программы продолжается не с оператора, следующего за прерванным оператором, а с начала прерванного оператора.

Оператор **return** завершает выполнение функции, в которой он задан, и возвращает управление оператору, непосредственно следующему за вызовом функции.

Оператор безусловного перехода **goto** передает управление оператору, помеченному меткой. Помеченный оператор должен находиться в той же функции, что и оператор **goto**, а используемая метка должна быть уникальной. Использование оператора **goto** в практике программирования на языке Си настоятельно не рекомендуется, так как он затрудняет понимание программ и возможность их модификации.

2.1.5. Функции

Функция – это именованная часть программы, предназначенная для решения определенной задачи, к которой можно обращаться из других частей программы без каких-либо ограничений. Каждая функция должна иметь имя, которое используется для ее объявления, определения и вызова. Важно отметить, что в любой программе на Си должна быть функция с именем **main** (главная функция); именно с этой функции, в каком бы месте программы она не находилась, начинается выполнение программы.

Обычно функция имеет некоторый набор аргументов (формальных параметров). При вызове функции ей передаются некоторые значения (фактические параметры), используемые во время выполнения функции, после чего она может возвращать некоторое значение. Это возвращаемое значение и есть результат выполнения функции, который при выполнении программы подставляется в точку вызова функции, где бы этот вызов ни встретился. Допускается также использовать функции, не имеющие аргументов, а также функции не возвращающие никаких значений. Действие таких функций может состоять, например, в изменении значений некоторых переменных, выводе на печать текстов и т.п.

С использованием функций в языке Си связаны три понятия – определение функции (описание действий, выполняемых функцией), объявление функции (задание формы обращения к функции) и вызов функции. Определение функции задает тип возвращаемого значения, имя функции, типы и число формальных параметров, а также объявления переменных и операторы, называемые телом функции, и определяющие действие функции. Пример:

```
char znak (int x)
{  if (x >= 0)                    // если x больше или равен нулю, то
     return 1;                    // функции znak присваивается 1;
  else                            // иначе
     return 0;                    // функции znak присваивается 0.
 }
```

В данном примере определена функция znak(x), имеющая один аргумент – целое число x. Если при вызове функции ей передается положительное число ($x \geq 0$), то функция возвращает значение 1, в противном случае – 0. Если функция имеет несколько параметров, то они при описании функции и при ее вызове

разделяется запятыми. Если функция предназначена для возвращения какого-то значения, то в теле функции обязательно должен быть оператор **return**. При этом все операторы после **return** игнорируются и происходит возврат в вызываемую функцию. Для функций, не возвращаемых никакого значения, при ее описании должен быть использован тип **void**, указывающий на отсутствие возвращаемого значения. Если функция не использует параметров, то наличие круглых скобок обязательно, а вместо списка параметров рекомендуется указать слово **void**.

В языке Си нет требования, чтобы определение функции обязательно предшествовало ее вызову. Определения используемых функций могут следовать за определением функции **main**, перед ним, или вообще находится в другом файле. Однако для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических параметров типам формальных параметров, до вызова функции нужно поместить объявление (прототип) функции. Объявление функции имеет такой же вид, что и определение функции, с той лишь разницей, что тело функции отсутствует, и имена формальных параметров тоже могут быть опущены. Для функции, определенной в последнем примере, прототип может иметь вид:

char znak (**int** x) или **char** znak (**int**)

В программах на языке Си широко используются, так называемые, библиотечные функции, т. е. функции, предварительно разработанные и записанные в библиотеки. Прототипы библиотечных функций находятся в специальных заголовочных файлах, поставляемых вместе с библиотеками в составе систем программирования, и включаются в программу с помощью директивы **#include** (директивы препроцессора будут рассмотрены ниже).

2.1.6. Структура программы на Си

Программа на Си обычно начинается с директив препроцессора, которые помечаются символом «#». Препроцессор – это часть компилятора, который выполняет макроподстановки, условную компиляцию и включение именованных файлов. Директивы по существу не являются конструкциями языка Си и обрабатываются до фактической компиляции программы. Их смысл – подстановка некоторого кода в программу. Более подробно директивы препроцессора будут рассмотрены ниже.

После директив следуют объявления глобальных переменных и констант, которые доступны для использования во всех функциях программы. Вместе с объявлением может осуществляться и инициализация переменных, то есть присваивание им некоторых начальных значений.

После этого обычно идет объявление и описание используемых в программе функций. Функции могут содержать свои собственные локальные переменные, которые доступны только в теле функции. В программе обязательно должна быть функция с именем **main** (главная функция). Именно с этой функ-

ции, в каком бы месте программы она не находилась, начинается выполнение программы.

Пример программы на Си:

```
#include <stdio.h>                  // подключение библиотечного файла stdio
#include <iom128v.h>                // подключение библиотечного файла iom128v
#include "UART.h"                   // подключение файла UART
#define Pi    (float) 3.1416        // присвоение значения числу π
…….                                 // другие директивы программы

unsigned int adc_value;             // объявление переменной adc_value
unsigned int T_0=120;               // объявление и инициализация переменной T_0
…….                                 // другие глобальные переменные программы

void port_init (void)               // функция инициализации портов
{
PORTA = (1<<PA6);
DDRA  = (1<<DDA6) | (1<<DDA0);
PORTB = 0x00;
DDRB = (1<<DDB5) | (1<<DDB6);
…….                                 // инициализация портов C, D, E и F
}                                   // конец функции port_init ()

…….                                 // другие функции программы

main ( )                            // главная функция программы main ()

{ unsigned char I = 0, k = 0;       // объявление локальных переменных
  unsigned int ii;
  init_devices( );                  // функция инициализации устройства
  PORTB &= ~(1<<PB5);               // принудительная установка вывода PB5 в лог.0
  …….                               // другие операторы функции main ()
}                                   // конец функции main ()
```

Важно отметить, что инициализация портов и других периферийных устройств микроконтроллера ATmega128 в компиляторе Image Craft, который будет рассмотрен ниже, может осуществляться в полуавтоматическом режиме.

2.1.7. Директивы препроцессора

Как уже отмечалось, директивы препроцессора в основном предназначены для подстановки некоторого кода в программу. Практически каждая программа содержит директиву **#include**, которая позволяет подключить к данной программе другие программы с полезными функциями. Для этого достаточно вместе с директивой **#include** указать так называемый заголовочный файл, то есть имя подключаемого файла с расширением «.h» (см. пример программы на Си в п.2.1.6). После этого функции, прототипы которых указаны в заголовочном файле, становятся доступными для использования их в данной программе. Эта директива имеет две формы:

```
#include  <имя файла>
#include  "имя файла"
```

Имя файла может состоять либо только из имени файла, либо из полного имени файла, включающего и путь к нему. Если имя файла задано в угловых скобках, то он считается частью стандартной библиотеки, поставляемой вместе с компилятором. Если имя файла указано в кавычках, то поиск файла осуществляется в соответствии с заданным путем к нему, а при его отсутствии – в текущем каталоге. Директива **#include** может быть вложенной, то есть во включаемом файле также могут содержаться свои директивы **#include**.

Другой, широко распространенной директивой, является **#define**, которая служит для замены часто использующихся констант, ключевых слов, операторов или выражений некоторыми идентификаторами. Идентификаторы, заменяющие текстовые или числовые константы, называют именованными константами. Идентификаторы, заменяющие фрагменты программ, называют макроопределениями, причем макроопределения могут иметь аргументы. Директива **#define** имеет следующий формат:

#define идентификатор текст

Если в тексте программы встретится указанный в директиве идентификатор, то компилятор заменит его на текст. Такой процесс называется макроподстановкой. Пример:

```
#define WIDTH 80
#define LENGTH (WIDTH+10)
```

Эти директивы изменят в тексте программы каждый идентификатор WIDTH на число 80, а каждый идентификатор LENGTH на выражение (80+10) вместе с окружающими его скобками.

В компиляторе Image Craft, который будет рассмотрен ниже, широко используется директива **#pragma**, которая объявляет функцию обработки прерываний, включая описание источника прерывания. Например, для прерывания, вызванного совпадением содержимого таймера/счетчика **TCNT1** и его регистра захвата **ICR1**, директива **#pragma** имеет вид:

```
#pragma interrupt_handler timer1_capt_isr:12
void timer1_capt_isr(void)
{
    *P_Data=ReadADC_ext();          // чтение результата преобразования АЦП
}
```

При возникновении запроса на прерывание выполнение программы приостанавливается, вызывается функция timer1_capt_isr(), которая осуществляет чтение результата преобразования внешнего АЦП (функция ReadADC_ext описана в другом месте программы) и присваивает его переменной, на которую указывает указатель *P_Data. После этого выполнение программы возобновляется с того места, на котором возник запрос на прерывание.

Существует довольно много других, менее широко используемых директив препроцессора, с которыми можно ознакомиться в специальной литературе по языку Си.

2.2. Интегрированная среда разработки ICCAVR

Среда программирования ICCAVR фирмы ImageCraft Creations Inc предназначена для разработки программного обеспечения (проекта) и его компиляции в исполняемый AVR-микроконтроллером файл. ICCAVR является одним из простейших и очень удобных компиляторов. Он имеет набор стандартных библиотечных функций и ряд подпрограмм, специально предназначенных для AVR-микроконтроллеров. Их можно использовать в своем проекте, предварительно подключив их к проекту директивой **#include**.

В работе ICCAVR используются следующие типы файлов (по расширениям):

C – исходный текст на языке Си;

S – исходный текст на ассемблере или выходной ассемблерный файл, генерируемый для каждого исходного C-файла;

H – заголовочный (header) файл;

PRJ – файл проекта;

SRC – список файлов проекта;

O – объектный файл, получаемый после компиляции ассемблерного файла;

HEX – выходной файл в формате Intel HEX для загрузки в ПЗУ программ микросхемы;

EEP – выходной файл в формате Intel HEX для загрузки в ПЗУ данных микросхемы;

COF – выходной файл в формате COFF, используется при отладке проекта в AVR Studio или других средах программирования;

LST – файл-листинг, содержащий информацию об адресах;

MP – MAP-файл, содержащий символическую информацию;

DBG – файл с отладочной информацией;

A – библиотечный файл.

При первоначальном изучении ICCAVR достаточно знать, что в данной среде пользователь разрабатывает проект (файл с расширением PRJ), состоящий из одного или нескольких исходных C-файлов и заголовочных H-файлов. В проект могут также входить текстовые файлы с дополнительной информацией, например, об особенностях работы алгоритма программы или какой-то справочной информацией. Если компиляция проекта проходит успешно, то образуется HEX-файл, который с помощью специальных средств «прошивается» в память программ микроконтроллера. HEX-файл можно также использовать в системах схемотехнического моделирования (например, в PROTEUS) для проверки правильности функционирования микроконтроллера.

После запуска ICCAVR на экране монитора появляется главное окно, которое разделено на три части (окна). Левая верхняя часть – окно редактора. В нем представлены тексты программ, которые образуют проект. Именно с этими текстами и работает программист, редактируя их или добавляя новые строки и блоки программ. При первом запуске ICCAVR окно, естественно, пустое.

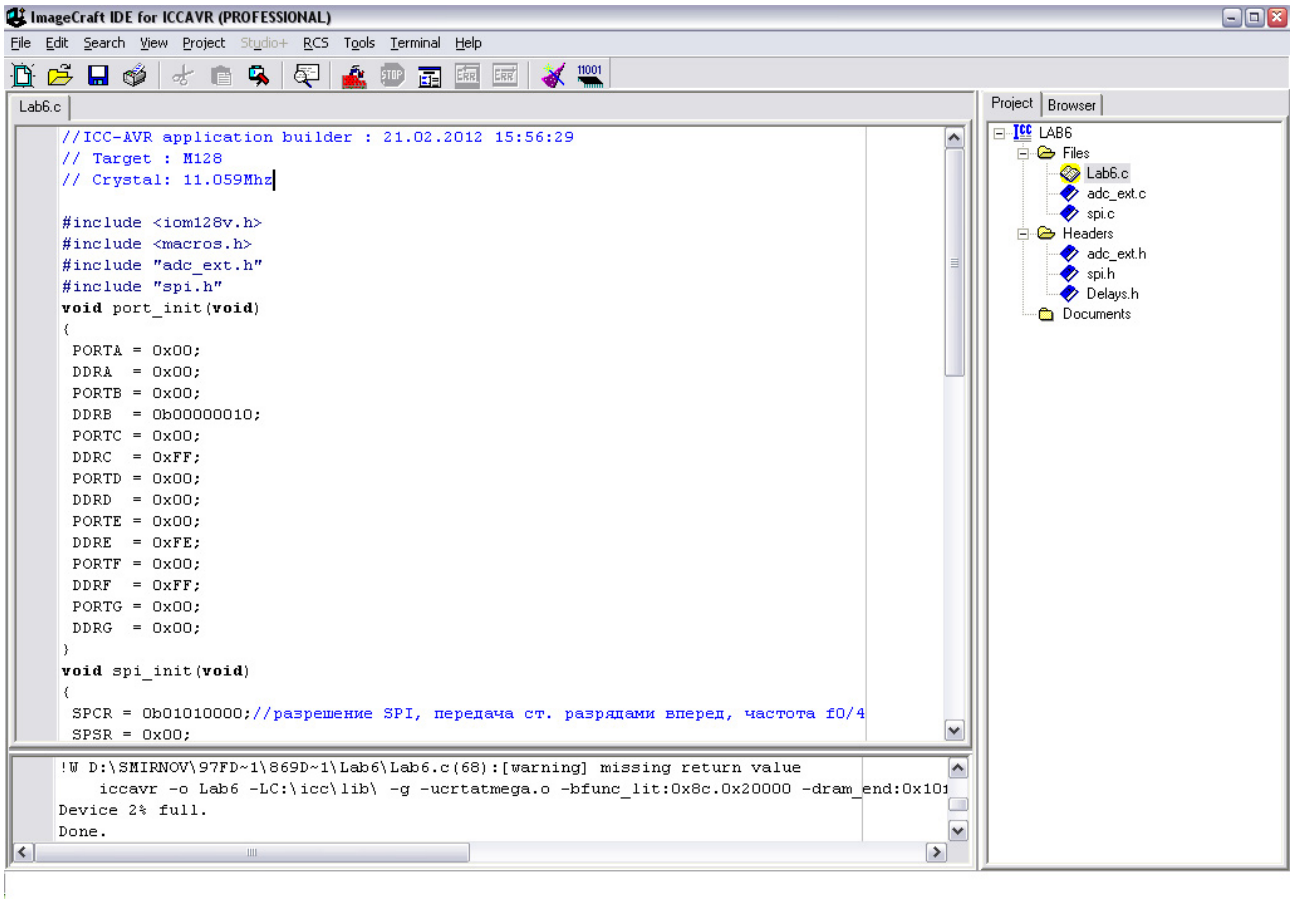

Рис. 2.1. Главное окно ICCAVR

Правая часть – окно менеджера проекта, содержащее две вкладки: Project и Browser. Первая содержит список файлов проекта, каждый из которых можно открыть в окне редакторе двойным щелчком мыши. Вторая вкладка – обозреватель кода, показывающий список функций и переменных, определенных в проекте. При двойном щелчке мыши на функции в обозревателе кода, курсор переместится на определение функции в исходном файле.

Нижняя часть – окно состояния. В нем показываются результаты компиляции отдельного файла или всего проекта в целом. Если имеются ошибки, то после компиляции появятся сообщения об обнаруженных ошибках с указанием их места расположения и подсказкой о типе ошибки. Следует помнить, что компилятор проверяет ошибки синтаксиса программы, а не правильность ее алгоритма.

Все свои действия по управлению проектом или отдельными файлами пользователь осуществляет, используя пункты меню в верхней части главного окна, кнопки панели инструментов (под пунктами меню) или правую кнопку мыши и всплывающее при этом контекстное меню. Рассмотрим кратко наиболее важные пункты меню.

File позволяет открыть какой-либо файл или создать новый, сохранить, закрыть или вывести на печать и т.д. Полезен пункт **Reopen...**, который позволяет выбрать и загрузить один из последних файлов, с которыми работал пользователь.

Edit позволяет осуществить действия по редактированию текста открытого в окне файла: скопировать в буфер или вставить из него фрагмент текста, «вырезать» выбранный фрагмент текста в буфер или удалить его совсем, отменить последнее исправление и т.д. Отметим, что каждую из этих операций можно быстро осуществить, используя соответствующую комбинацию клавиш.

Search позволяет находить заданные пользователем слова в тексте открытого файла проекта или во всем проекте, заменять отдельные фрагменты текста на другой текст, перемещать курсор на нужный номер строки текста открытого файла или на нужную метку, добавлять или удалять метки и т.д.

Project позволяет открыть какой-либо проект или создать новый; открыть или закрыть все файлы проекта; открыть один из последних проектов, с которым работал пользователь; закрыть проект или сохранить его под новым именем; добавить открытый в окне файл в проект или удалить из проекта файл, который выбран в окне менеджера проекта. Пункт **Make Project** позволяет осуществить компиляцию открытого в окне редактора файла, а пункт **Rebuild All** – компиляцию всех файлов проекта. Важным пунктом является **Option...**, который открывает диалоговое окно, показанное на рис. 2.2.

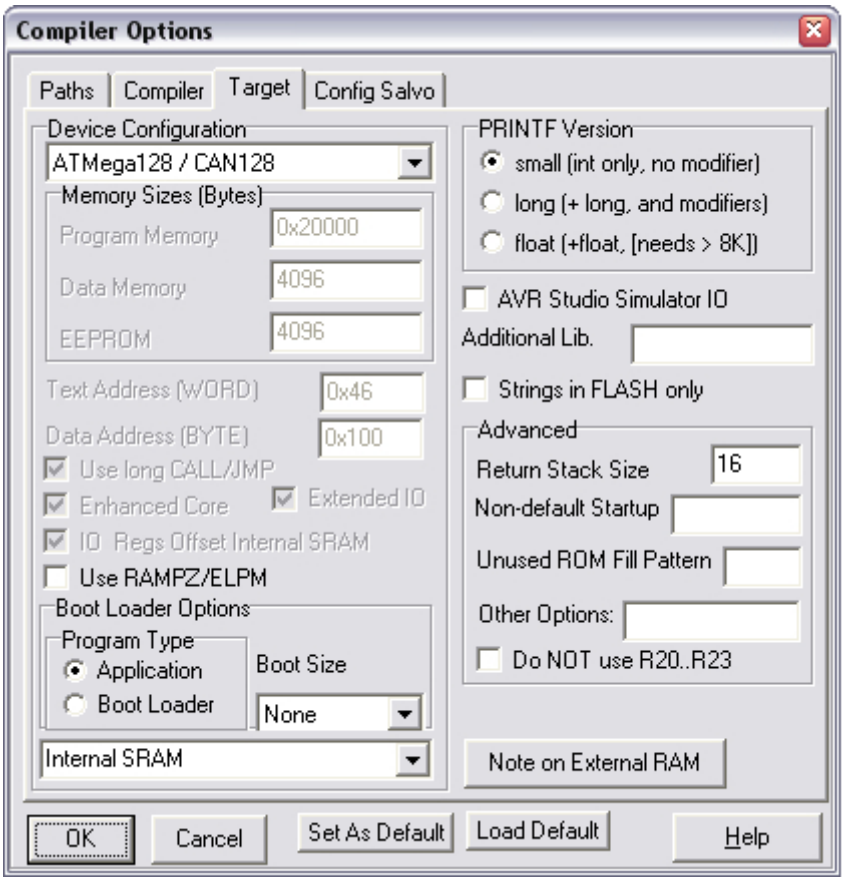

Рис. 2.2. Окно при выборе пункта меню **Project > Option**

В пункте **Target** в окне **Device Configuration** следует задать микроконтроллер, для которого разрабатывается проект, например ATmega128/CAN128 (см. рис. 2.2). В пункте **Paths** необходимо установить пути к библиотечным файлам и файлам, которые пользователь подключает к проекту, например, для

библиотечных файлов **c:\icc\lib**. В пункте **Compiler** в окне **Output Format** установить формат выходных файлов – COFF/HEX.

В пункте **Tools** главного окна важными являются три пункта: **Editor Options, Application Builder** и **In System Programmer**. Пункт **Tools>Editor Options** позволяет установить нужные опции для редактора текста. Здесь лучше все оставить без изменений. Но если возникает необходимость писать комментарии в программе на русском языке, то в пункте **Tools>Editor Options> Highlighting** необходимо в окне **Characters** установить опцию **Russian**.

Пункт **Tools>Application Builder** предназначен для инициализации периферийных устройств микроконтроллера, а также для настройки системы внешних прерываний. Настройку периферийных устройств можно выполнить и вручную, присвоив в программе соответствующим регистрам управления и статуса нужные значения. Однако с помощью модуля **Application Builder** эта инициализация осуществляется гораздо быстрее и надежнее. Окно **Application Builder** с опциями настройки портов ввода/вывода представлено на рис. 2.3.

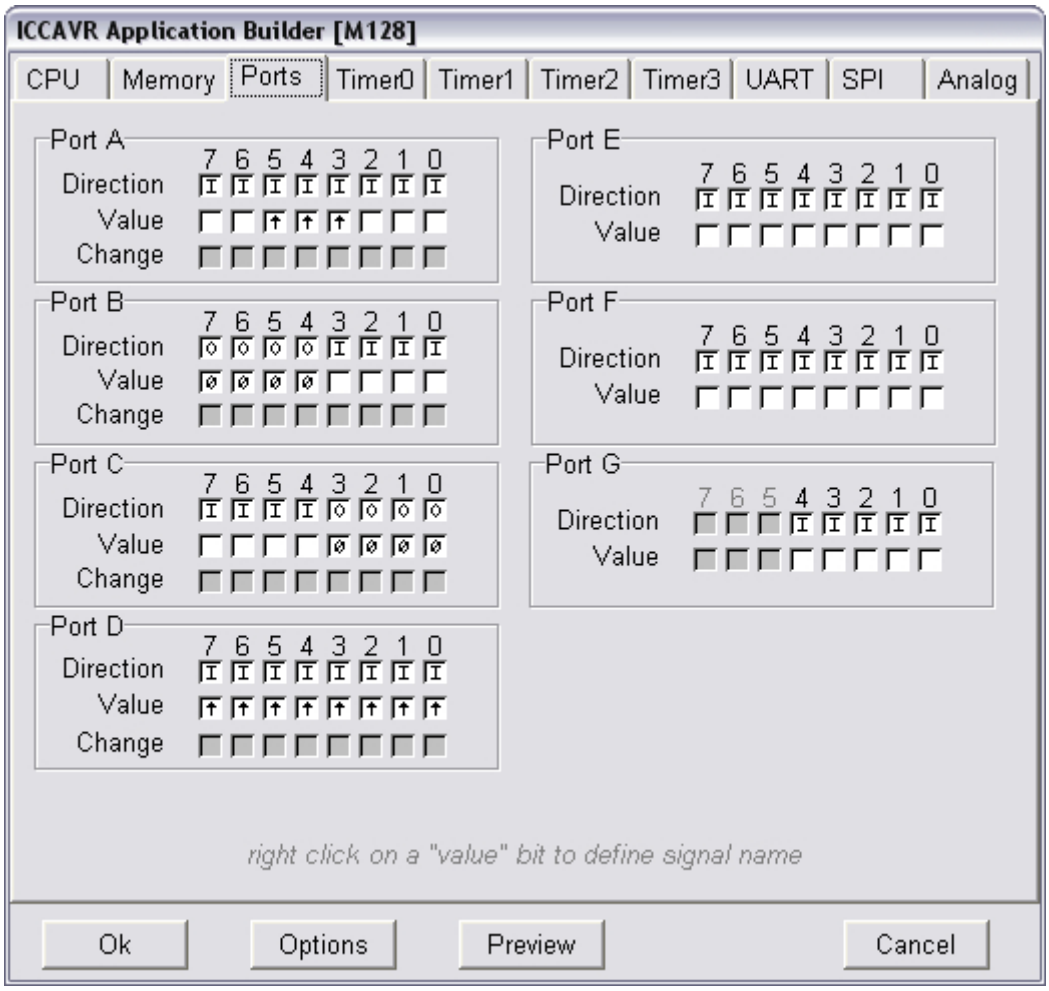

Рис. 2.3. Окно **Application Builder** с опциями настройки портов ввода/вывода

В качестве примера на рис. 2.3 показана инициализация портов A, B, C и D, а для портов E, F и G состояние регистров направления **DDRx** и данных **PORTx** (x = E, F, G) оставлено без изменений. Видно, что все выводы портов

(за исключением 4-х старших у порта B и 4-х младших у порта C) настроены на вход. Кроме того, у выводов 3, 4 и 5 порта A и у всех выводов порта D подключены подтягивающие резисторы. В результате **Application Builder** сгенерирует программный код с функцией void port_init(void), при выполнении которой всем регистрам **DDRx** и **PORTx** будут присвоены значения, соответствующие заданным пользователем установкам (см. рис. 2.3).

```
void port_init(void)
{
  PORTA = 0x38;      // в двоичной системе равно 0b00111000
  DDRA  = 0x00;
  PORTB = 0x00;
  DDRB  = 0xF0;      // в двоичной системе равно 0b11110000
  PORTC = 0x00;
  DDRC  = 0x0F;      // в двоичной системе равно 0b00001111
  PORTD = 0xFF;      // в двоичной системе равно 0b11111111
  DDRD  = 0x00;
  PORTE = 0x00;
  DDRE  = 0x00;
  …….                // порты F и G инициализированы также, как и порт E
}
```

Подобным образом с помощью **Application Builder** можно настроить работу системы внешних прерываний, таймеров/счетчиков, всех интерфейсов (USART, SPI, TWI), АЦП, компаратора, а также решить ряд других задач, связанных с работой памяти EEPROM, сторожевого таймера и т.д. Если разрешены какие-нибудь прерывания, например, от таймеров или USART, то **Application Builder** генерирует программный код, содержащий шаблоны функций для обработки этих прерываний.

Пункт **Tools> In System Programmer** предназначен для «прошивки» HEX-кода программы в память микроконтроллера с помощью программаторов. После настройки программатора (типа и интерфейса) клавишей Program FLASH/EEPROM инициирует запись кода в память микроконтроллера. Можно также прочитать из памяти микроконтроллера Fuse-биты, определяющие особенности его работы. Можно произвести корректировку Fuse-битов в соответствующем окне, после чего записать скорректированные значения в память микроконтроллера. Следует соблюдать особую осторожность при работе с Fuse-битами, так как запись ошибочного значения в память может создать серьезные проблемы по возвращению микроконтроллера в исходное состояние.

Выполнение наиболее важных пунктов меню можно осуществить с помощью кнопок на панели инструментов или, используя правую кнопку мыши и всплывающее при этом контекстное меню. Это, в частности, относится к вызову таких пунктов, как **Build Project** (компиляция файлов), **Project Options** (установка опций проекта), **Application Builder** (инициализация периферийных устройств), **ICP Dialog** (прошивка кода программы). Если курсор находится в окне редактора, то с помощью правой кнопки мыши можно вызвать функции

редактирования текста, поиска меток или добавления новых файлов в проект. Если курсор находится в окне менеджера проекта, то можно осуществить компиляцию, добавление или удаление файлов из проекта и т.д.

Рассмотрим последовательность действий при создании нового проекта. Для начала создадим рабочую папку, в которой будут храниться все файлы проекта. Затем выберем пункт меню **Project>New** и в открывшемся окне зададим имя проекта и путь к нему. В окне менеджера появится имя нового проекта и пустые папки для хранения С-файлов, Н-файлов и файлов для документов, например, в текстовом формате.

Далее необходимо произвести настройки проекта, установив с помощью **Project>Options>Target** в окне Device Configuration нужный тип микроконтроллера, например, ATMega128/CAN128. С помощью **Project>Options>Paths** установить в окне **Library Path** пути к стандартным библиотечным файлам и в окне **Include Paths** – пути к подключаемым к проекту файлам. Эти файлы находятся в папках **icc\lib** и **icc\include**, которые, в сою очередь, находятся в папке, где установлен компилятор ICCAVR. Если это не первый проект, создаваемый на данном компьютере, то правильные пути к библиотекам уже установлены, надо только это проверить.

Теперь надо подключить файл с исходным кодом, предварительно выбрав пункт **File>New**. Можно написать исходный код непосредственно в окне редактора. В этом случае очень полезна комбинация клавиш Ctrl+j, которая открывает окно с шаблонами операторов. Использование этой комбинации клавиш поможет избежать синтаксических ошибок при разработке программ.

Если текст программы (С-файл) уже есть и его надо лишь подредактировать, то можно открыть этот файл и произвести необходимые исправления. Для начального обучения полезно поработать с исходными С-файлами, расположенными в папке **icc\examples.avr**. Эта папка создается при установке компилятора ICCAVR. Полезно, например, открыть программу **led.c**, которая позволяет переключать состояние одного из выводов микроконтроллера из низкого состояния в высокое и наоборот.

Если программа относительно сложная, то следует воспользоваться **Tools>Application Builder**. Он осуществит начальную инициализацию периферийных устройств и сгенерирует начальный фрагмент программного кода, который пользователь может дорабатывать по своему усмотрению. Необходимо помнить, что после вызова **Application Builder** в пункте CPU следует установить тип микроконтроллера, например, M128 (сокращение от ATMega128) и частоту синхронизации Xtal speed, например, 11.059MHz. Все остальные периферийные устройства настраиваются (или не настраиваются) в соответствии с решаемыми в данной программе задачами. Обычно этот фрагмент кода заканчивается функцией init_devices. Имя этой функции следует вставить в текст главной функции программы main одной из первых (сразу после объявления переменных в функции main), так как именно с инициализации периферийных устройств начинается выполнение программы. Следует помнить, что в про-

грамме обязательно должна быть главная функция с именем main, иначе компилятор будет выдавать сообщения об ошибке.

После создания исходного файла, разработанного на языке Си, его необходимо сохранить в рабочей папке, не забыв про расширение к имени файла, так как автоматически расширение файла не формируется. После этого следует добавить файл к проекту, используя, например правую кнопку мыши и выпадающее контекстное меню. В результате в окне менеджера проекта в папке Files появится имя первого файла проекта. К проекту можно добавлять новые С-файлы или Н-файлы, предварительно выбрав папку Headers. Таким образом, создается проект, включающий в себя несколько файлов. Такая модульная структура проекта позволяет лучше воспринимать его алгоритм функционирования. Для контроля ошибок при разработке программы следует периодически осуществлять компиляцию отдельного файла или всех файлов проекта. Успешная компиляция заканчивается сообщением Done, неуспешная – сообщениями об ошибках и дополнительной информацией о сути этих ошибок и их месте нахождения в программе.

2.3. Примеры программирования периферийных устройств

Пример 1. Установка отдельных бит в регистрах периферийных устройств в состояние лог.1 или лог.0.

```
PORTB | = (1<<PB4)|(1<<PB0);          // выводы 0 и 4 порта В установлены в лог.1
DDRB & = ~(1<<DDB5);                  // установка направления вывода PB5 на вход
```

Первый оператор устанавливает в состояние лог.1 два вывода порта В, оставляя остальные выводы без изменений. Второй оператор обнуляет пятый бит регистра DDR порта В, настраивая вывод PB5 на вход. Остальные выводы остаются при этом без изменений. Альтернативный вариант этих же операций:

```
PORTB | = 0b00010001;                 // выводы 0 и 4 порта В установлены в лог.1
DDRB & = 0b11011111;                  // установка направления вывода PB5 на вход
```

Пример 2. Ожидание нажатия кнопки, в результате чего на выводе 1 порта В устанавливается низкое напряжение (бит PINB1 в регистре PINB становится равным 0) и далее программа продолжает выполнять следующие операторы.

```
while (PINB&0b00000010)   // ожидание нажатия кнопки: 0 - нажата: 1 - не нажата
{;}                       // пока кнопка не нажата, следующие операторы не выполняются
```

Данный фрагмент программы обычно используется, кода требуется инициировать какое-то действие путем нажатия кнопки. В ожидании нажатия кнопки программа с помощью оператора **while** «зациклена» в одном месте. Следующие за ним операторы будут выполняться лишь тогда, когда кнопка будет нажата и 1-й бит регистра PINB установится в лог.0.

Пример 3. Передача данных через универсальный асинхронный приемопередатчик USART0 с использованием опроса флага освобождения буфера передатчика **UDRE** и предварительной инициализацией USART0.

```
void uart0_init(void)            // инициализации USART0 (генерируется Application Builder)
{
 UCSR0A = 0x00;
 UCSR0C = 0x06;                  // установка формата данных в посылке: 8 бит, 1 стоп-бит
 UBRR0L = 0x0B;                  // установка скорости передачи данных: 57600 бод
 UBRR0H = 0x00;
 UCSR0B = 0x08;                  // разрешение работы передатчика
}
void USART_Transmit( unsigned char data )  // функция передачи данных через USART
{
while ( !( UCSRA & (1<<UDRE)) );           // ожидание освобождения буфера передатчика
UDR = data;                                // помещение данных в буфер и их пересылка
}
```

Программа будет ожидать, пока первый бит в регистре **UCSRA**, т.е. бит UDRE, сигнализирующий об освобождении буфера передатчика, станет равным лог.1. После этого будет выполнен следующий за **while** оператор – в буфер передатчика **UDR** будет записано некоторое числовое значение, которое затем автоматически будет пересылаться через вывод микроконтроллера TXD0 получателю этой информации, например, в компьютер.

Пример 4. Чтение результата преобразования внутреннего 10-разряжного АЦП. Предварительно в функции инициализации adc_init() произведены все необходимые настройки АЦП, включая установку источника опорного напряжения (ИОН), предделителя частоты и т.д. Операцию чтения АЦП выполняет функция ReadADC(), которой передается результат преобразования.

```
void adc_init(void)                //инициализация АЦП (генерируется Aplacation Builder)
{
 ADCSRA = 0x00;                    // работа АЦП пока запрещена
 ADMUX = 0b11000000;               // подключен внутренний ИОН на 2,56 В
 ACSR  = 0x80;                     // отключен аналоговый компаратор
 ADCSRA = 0xC6;                    // установлены биты ADEN, ADSC (разрешение и старт),
                                   //а также ADPS2 и ADPS1 (предделитель на 64)
}
int ReadADC(void)                  //функция чтения внутреннего АЦП
{
   int data_ADC;                   // объявление переменной data_ADC
   ADMUX |= (1<<MUX0);             // выбор канала мультиплексора (канал №1).
   ADCSRA |= (1<<ADSC);            // старт преобразованию установкой бита ADSC в лог. 1
   while (ADCSRA & (1<<ADSC));     // ожидание, пока закончится преобразование и
                                   // бит ADSC в регистре ADCSRA обнулится
   data_ADC = ADCL;                // чтение младшего байта результата преобразования
   data_ADC += (int)ADCH << 8;     // прибавление к результату старшего байта
   return data_ADC;                // передача результата на выход функции
}
```

Если теперь в основной программе, содержащей функцию main, встретится оператор вида:

```
X = ReadADC();
```

то будет инициирована работа внутреннего АЦП, а результат аналого-цифрового преобразования будет присвоен переменной X. Разумеется, эта переменная должна быть объявлена ранее до вызова функции ReadADC(), причем переменная X должна иметь тип **int**.

Пример 5. Чтение результата преобразования внешнего 16-разрядного АЦП с последовательным выходом, например, AD7683. АЦП взаимодействует с ATmega128 посредством SPI-интерфейса. Особенностью работы АЦП данного типа является то, что до начала выполнения аналого-цифрового преобразования необходимо на тактовый вход АЦП в «ручном» режиме подавать чередующиеся лог.0 и лог.1 и ждать, когда на выходе АЦП появится лог.0. После этого инициируется работа SPI-интерфейса и с большой скоростью происходит чтение результата преобразования. АЦП активен, т.е способен выполнять свои функции под управлением микроконтроллера, если у него на выводе CS (выбор кристалла) установлен лог.0. Это позволяет микроконтроллеру из нескольких устройств, с которыми он взаимодействует посредством SPI-интерфейса, выбрать тот, который нужен в данный момент.

```
int ReadADC_ext(void)          //функция чтения внешнего АЦП с помощью SPI-интерфейса
{
unsigned int tmp=0;            // объявление переменной целого типа tmp, ей будет передан
                               // результат преобразования
SPCR=(1<<MSTR);                // выбор режима работы "Master", первым передается старший
                               // значащий разряд, скорость передачи fclk/4
PORTE&=~(1<<PE7);              // сигнал PE7 (сигнал CS у АЦП)  - в лог.0
while (PINB&(1<<PB3))
  {                            // подготовка АЦП к работе, что включает в себя:
   PORTB|=(1<<PB1);            //  формирование чередующихся лог.1 и лог.0, ожидание
   PORTB&=~(1<<PB1);           //  появления лог.0 на выводе Dout у АЦП
  }
PORTB|=(1<<PB1);               //  установка вывода в лог.1
SPCR|=(1<<SPE);                //  разрешение работы модуля SPI при частоте fclk/4:
SPSR|=(1<<SPI2X);              //  включение бита SPI2X – бита удвоения скорости обмена
SPDR=123;                      //  запись в регистр данных SPDR произвольного числа для
                               //  инициализации работы SPI (это особенность работы АЦП)
while(!(SPSR & (1<<SPIF)));  // ожидание конца передачи ст. байта и установки флага SPIF
tmp=SPDR<<8;                   //  запись ст. байта  рез-тата преобразования в переменную tmp
SPDR=123;                       //  запись в регистр данных SPDR произвольного числа
while(!(SPSR & (1<<SPIF)));  // ожидание конца передачи мл. байта и установки флага SPIF
tmp|=SPDR;                     //  дополнение результата младшим байтом
 SPCR&=~(1<<SPE);              //  запрещение работы модуля SPI
PORTE|=(1<<PE7);               //  отключение сигнала CS - выбор микросхемы АЦП
return tmp;                    //  результат выполнения функции передан переменной tmp
}
```

3. СХЕМОТЕХНИЧЕСКОЕ МОДЕЛИРОВАНИЕ МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ В СИСТЕМЕ PROTEUS

При проектировании микропроцессорных устройств неизбежно возникают ошибки, которые можно обнаружить на основе анализа функционирования предварительно изготовленного макета. Но изготовить макетную плату устройства, имеющего в своем составе микросхемы с большим количеством выводов, весьма проблематично. Решить данную проблему можно, используя программные средства схемотехнического моделирования. В этом случае объектом анализа является принципиальная схема устройства, включающая в себя компоненты, соединенные друг с другом определенным образом. Используя модели всех компонентов и зная схему их соединений, можно смоделировать работу устройства в целом. Такая возможность особенно важна на стадии изучения микроконтроллеров и особенностей проектирования микропроцессорных устройств. Нет необходимости изготавливать реальную плату или ее макет, невозможно сжечь какой-либо компонент при неправильном его монтаже.

Среди средств схемотехнического моделирования (PSpice, MicroCap, Multisim, DesignLab и др.) следует отметить систему Proteus фирмы Labcenter Electronics, позволяющую производить моделирование принципиальных схем, используя обширную библиотеку моделей электронных компонентов, включая широкий набор микроконтроллеров (AVR, MCS51, PIC, ARM и др). В Proteus есть набор виртуальных измерительных приборов таких, как осциллограф, логический анализатор, вольтметр, спектроанализатор и др. Они позволяют определить и визуально представить электронное состояние в любой точке моделируемой схемы, а также наблюдать процессы, происходящие в ней.

Программный пакет Proteus состоит из двух модулей: ISIS – программа синтеза и моделирования непосредственно электронных схем и ARES – программа разработки печатных плат. Вместе с самим пакетом Proteus устанавливается широкий набор демонстрационных проектов для ознакомления с работой системы. Перед изучением системы Proteus очень полезно с ним ознакомиться. Мы будем рассматривать только модуль ISIS. При этом ограничимся рассмотрением его основных возможностей, достаточных для работы с этой системой. Более детальную информацию можно получить «экспериментальным» путем, либо изучив информацию, выбрав пункт меню **«Справка»**.

3.1. Интерфейс системы схемотехнического моделирования Proteus

При запуске Proteus на мониторе появляется главное окно, представленное на рис. 3.1. Все его рабочее пространство разделено на несколько областей. Большую часть занимает окно редактирования. В нем проектируется принципиальная схема устройства. Сюда из библиотеки вставляются электронные компоненты вместе со средствами измерения и индикации, здесь они соединяются проводниками согласно замыслу проектировщика, редактируются при наличии ошибок, после чего запускается процесс моделирования.

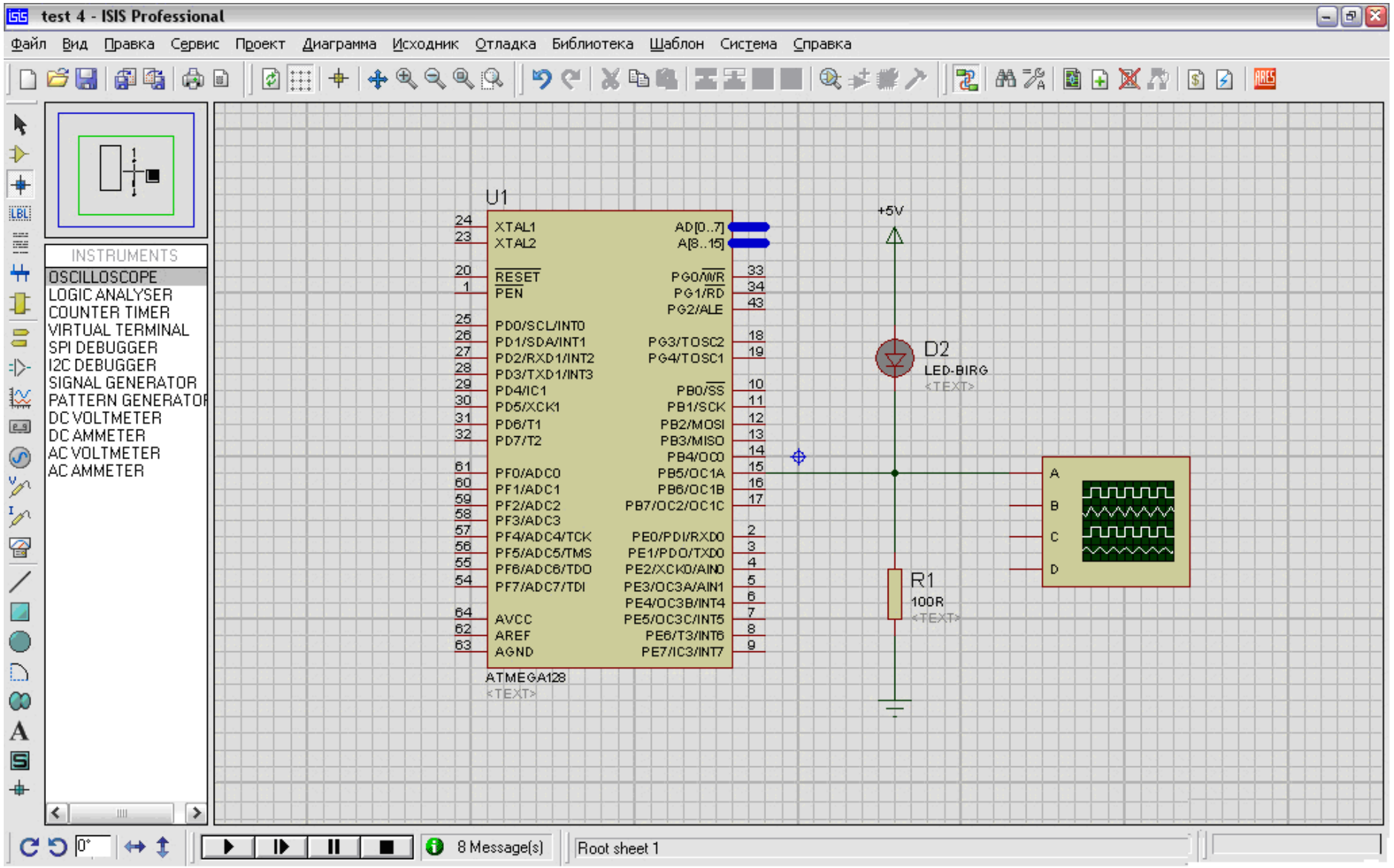

Рис. 3.1. Главное окно системы схемотехнического моделирования Proteus

Вверху находятся пункты меню, предоставляющие пользователю полный набор возможных действий, имеющихся в Proteus. Под ним расположены кнопки верхней панели инструментов, позволяющие выполнить часто используемые команды такие, как открытие и сохранение проекта, масштабирование и позиционирование принципиальной схемы устройства, копирование и перемещение выделенных областей схемы. На левом краю главного окна расположена левая панель инструментов, с помощью которых, в основном, и проектируется принципиальная схема устройства. Следует понимать, что одно и то же действие можно выполнить различными способами, использую систему меню, панель инструментов или контекстное меню, выпадающее при нажатии правой кнопки или двукратном нажатии левой кнопки мыши.

В левом верхнем углу главного окна располагается окно предварительного просмотра, позволяющее оперативно перемещаться по схеме проекта. Под ним расположено окно, куда выводится различная информация, характер которой зависит от того, какая из кнопок нажата на левой панели инструментов. Это может быть список компонентов схемы, меток, виртуальных инструментов, пробников и т.д. Внизу на краю главного окна расположены четыре кнопки управления процессом моделирования: **«Воспроизвести»** – старт процессу моделирования, **«Шаг»** – пошаговое выполнение программы микроконтроллера, **«Пауза»** – пауза процесса моделирования, **«Стоп»** – остановка процесса моделирования.

3.2. Панель инструментов системы Proteus

Проектирование принципиальной схемы устройства обычно осуществляют с помощью кнопок на левой панели инструментов. Рассмотрим их назначение.

Режим выбора. При нажатой кнопке пользователь получает возможность редактировать схему устройства, т.е. выделять отдельные компоненты или структурные блоки, копировать их в буфер, удалять, масштабировать, и т.д. Можно позиционировать схему, перемещая в окне предварительного просмотра (справа от кнопки) перекрестие «прицела» при нажатой левой кнопке мыши. Вторичное нажатие кнопки мыши фиксирует положение схемы в окне редактирования. Используя также кнопки масштабирования на верхней панели инструментов, можно настраивать любой фрагмент схемы в удобном для пользователя масштабе.

Компоненты. При нажатии кнопки в соседнем с ней окне появляется список используемых в открытом проекте компонентов (если проект еще не создан, список пустой). Выделив левой кнопкой мыши любой из компонентов списка, можно вторичным нажатием левой кнопки установить его в произвольном месте окна редактирования. Двукратное нажатие правой кнопки мыши удаляет компонент из окна редактирования. Важной является кнопка **«P»**, расположенная под окном предварительного просмотра. Ее нажатие открывает окно поиска компонентов **Pick Device** (рис. 3.2), предназначенное для входа в

библиотеку компонентов системы Proteus. В этом окне все компоненты структурированы по категориям, подкатегориям и изготовителям. Библиотека имеет очень большое количество самых разных компонентов. Например, в категории Microprocessors представлено 16 различных семейств микроконтроллеров, включая семейство AVR, а в нем, в свою очередь, представлено 80 различных типов микроконтроллеров. Каждый компонент имеет краткое описание и графическое изображение на принципиальной схеме. Нажатие в окне клавиши **«Ok»** помещает выбранный библиотечный компонент в список компонентов проекта, откуда его можно переместить в окно редактирования. Таким способом все необходимые для проекта компоненты из библиотеки перемещаются в окно редактирования.

Точка соединения. При нажатой кнопке можно соединить любое пересечение проводников на схеме. Соединение выводов компонентов друг с другом можно осуществлять и при других нажатых кнопках, например, сразу после установки компонентов в окне редактирования, т.е. при нажатой кнопке **«Компоненты»**.

Метка соединения. При нажатой кнопке можно присвоить любому проводнику имя. Это позволяет соединять отдельные выводы компонентов и цепи условно, без явного использования проводников, что часто позволяет улучшить восприятие проектируемой схемы устройства.

Текстовый скрипт. При нажатии кнопки можно вставить текст в любое место схемы в окне редактирования. Полезно для создания комментариев при проектировании устройств.

Шина. При нажатой кнопке можно проложить на принципиальной схеме шину, состоящую из нескольких проводников. Например, для микроконтроллера ATMega128 соединение портов A и C с внешними устройствами (статической памятью, ЖК-индикатором и т.д.) осуществляется с помощью шин.

Субсхема. Кнопка позволяет создать субсхемы, представляющие собой некие функциональные блоки с выводами-соединителями.

Терминал. Кнопка позволяет установить на принципиальной схеме устройства такие элементы, как питание, общая шина, межблочные соединения, выводы.

Пины устройства. Кнопка позволяет добавить вывод к создаваемому компоненту.

Диаграмма. При нажатой кнопке в окне редактирования можно установить набор графических инструментов, предоставляющих пользователю широкие возможности по отображению сигналов, их детальному анализу и математической обработке, а также сохранению результатов моделирования.

Генератор. Нажатие кнопки выводит список генераторов сигналов различной формы: синусоидальный, импульсный, экспоненциальный и т.д. Очень удобный и важный инструмент для задания тестовых сигналов при отладке проектируемого устройства.

Щуп напряжения. Предназначен для указания точки проводника, в которой необходимо измерить напряжение, и присвоения имени участку цепи с данной точкой. Используется совместно с графическими средствами измерения напряжения (см. кнопку **«Диаграмма»**).

Щуп тока. Предназначен для решения тех же задач, что и щуп напряжения, т.е. выбора и присвоения имени участку цепи, в которой производится измерение тока.

Виртуальные инструменты. При нажатии кнопки появляется список виртуальных инструментов, среди которых генератор сигналов специальной формы, 4-канальный осциллограф, вольтметры и амперметры постоянного и переменного токов, виртуальный терминал и др. Виртуальный генератор сигналов, в отличие от выше рассмотренного генератора (вызываемого кнопкой **«Генератор»**), может изменять параметры тестового сигнала непосредственно в процессе моделирования работы устройства. Для этого на его лицевой панели имеются специальные органы управления, позволяющие оперативно, без остановки процесса моделирования изменять форму сигнала, его частоту и амплитуду. С помощью осциллографа можно исследовать сигнал в реальном масштабе времени. Как и у других виртуальных инструментов в нем имеются органы управления, позволяющие изменять частоту развертки, чувствительность, настраивать цветовую гамму изображения (луч, дисплей, сетка, курсор) и т.д.

Для измерения напряжения или тока к участку цепи на принципиальной схеме устройства подключают вольтметр или амперметр так, как если бы это были реальные приборы, после чего запускают процесс моделирования. При остановленном процессе моделирования можно изменить диапазон измерения, т.е. превратить вольтметр, например, в милливольтметр. Виртуальный терминал позволяет смоделировать обмен данными между микроконтроллером и персональным компьютером, использующими для этого последовательный RS-интерфейс. Другие виртуальные инструменты позволяют решать задачи, связанные с отладкой работы периферийных устройств, взаимодействующих с микроконтроллером посредством SPI- или I2C-интерфейсов.

3.3. Основные приемы работы с системой Proteus

Процесс моделирования электронного устройства начинается с проектирования его принципиальной схемы. Для этого необходимо найти в библиотеке Proteus нужные компоненты, разместить их в окне редактирования и соединить проводниками. При необходимости в схему добавить источники питания и общую шину, а также виртуальные средства измерения и контроля, например, вольтметры и осциллографы. Для того, чтобы читателю проще было приобретать навыки работы с системой Proteus, рассмотрим основные действия, которые ему необходимо совершить, прежде чем запустить процесс моделирования.

Формирование списка компонентов для проектируемого устройства. Для этого на левой панели инструментов следует нажать кнопку **«Компоненты»** и

затем кнопку **«P»** (под окном предварительного просмотра). Откроется окно выбора библиотечных компонентов **Pick Devices** (рис. 3.2), которые сгруппированы по категориям, подкатегориям и фирмам-изготовителям. Кроме обычных компонентов таких, как резисторы, конденсаторы, транзисторы, микросхемы библиотека содержит широкий набор вспомогательных компонентов и устройств. Это и оптоэлектронные устройства (буквенно-цифровые и графические индикаторы и дисплеи, оптроны), электромеханические устройства (электродвигатели, реле), разнообразные разъемы и коммутаторы, сенсоры, функциональные блоки с заданной передаточной функцией и многое другое. Каждый компонент имеет свой графический образ на принципиальной схеме, что позволяет быстро определить его назначение и отличительные особенности.

Пусть, например, в проектируемом устройстве имеется микроконтроллер ATmega128. Выберем в окне **Pick Devices** категорию Microprocessor ICs, подкатегорию AVR Family и в колонке устройств – ATmega128. В правом верхнем окне появится графический образ компонента. Нажмем кнопку «**Ok**», компонент ATmega128 появится в списке (справа от панели инструментов). Аналогично выбираются другие компоненты, входящие в состав проектируемого устройства. Таким способом формируется список нужных компонентов, который в дальнейшем можно корректировать (добавлять новые компоненты или удалять из списка ненужные). Для включения в список можно использовать окно поиска по маске, в которое вводится имя компонента (или начальная часть имени).

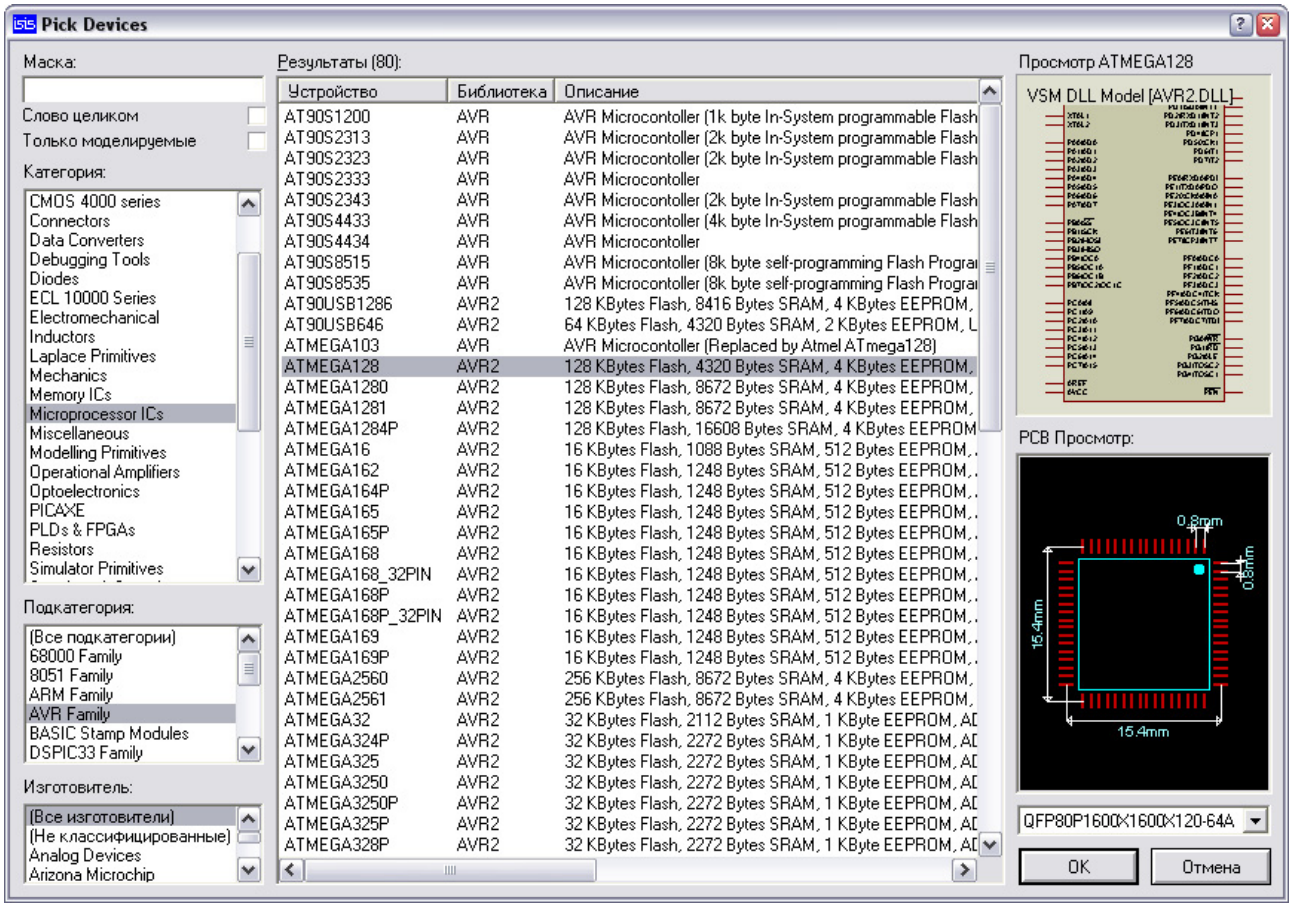

Рис. 3.2. Окно выбора библиотечных компонентов **Pick Devices**

Размещение компонентов в окне редактирования. Для этого с помощью мыши или клавиатуры следует выделить нужный компонент в списке, затем установить курсор в свободном месте окна редактирования и нажать левую кнопку мыши – компонент появится в указанном месте. Таким способом размещаются все компоненты. В дальнейшем их положение можно корректировать, перемещая или поворачивая на нужные углы, копировать через буфер или удалять из окна редактирования. Все эти операции можно совершать как с отдельными компонентами, так и с блоками компонент, выделив предварительно нужный блок при нажатой левой кнопке мыши. Используя выпадающее меню при двойном нажатии левой кнопки мыши на компоненте, можно редактировать его свойства, например, номиналы резисторов или конденсаторов. Для удобства восприятия схемы можно на ней поместить дополнительную текстовую информацию, например, функциональное назначение какой-то группы компонентов. Для этого в меню, выпадающем при нажатии правой кнопки мыши в свободном месте окна редактирования, выбрать пункт **«Разместить > Текстовый скрипт»**, после чего выбрать левой кнопкой нужное место, где будет располагаться текст.

Соединение компонентов между собой. После размещения компонентов в окне редактирования их следует соединить друг с другом в соответствии с принципиальной схемой устройства. Для этого следует подвести курсор к выводу компонента (при этом на схеме появится образ контактной площадки), нажать левую кнопку мыши и провести проводник к другому выводу компонента или другому проводнику. После этого для фиксации соединения нажать левую кнопку мыши. Если в процессе соединения компонентов проводник не устанавливается в задуманное проектировщиком положение, то можно прокладывать проводник по частям, фиксируя положение точек изгиба проводника нажатием левой кнопки мыши. Если после формирования проводников потребуется соединить какие-то точки их пересечений, то это можно сделать, выбрав на левой панели инструментов кнопку **«Точка соединения»**. Разумеется, соединение на схеме проводников друг с другом можно делать и непосредственно при их прокладке.

Для лучшего восприятия схемы вместо непосредственного соединения иногда используют присвоение одинакового имени двум или нескольким проводникам, после чего они считаются электрически соединенными друг с другом. Для этого на левой панели инструментов следует выбрать кнопку **«Метка соединения»**, после чего подвести курсор к нужному проводнику, нажать левую кнопку мыши и в открывшемся окне ввести имя (метку) проводника. У отмеченных таким способом проводников один конец может оказаться не подключенным, т.е оказаться оборванным. Обрыв проводника формируется двойным нажатием левой кнопки мыши в точке обрыва.

Другим способом улучшить восприятие схемы при наличии большого количества проводников является использование шин, которые могут включать в себя несколько проводников. Для этого на левой панели инструментов следу-

ет нажать кнопку **«Шина»**, после чего проложить шину в нужном месте окна редактирования и подсоединить отдельные проводники шины к нужным точкам проектируемого устройства. При этом все используемые проводники в шине должны иметь свои имена (метки). Пример использования шины представлен на рис. 3.7.

В процессе проектирования принципиальной схемы часто возникает необходимость ее масштабирования, для чего используются соответствующие кнопки на верхней панели инструментов. Для перемещения всей схемы используется окно предварительного просмотра, расположенное в левом верхнем углу главного окна. Установив курсор в это окно, и перемещая перекрестие «прицела» при нажатой левой кнопке мыши, можно перемещать принципиальную схему по окну редактирования. Еще одно нажатие левой кнопки фиксирует положение схемы. При получении нежелательных результатов не забывайте, что всегда есть возможность отменить последние действия. Кроме того, всегда есть возможность вернуть схему в исходное положение, нажав в верхней панели инструментов кнопку **«Вписать во весь лист»**.

Формирование общей шины и источников питания. Любое устройство имеет общую шину и одно или несколько напряжений питания. Для их размещения в окне редактирования следует на левой панели инструментов нажать кнопку **«Терминал»** и выбрать пункт **«GROUND»**, после чего дважды нажать левую кнопку мыши в нужном месте окна редактирования. Источник питания устанавливается после выбора пункта **«POWER»** аналогично, но для него дополнительно требуется задать величину питающего напряжения. Для этого надо дважды нажать левой кнопкой мыши на графическом образе источника и в появившемся окне набрать значение напряжения источника, например, +5.2V.

Использование источников тестовых сигналов. Для моделирования работы проектируемого устройства часто используются тестовые сигналы, поступающие на вход устройства или в какие-то другие участки схемы. Для подключения источников тестовых сигналов в принципиальную схему устройства следует на левой панели инструментов нажать кнопку **«Генератор»** и выбрать нужный тип источника сигнала (синусоидальный, импульсный и т.д.). После этого следует дважды нажать левую кнопку мыши в окне редактирования, затем также дважды нажать левую кнопку на графическом образе генератора и задать параметры тестового сигнала. Например, для синусоидального сигнала (рис. 3.3) это амплитуда, пиковое или среднеквадратичное значения, частота или период сигнала, фаза и коэффициент затухания, величина постоянной составляющей в синусоидальном сигнале (смещение). В Proteus имеется широкий выбор различных источников тестового сигнала. Возможен, например, выбор источника сигнала, заданного в цифровой форме и сохраненного в файле. Есть возможность задать нужную форму тестового сигнала, начертив его с помощью мыши на графике в координатах напряжение – время. Для этого необходимо выбрать настраиваемый вид аналогового сигнала.

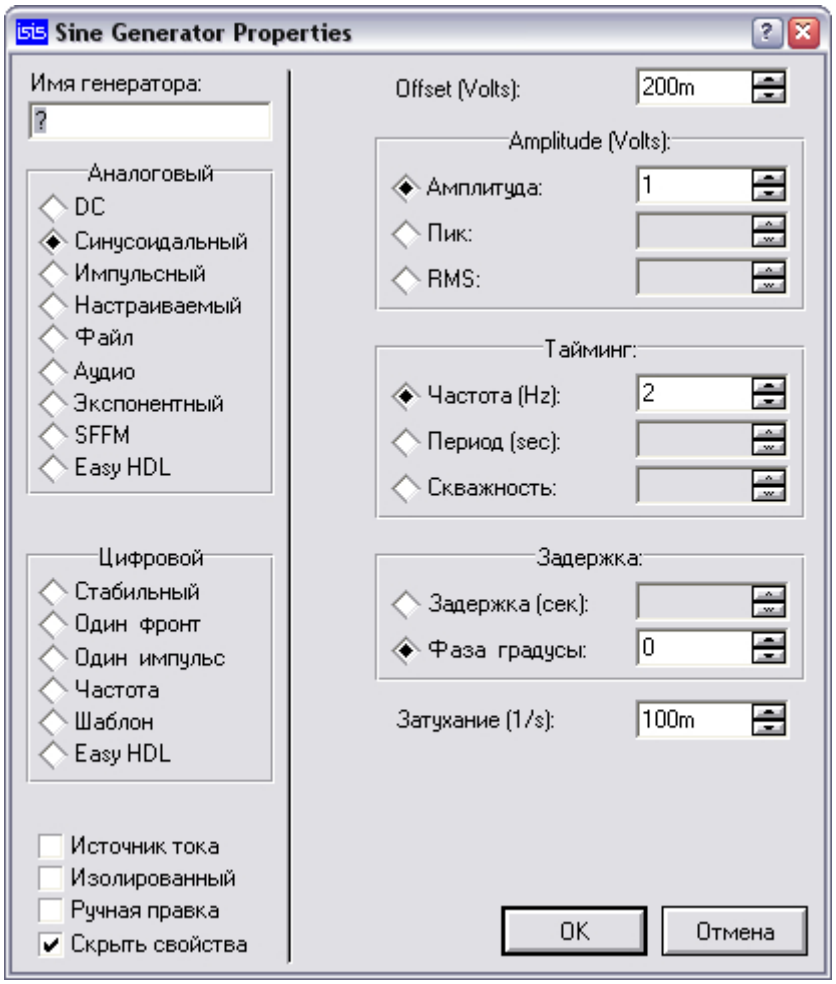

Рис. 3.3. Окно задания параметров синусоидального тестового сигнала

Использование виртуальных инструментов. В процессе моделирования работы проектируемого устройства часто возникает необходимость измерить параметры тестового сигнала в отдельных точках схемы, проверить работу отдельных блоков устройства, в частности, работу различных интерфейсов, имеющихся в микроконтроллерах. Для решения этих и других подобных задач используют виртуальные инструменты, вызываемые кнопкой **«Виртуальные инструменты»** на левой панели инструментов. Среди этих средств наиболее часто используются осциллограф, вольтметры и амперметры, а также генератор сигналов специальной формы и виртуальный терминал, предназначенный для передачи информации от микроконтроллера в компьютер.

Для установки виртуальных инструментов следует выбрать их в окне списка виртуальных инструментов (см. рис. 3.1), после чего нажать левую кнопку мыши в месте окна редактирования. Появится графический образ инструмента, который затем следует подключить к контролируемым точкам схемы или соответствующим выводам компонентов. После запуска процесса моделирования графические образы ряда инструментов (осциллографа, генератора сигналов, логического анализатора и др.) преобразуются в лицевые панели, позволяющие с помощью органов управления настраивать инструменты на реше-

ние конкретных задач. Например, в 4-канальном осциллографе можно изменять развертку по оси времени и чувствительность каждого из каналов, производить инвертирование сигналов или складывать их друг с другом. Можно для каждого из каналов сделать вход открытым или закрытым, что позволит измерять либо весь сигнал, либо только его переменную составляющую. Нажав правую кнопку мыши на лицевой панели и выбрав в появившемся окне пункт **«Setup…»**, можно изменить цветовую гамму дисплея осциллографа.

Для вольтметра и амперметра имеется возможность устанавливать диапазон измерения, превращая, например, вольтметр в милливольтметр или микровольтметр. Для генератора сигналов специальной формы (рис. 3.4) можно задать форму выходного сигнала, а также его амплитуду и частоту. При этом все регулировки производятся непосредственно в процессе моделирования, полностью имитируя работу реального прибора.

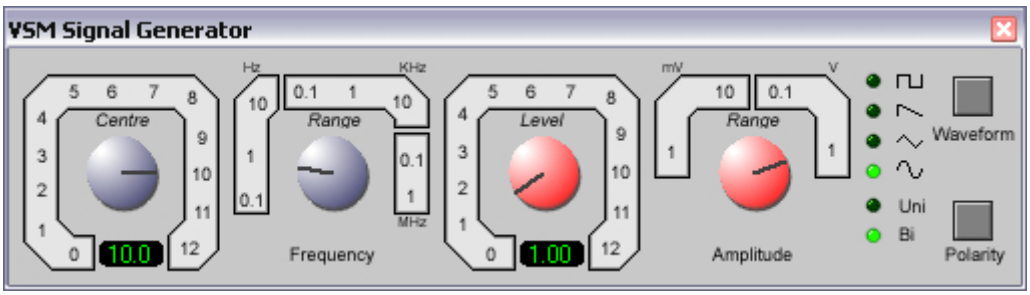

Рис. 3.4. Лицевая панель виртуального генератора сигналов

При использовании виртуального терминала, предназначенного для приема и отображения информации от микроконтроллера, необходимо до запуска процесса моделирования установить его основные рабочие настройки (рис. 3.5). При этом важно, чтобы они были полностью согласованы с настройками микроконтроллера. Это в первую очередь относится к скорости передачи данных (стандартные значения от 300 до 57600 бод), количеству бит данных в посылке (7 или 8 бит), биту паритета, количеству стоп-бит в посылке. Остальные настройки можно оставить без изменения.

Использование графопостроителей для анализа сигналов. В процессе моделирования часто возникают задачи, связанные с анализом формы сигналов, точным определением напряжений или токов в контролируемых точках схемы в заданные моменты времени, математической обработки сигналов, например, определении их спектрального состава и т.д. Эти задачи решает группа графических инструментов, вызываемых нажатием кнопки **«Диаграмма»** на левой панели инструментов, в результате чего в соседнем окне появляется перечень графопостроителей. В Proteus имеется довольно широкий набор средств анализа аналоговых и цифровых сигналов. Например, для временного анализа аналоговых сигналов в наибольшей степени подходит аналоговый графопостроитель **«ANALOGUE»**, для спектрального анализа – спектроанализатор **«FOURIER»**, осуществляющий Фурье-преобразование сигнала и визуальное представление спектра на графике.

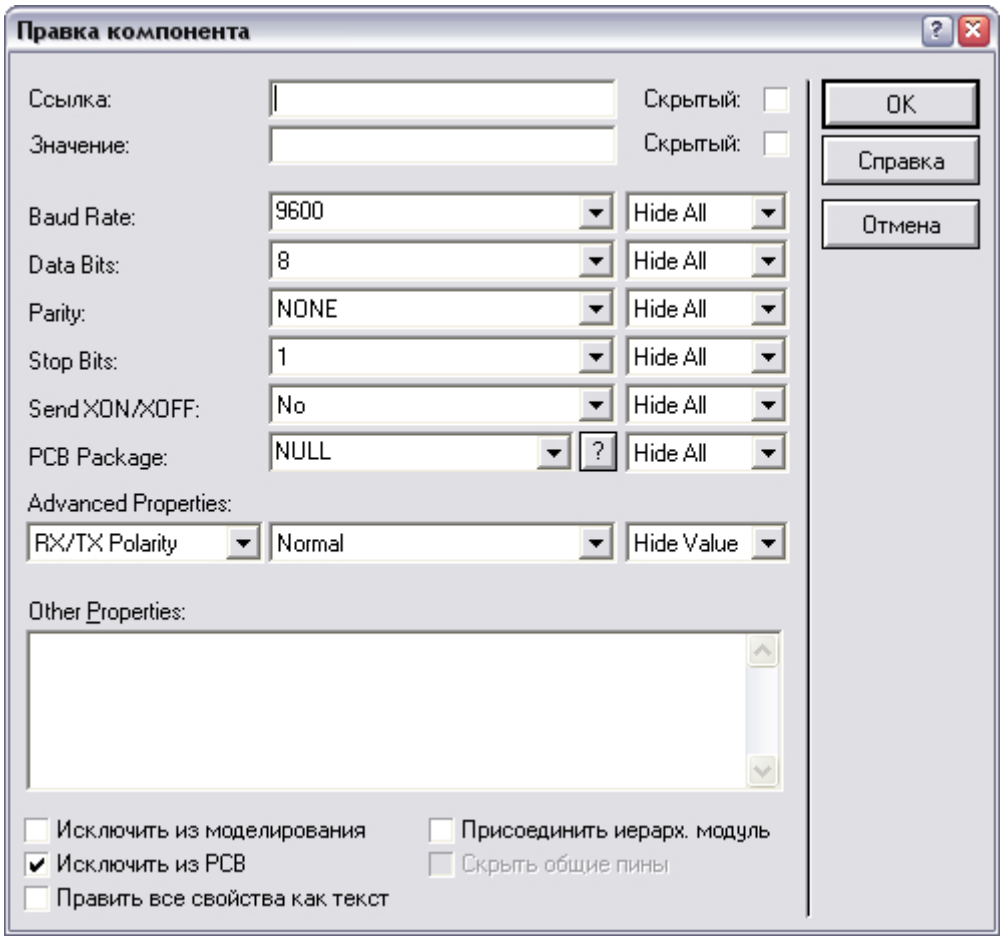

Рис. 3.5. Окно задания настроек виртуального терминала

Для использования нужного инструмента необходимо очертить в свободном месте окна редактирования прямоугольник и нажать левую кнопку мыши, в результате чего появится графический образ выбранного инструмента. Используя правую кнопку мыши и выпадающее при этом окно, можно развернуть графический образ инструмента в рабочее окно на весь экран, далее установить параметры моделирования и контрольные точки на схеме, в которых будет производиться анализ (добавить трассы). Для этого необходимо отметить на принципиальной схеме устройства контрольные точки, выбрав на левой панели инструментов щупы напряжения или тока, затем нажать левую кнопку мыши в нужной точке схемы. После этого часть цепи, содержащая выбранную таким образом точку, автоматически получит свое имя (имя трассы) и оно будет включено в перечень трасс, доступных для анализа. Добавив в проект нужные трассы, и запустив процесс моделирования диаграммы, можно получить графики временных зависимостей сигналов или их спектры. После этого, используя средства масштабирования, можно с помощью курсора детально исследовать график. Имеется возможность выводить в одно и то же окно несколько графиков, настраивать их цветовую гамму и т.д.

«Прошивка» кода программы в микроконтроллер. Если проектируется микропроцессорное устройство, то для его схемотехнического анализа необхо-

димо «прошить» управляющую программу в память микроконтроллера. В системе Proteus это фактически означает установку в проекте пути к каталогу, в котором находится управляющая программа, и ее имени. Желательно при этом, чтобы разрабатываемый в Proteus проект и управляющая программа, разработанная в ICCAVR, были сохранены в одном каталоге. В том случае после любых исправлений в коде программы необходимости в ее новой «прошивке» нет, после компиляции она автоматически подключается к проекту.

Для «прошивки» управляющей программы следует двойным нажатием левой кнопки мыши на графическом изображении микроконтроллера открыть окно для его редактирования (рис. 3.6). Затем следует произвести нужные настройки, а именно, установить частоту синхронизации и значения бит SKSEL Fuses (по умолчанию стоит RC-генератор на 1 МГц). После этого в окне **Program File** следует выбрать в папке с проектом hex- или cof-файл управляющей программы, сгенерированный компилятором, «прошив» его тем самым в память микроконтроллера. Остальные параметры, показанные в окне, можно оставить без изменения.

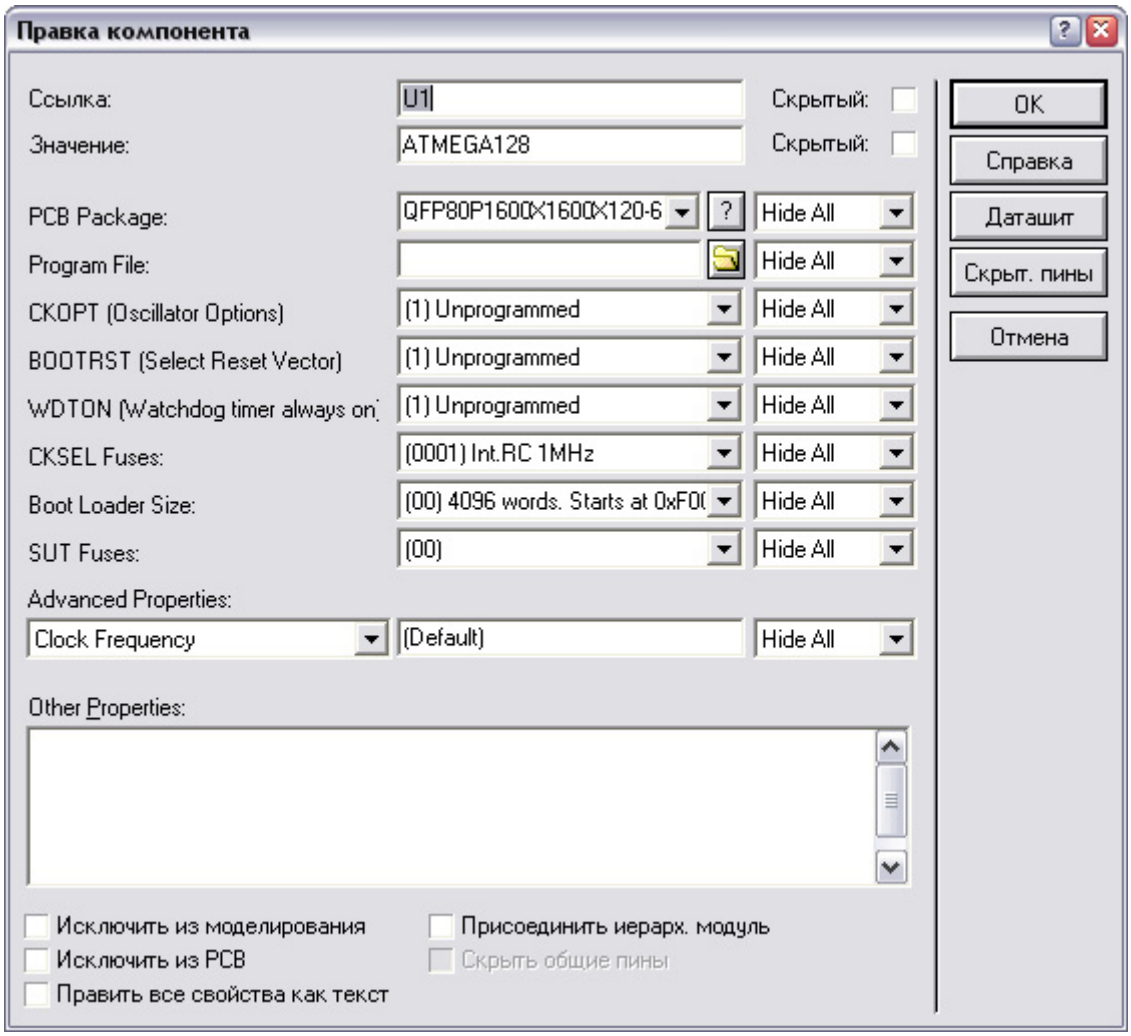

Рис. 3.6. Окно редактирования микроконтроллера

Если отладка управляющей программы не предполагается, то можно «прошить» hex- или cof-файл управляющей программы. В противном случае следует выбирать только cof-файл. После «прошивки» программы проект готов к работе. Отладка производится в случае, если программа имеет скрытые ошибки в функционировании, которые проектировщик не может обнаружить при обычном чтении текста программы. Для отладки требуется предварительно указать путь к месту нахождения исходного С-файла. Это осуществляется выбором пункта меню **«Исходник> Добавить/удалить файлы исходника»** и установкой имени исходного С-файла.

Запуск процесса моделирования и отладки. Когда принципиальная схема устройства спроектирована, можно запустить процесс моделирования, нажав кнопку **«Воспроизвести»** в нижней части главного окна. При необходимости внесения каких-либо исправлений в принципиальную схему устройства процесс моделирования следует остановить, нажав кнопку **«Стоп»**. Если проектируемое устройство содержит микроконтроллер и требуется отладка управляющей программы, то необходимо выбрать пиктограмму **«Шаг»**, после чего в пункте меню **Отладка** выбрать из сгенерированного отладчиком списка код программы (AVR Source Code). Это позволит визуально наблюдать в пошаговом режиме выполнение программы и проконтролировать алгоритм ее работы.

Следует отметить, что одно и то же действие можно осуществить различными способами, используя пункты меню, кнопки инструментальных панелей или клавиши мыши. Например, при работе с аналоговым анализатором (кнопка меню **«Диаграмма»**) установку дополнительной трассировки можно осуществить с помощью кнопки на левой панели инструментов, выпадающего меню при нажатии правой кнопки мыши или пункта меню **Диаграмма>Добавить трассировку**. Описать все возможные варианты действий пользователя не представляется возможным. Да это и не нужно. Приобретение необходимых навыков работы с системой Proteus возможно только опытным путем, используя метод проб и ошибок. Это можно делать безбоязненно, т.к. любые действия проектировщика никаких негативных последствий для микроконтроллера и других электронных компонентов вызвать не могут. Кроме того, не следует забывать про пункт меню **Справка**, по которой можно получить много полезной информации.

3.4. Разработка проекта в Proteus на примере цифрового вольтметра

Рассмотрим детально процесс разработки проекта на примере микропроцессорного цифрового вольтметра, который производит преобразование измеряемого напряжения в код и отображает результат измерения на жидкокристаллическом индикаторе. Для повышения быстродействия и точности измерений будем использовать внешний АЦП с последовательным выходным кодом, который взаимодействует с микроконтроллером ATmega128 посредством SPI-интерфейса. У большинства современных АЦП с последовательным выходом разрядность кода равна 16, но в библиотеке Proteus таких АЦП нет, есть только

10-разрядный АЦП. Для моделирования работы вольтметра и отладки программного обеспечения достаточно и его. Напряжение на вход АЦП будет поступать с потенциометра, подключенного к источнику питания +5V. Для контроля правильности функционирования микропроцессорного вольтметра будем в процессе моделирования дополнительно измерять напряжение каким-нибудь виртуальным инструментом, например DC-вольтметром.

Будем считать, что управляющая программа для микроконтроллера разработана в среде ICCAVR и все файлы (C, H, HEX, COF, PRJ и т.д.) хранятся в папке с именем, например, Work. Откроем систему Proteus и в пункте меню **Файл** выберем **Новый проект**. По запросу системы выберем шаблон по умолчанию Default и сохраним пустой проект под каким-нибудь именем в папке, где находятся файлы управляющей программы, в данном случае это папка Work. Пусть имя проекта будет Voltmeter.DSN (расширение DSN формируется системой автоматически).

Сформируем список нужных компонентов устройства. Откроем окно поиска библиотечных компонентов **Pick Devices** (см. рис. 3.2). Нам потребуется микроконтроллер. Выберем категорию Microprocessor ICs, подкатегорию AVR Family и в колонке устройств – ATmega128. В правом верхнем окне появится графический образ компонента. Нажмем кнопку **«Ok»**, компонент ATmega128 появится в списке.

Таким же способом найдем в **Optoelectronics>Alphanumeric LCDs** буквенно-цифровой ЖК-индикатор LM032L (4 строки по 20 символов в строке) и добавим в список компонентов. Найдем в **Data Converters>A/D Converters** 10-разрядный АЦП MCP3001 с последовательным SPI выходом и добавим в список. Добавим в список также потенциометр POT-LIN, находящийся в **Resistors>Variable**. Потенциометр является интерактивным, что позволяет изменять снимаемое с него напряжение непосредственно в процессе моделирования.

Разместим все компоненты вольтметра в окне редактирования, выделив их в списке и нажав в нужном месте левую кнопку мыши. При необходимости любой компонент можно переместить, развернуть или удалить совсем из окна редактирования, для чего используются кнопки мыши или верхняя панель инструментов. Добавим в окно редактирования генератор сигналов (из виртуальных инструментов), источник питания и общую шину, нажав кнопку **«Терминал»** на левой панели инструментов. При этом напряжение питания +5V и метку GND необходимо задать в окне редактирования меток компонента, вызываемого двукратным нажатием левой кнопки мыши. Примерное размещение компонентов показано на рис. 3.7.

Далее следует соединить компоненты устройства друг с другом. Вначале можно проложить две шины, соединяющие порты A и C микроконтроллера с ЖК-индикатором, пометив при этом все проводники, образующие шины. Если какие-то проводники нецелесообразно прокладывать, так как это ухудшит восприятие принципиальной схемы, то можно использовать метки, например метка CS ADC или GND.

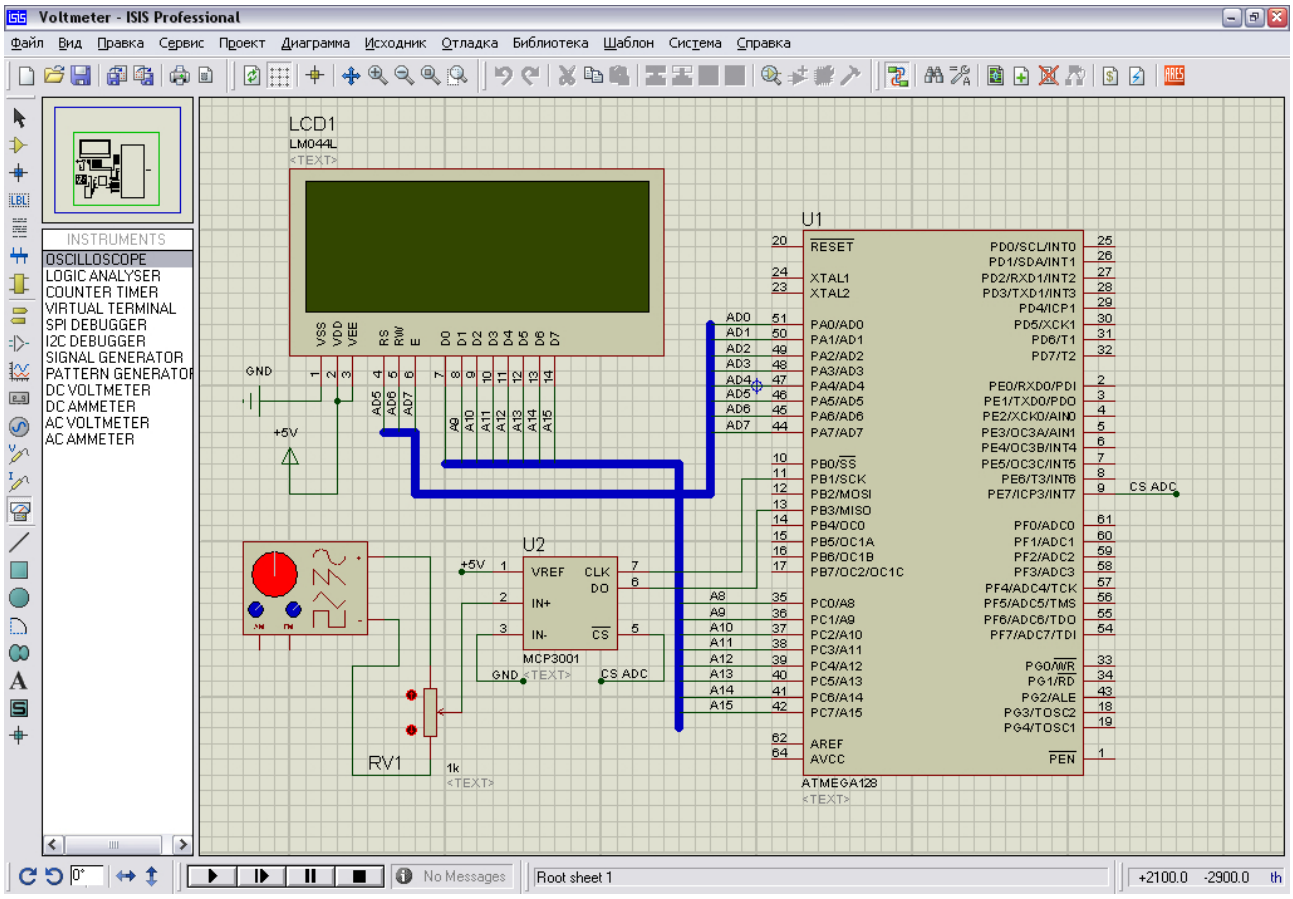

Рис. 3.7. Главное окно с проектом Voltmeter

Следует обратить внимание на интерактивный потенциометр, его движок можно перемещать нажатием кнопки мыши на кружках справа от потенциометра. Если в процессе редактирования чертеж смещается из центра, изменяется масштаб, проводники прокладываются не там, где планировал проектировщик, помните про возможность отмены последних операций и возврата схемы в центр окна редактирования (пиктограммы на верхней панели инструментов), а также не забывайте регулярно сохранять проект в памяти.

После того, как все компоненты устройства будут соединены друг с другом в соответствии с его принципиальной схемой необходимо «прошить» управляющую программу в память микроконтроллера. Для этого следует дважды нажать левую кнопку мыши на графическом образе микроконтроллера, в открывшемся окне **«Правка компонента»** (рис. 3.6) левой кнопкой открыть окно **«Program File»**, после чего установить имя файла с управляющей программой (cof- или hex-файл). Это означает привязку программного кода к проекту, т.е. «прошивку» управляющей программы в память микроконтроллера. После этого проект Voltmeter готов к работе. Для запуска процесса схемотехнического моделирования необходимо в нижней части главного окна нажать кнопку **«Воспроизвести»**. Разумеется, для того, чтобы проект заработал, необходимо «прошить» реальную управляющую программу для микроконтроллера. Для того чтобы разработать ее, читателю потребуется изучить примеры проектирования микропроцессорных устройств, рассмотренные в следующей главе.

4. ПРАКТИЧЕСКИЕ ПРИМЕРЫ ПРОЕКТИРОВАНИЯ МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ

В микропроцессорных устройствах микроконтроллеры обычно решают следующие задачи.

1. Установка на выводах микроконтроллера в определенные моменты времени высокого или низкого уровня напряжения (лог.1 или лог.0). Это требуется, например, для формирования предупреждающей световой или звуковой сигнализации при возникновении определенных ситуаций, контролируемых микроконтроллером. Если микроконтроллер управляет каким-либо объектом, то сигнал с выхода после усиления может поступить на исполнительные устройства, например, электродвигатели, оказывающие воздействие на объект. Можно привести множество других примеров, например, включение и выключение реле, коммутаторов, мультиплексоров, а также формирование сигналов для запуска каких-либо процессов или устройств.

2. Опрос состояния выводов микроконтроллера. Это может быть, например, установление факта нажатия на какую-либо управляющую кнопку устройства, после чего микроконтроллер должен выполнить какие-то действия. Это может быть опрос так называемых бинарных датчиков, например, индуктивных датчиков положения, биметаллических терморегуляторов или герконов, которые замыкают или размыкают цепь сигнализации при наступлении определенных событий. При этом реакция микроконтроллера на событие может быть реализована с использованием системы внешних прерываний или прерываний таймеров/счетчиков.

3. Передача буквенно-цифровой информации на индикаторные устройства. Это могут быть результаты измерений или какая-то информация для пользователя, например, пункты системы меню, с помощью которых пользователь может инициировать те или иные действия.

4. Передача буквенно-цифровой информации в компьютер или какое-то другое удаленное устройство. Обычно для этого используется какой-либо стандартный последовательный интерфейс, входящий в состав микроконтроллеров.

5. Преобразование аналогового сигнала в цифровой код для последующей обработки. Это можно выполнить с помощью встроенных в микроконтроллер АЦП или с помощью внешних АЦП, обладающих, как правило, более широкими возможностями в плане разрешения и быстродействия. Возможна и обратная задача – преобразование цифрового кода в аналоговый сигнал с помощью внутренних или внешних ЦАП с целью формирования управляющего сигнала для исполнительных устройств.

6. Формирование на выводах электрических импульсов с изменяющейся длительностью или периодом следования – импульсов с широтно-импульсной модуляцией (ШИМ-импульсов). Возможна и обратная задача – определение временных характеристик импульсов, поступающих на входы микроконтролле-

ра. Такая задача обычно решается с помощью таймеров/счетчиков, входящих в состав микроконтроллеров.

Ниже рассмотрен ряд примеров проектирования и программирования микропроцессорных устройств, в которых задействованы те или иные периферийные модули микроконтроллера ATmega128. Они оформлены в виде заданий, в которых приведен примерный порядок действий проектировщика. Это позволяет читателю самостоятельно изучить все особенности работы микроконтроллера. Также приведены и тексты исходных кодов программ с необходимыми комментариями. Разумеется, для каждой задачи читатель может самостоятельно разработать управляющую программу. В Приложении приведен ряд программ (C- и H-файлы), с помощью которых осуществляется взаимодействие микроконтроллера с внешними периферийными устройствами, в частности, ЖК-индикатором и АЦП. Не следует забывать, что любая задача имеет несколько вариантов решения, поэтому для любого из заданий читатель может предложить свои решения, отличающиеся от тех, что приведены в книге.

Для получения практических навыков по проектированию и программированию микропроцессорных устройств их разработка ведется в системе Proteus, что позволяет оперативно проверить корректность работы спроектированных устройств, и в случае обнаружения ошибок внести необходимые исправления в управляющую программу. Перед изучением данного материала рекомендуется повторить п. 2.3, в котором рассмотрены примеры программирования периферийных устройств микроконтроллера, а также п. 3.3, в котором описаны основные приемы работы с системой Proteus.

4.1. Изучение работы виртуальных инструментов в системе схемотехнического моделирования Proteus

Задание 1. В Proteus спроектировать принципиальную электрическую схему, содержащую виртуальный генератор сигналов специальной формы и виртуальный осциллограф (рис. 4.1). Сигналы с генератора поступают на резистивную нагрузку. С помощью осциллографа исследовать форму сигналов и их основные параметры, такие как амплитуда, частота и скважность.

Задание 2. Заменить в схеме виртуальный генератор на обычный генератор сигналов, устанавливаемый кнопкой **«Генератор»** на левой панели инструментов и провести аналогичные измерения, варьируя форму выходных сигналов и их основные параметры.

Задание 3. Подключить к нагрузке вольтметр переменного тока и измерить действующее значение напряжения, изменяющегося по синусоидальному закону.

Задание 4. Подключить к нагрузке пробник для измерения постоянного напряжения, задать период синусоидального сигнала около 100 с и произвести измерения напряжения на нагрузке с помощью пробника.

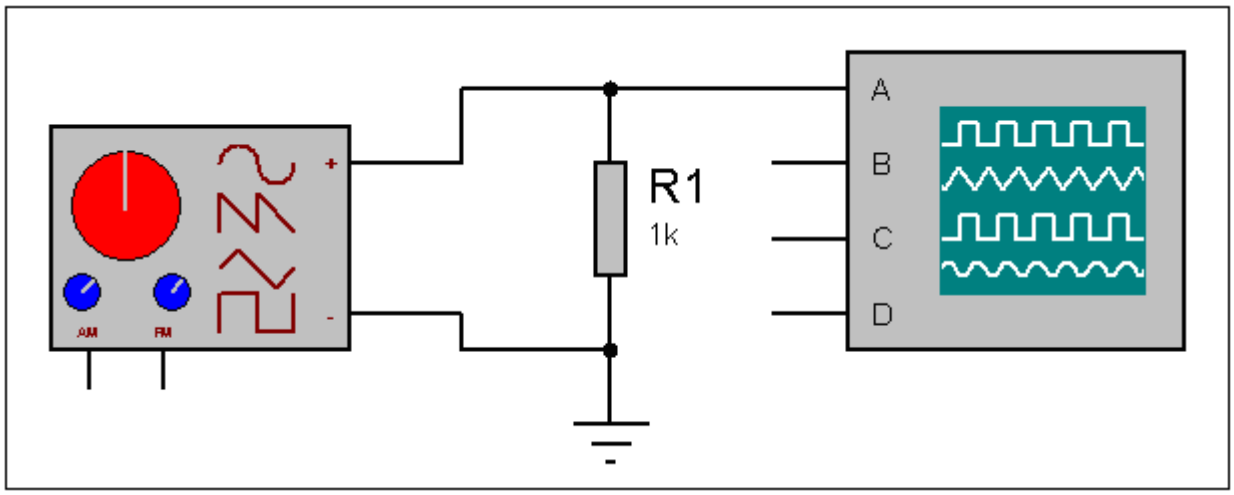

Рис. 4.1. Схема для изучения системы схемотехнического моделирования Proteus

Задание 5. Дополнить схему аналоговым графопостроителем **«ANALOGUE»** (кнопка **«Диаграмма»** на левой панели инструментов) и произвести измерение и анализ синусоидального сигнала частотой около 20 Гц. Конечное время моделирования установить равным 5 с.

Задание 6. Дополнить схему графическим спектроанализатором **FOURIER** и произвести измерение спектра синусоидального и импульсного сигналов частотой около 10 Гц. Конечное время моделирования установить равным 10 с, максимальную частоту – 100 Гц, разрешение – 1 Гц. Перемещая курсор по спектру, измерить амплитуду основной гармоники и сравнить ее со значением, устанавливаемым в настройках генератора синусоидального сигнала.

4.2. Изучение взаимодействия микроконтроллера с кнопкой и светодиодом

Задание 1. В среде Proteus спроектировать схему, содержащую микроконтроллер ATmega128, к одному из выводов которого (например, выводу PB2) подключен светодиод с токозадающим резистором (рис. 4.2). К другому выводу (например, выводу PB1) с делителя напряжения поступает напряжение лог.1. С помощью кнопки состояние вывода PB1 можно переключать из лог.1 в лог.0 и обратно.

Задание 2. В среде ICCAVR разработать управляющую программу для микроконтроллера, с помощью которой осуществляется мигание светодиода. Для этого осуществить следующие действия:

- настроить выводы, к которым подключен светодиод (например, PB2) – на выход; а выводы, к которым подключена кнопка (например, PB1) – на вход;
- с помощью оператора **while** сформировать «бесконечный» цикл и переключать вывод PB2 из состояния лог.1 в лог.0 и обратно, причем переключение осуществлять с временной задержкой, реализованной программно с помощью операторов цикла (**for**, **while** и т.д.);

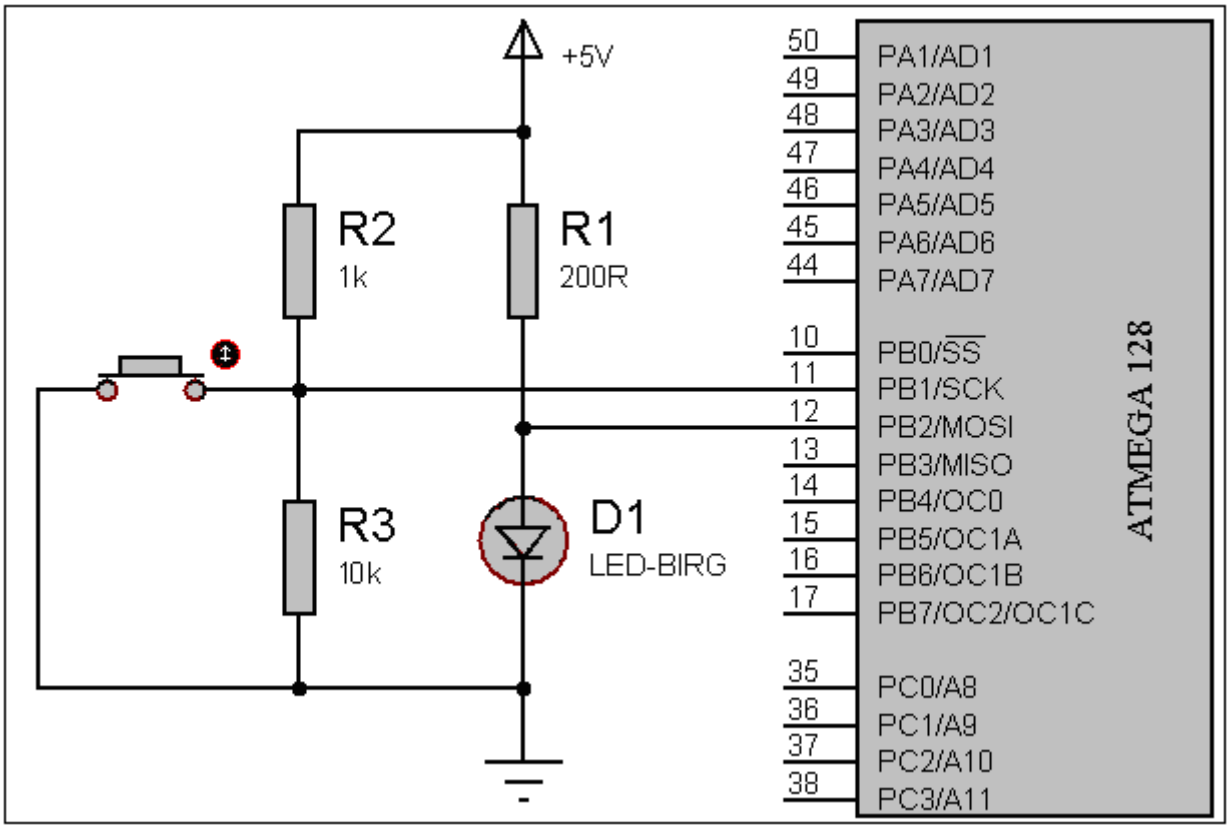

Рис. 4.2. Схема для изучения взаимодействия микроконтроллера с кнопкой и светодиодом

- произвести компиляцию программы, получив в случае успешной компиляции файлы для «прошивки» программы в память контроллера (файлы с расширением hex и cof).

Задание 3. В Proteus «прошить» в память микроконтроллера cof-файл управляющей программы и кнопкой **«Воспроизвести»** инициировать начало ее работы. Экспериментально подобрать параметры цикла в программе, реализующего временную задержку, которые обеспечивают мигание светодиода с частотой около 1 Гц.

Задание 4. Добавить в программу оператор, опрашивающий состояние вывода микроконтроллера, к которому подключена кнопка. Реализовать программно два режима работы светодиода (мигание или непрерывное свечение) в зависимости от нажатия кнопки.

Задание 5. Перейти в режим отладки управляющей программы, выбрав для исполнения пункт меню **«Отладка>Запуск/Перезапуск отладки»**. При повторном нажатии пункта **«Отладка»** в окне появится сгенерированный отладчиком список, из которого следует выбрать исходный код программы (AVR>Source Code). Затем в пошаговом режиме, используя кнопку **«Шаг»**, инициировать выполнение программы, контролируя ее алгоритм работы.

Текст основной программы:

```
#include <iom128v.h>                    // подключение библиотечных функций
#include <macros.h>
```

```
void port_init(void)
{
 PORTA = 0x00;
 DDRA  = 0x00;
 PORTB = 0x00;
 DDRB  = 0b00000100;                  // вывод PB2 настроен на выход, PB1 – на вход
 ……                                   // остальные порты настроены аналогично порту A
}
void init_devices(void)               // данная функция сгенерирована Application Builder
{
 CLI();                               // disable all interrupts
 XDIV  = 0x00;                        // xtal divider
 XMCRA = 0x00;                        // external memory
 port_init();
 MCUCR = 0x00;
 EICRA = 0x00;                        // extended ext ints
 EICRB = 0x00;                        // extended ext ints
 EIMSK = 0x00;
 TIMSK = 0x00;                        // timer interrupt sources
 ETIMSK = 0x00;                       // extended timer interrupt sources
 SEI();                               // re-enable interrupts
}
   main()                             // начало основной программы - функция main
        {
        int i=0;
        char knopka=1;
        init_devices();
          while (1)                   // «бесконечный» цикл
          {
           PORTB |= (1<<PB2);         // вывод PB2 в  лог.1 - диод гаснет
                 for (i=0 ;i<5000 ;i++)  // временная задержка, реализованная
                 {;}                 // на операторе for
          knopka=PINB&0b00000010;    // опрос кнопки: 0 - нажата: 1 - не нажата
           if (knopka)               // проверка, нажата ли кнопка
             {
              PORTB &= ~(1<<PB2);    // вывод PB2 в  лог.0 - диод загорается
                   for (i=0 ;i<5000 ;i++)  // временная задержка на операторе for
                     {;}
        }  } }                       // конец функции main и операторов while и if
```

4.3. Изучение взаимодействия микроконтроллера с ЖК-индикатором

Задание 1. В среде Proteus спроектировать схему (рис. 4.3), содержащую микроконтроллер ATmega128 и буквенно-цифровой жидкокристаллический индикатор LM044L (4 строки по 20 символов в строке). Вывод питания индикатора V_{DD} и вывод управления контрастностью V_{EE} подключить к +5В, вывод V_{SS} – к общей шине GND. Выводы RS, RW и E индикатора подключить соответственно к выводам AD5, AD6 и AD7 порта A. Выводы данных индикатора D0…D7 подключить к выводам A8…A15 порта C микроконтроллера.

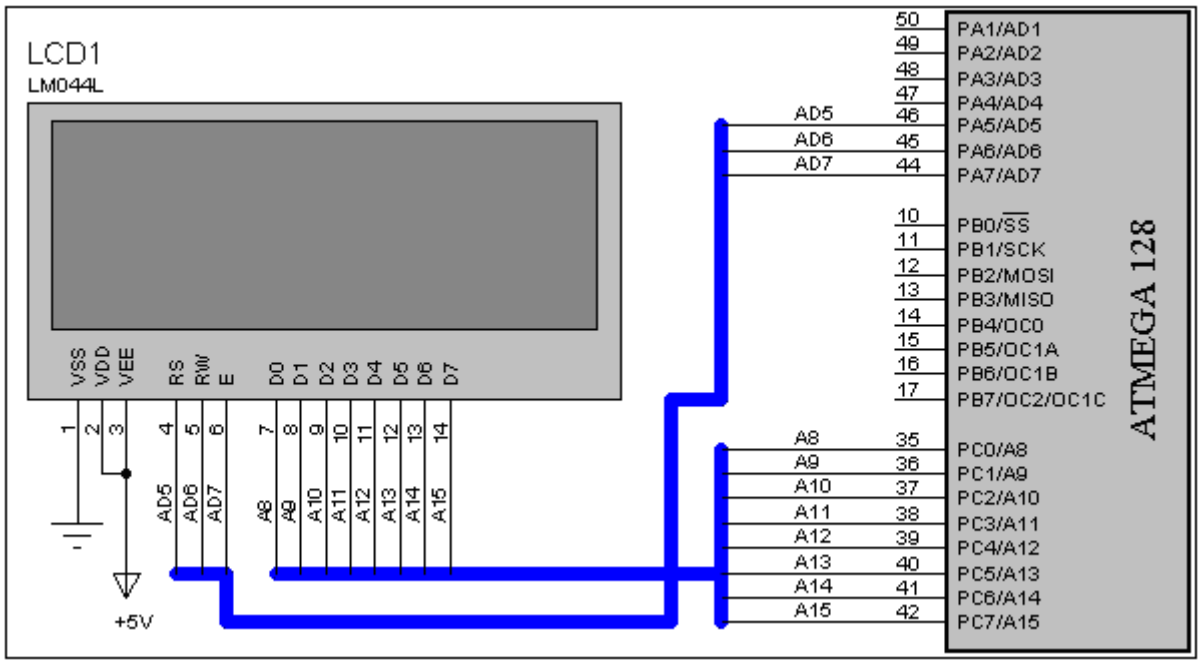

Рис. 4.3. Схема для изучения взаимодействия микроконтроллера с ЖК-индикатором

Задание 2. В среде ICCAVR разработать управляющую программу для микроконтроллера, с помощью которой на экран ЖК-индикатора выводится буквенно-цифровая информация. Для этого осуществить следующие действия:

- настроить все выводы микроконтроллера, к которым подключен индикатор, на выход;

- сформировать C- и H-файлы для управления временными задержками и ЖК-индикатором (тексты программ Delays и LCD находятся в Приложении), сохранив их в папке с разрабатываемым проектом. Добавить все эти файлы в проект и, используя директиву **#include,** подключить к основной программе проекта, содержащую функцию main. Файл LCD.c обеспечивает взаимодействие микроконтроллера с индикатором: установку курсора в нужную позицию дисплея, включение/выключение дисплея и курсора, очистку дисплея, вывод в заданную строку последовательности символов или в заранее установленную позицию отдельного символа и т.д. Файл Delays.c обеспечивает временные задержки, необходимые для решения различных задач, в том числе и для работы ЖК-индикатора. Он содержит две функции: timer0_delay() и timer0_N_sec(). Первая позволяет сформировать временные задержки в диапазоне от нескольких десятков микросекунд до нескольких секунд, что определяется параметрами, передаваемыми в функцию. Вторая функция позволяет реализовать временные задержки до нескольких сотен секунд;

- разработать управляющую программу для микроконтроллера, с помощью которой на индикатор выводится буквенно-цифровая информация, например, фамилия, год рождения и т.д.

Задание 3. В Proteus «прошить» в память микроконтроллера hex-файл (или cof-файл) управляющей программы и инициировать ее работу.

Текст основной программы:

```
#include <iom128v.h>
#include <macros.h>
#include "LCD.h"
#include "Delays.h"
void port_init(void)
{
 PORTA = 0x00;
 DDRA  = 0xFF;
 PORTB = 0x00;
 DDRB  = 0x00;
 PORTC = 0x00;
 DDRC  = 0xFF;
…….                              // порты D, E, F и G настраиваются аналогично порту B
}
void init_devices(void)          // данная функция сгенерирована Application Builder
{
 CLI();
 XDIV  = 0x00;
 XMCRA = 0x00;
 port_init();
 MCUCR = 0x00;
 EICRA = 0x00;
 EICRB = 0x00;
 EIMSK = 0x00;
 TIMSK = 0x00;
 ETIMSK = 0x00;
 SEI();
}
main ()                          // начало основной программы - функция main
{
char Family[]={'S','i', 'd', 'o', 'r', 'o', 'v', ' ', 'V', '.', ' ', 'I', '.' };          //Family="Sidorov V. I."
 init_devices();                  // инициализация МК
 Init_LCD_8bits();                // инициализация ЖК-индикатора
 LCD_Cursor_Off ();               // выключение курсора
 LCD_Cursor(2, 10);               // установка курсора в строку 2, колонку 10
 LCD_Cursor_On();                 // включение курсора
 LCD_DisplayCharacter('A');       // вывод на ЖК-индикатор символа "A"
 timer0_delay(128, 5);            // временная задержка (примерно на полсекунды)
 LCD_Cursor_Off();

while (1)                        // «бесконечный» цикл
   {
 LCD_Cursor(2, 10);//
 LCD_Cursor_On();
 timer0_delay(128, 5);
 LCD_DisplayCharacter('A');
 timer0_delay(128, 5);
 LCD_Cursor_Off();
```

```
LCD_Display_String (2, 1, Family, sizeof (Family));        // вывод на ЖК строки символов
timer0_delay(128, 5);
LCD_Clear();                                               // очистка дисплея
LCD_Display_String (3, 1, Family, sizeof (Family));
timer0_delay(128, 5);
LCD_Clear();
}  }
```

4.4. Изучение работы последовательного интерфейса USART на примере взаимодействия с внешним терминалом

Задание 1. В Proteus спроектировать схему, содержащую микроконтроллер ATmega128 и виртуальный терминал VIRTUAL TERMINAL (рис. 4.4). Терминал подключить к выводам USART0 микроконтроллера. Обратить внимание, что с точки зрения терминала микроконтроллер для него является передатчиком, поэтому вывод RXD терминала подключается к выводу PE1/TXD0 микроконтроллера, а вывод TXD терминала – к выводу PE0/RXD0.

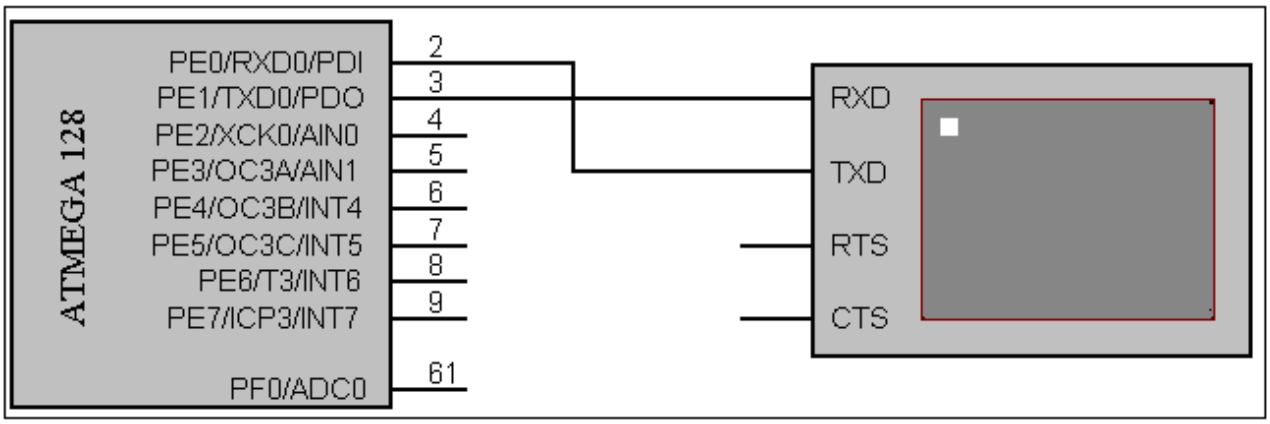

Рис. 4.4. Схема для изучения работы последовательного интерфейса USART

Задание 2. В среде ICCAVR разработать управляющую программу, с помощью которой микроконтроллер будет пересылать данные в компьютер, в качестве которого в Proteus используется виртуальный терминал. Пересылка данных осуществляется с помощью последовательного интерфейса USART. Для этого осуществить следующие действия:

- в **Applacation Builder** установить тип микроконтроллера, частоту синхронизации 11,059 МГц, инициализировать USART0, разрешив работу передатчика и установив формат посылки (скорость – 57600 бод, 8 бит передаваемых данных, 1 стоп-бит, без бита паритета). Сгенерировать функции uart0_init() и init_devices(), осуществляющие инициализацию USART0 и микроконтроллера в целом;

- разработать основную функцию main (), в которой производится вычисление квадратов и кубов целых чисел (от 1 до 16), а также вывод результатов вычислений на виртуальный терминал. Вывод данных осуществляется с помо-

щью библиотечной функции printf, для использования которой в программу необходимо включить директиву #include <stdio.h>;

Задание 3. В Proteus произвести настройку микроконтроллера: частота синхронизации – 11.059 MHz, значения бит SKSEL Fuses – (1111). «Прошить» в память микроконтроллера hex-файл (или cof-файл) управляющей программы. Настроить виртуальный терминал: скорость – 57600 бод, количество бит данных в посылке – 8, бит паритета – NONE, стоп-бит – 1. Инициировать старт программы.

Задание 4. Варьируя спецификаторы функции printf, произвести вывод на терминал буквенно-цифровой информации (слов, строк, чисел и т.д.).

Текст основной программы:

```
#include <iom128v.h>
#include <macros.h>
#include <stdio.h>
void port_init(void)
{
 PORTA = 0x00;
 DDRA  = 0x00;
……..                          // остальные порты настроены аналогично порту A
}
void uart0_init(void)          // данная функция сгенерирована Application Builder
{
 UCSR0B = 0x00;                // disable while setting baud rate
 UCSR0A = 0x00;
 UCSR0C = 0x06;
 UBRR0L = 0x0B;                // set baud rate lo
 UBRR0H = 0x00;                // set baud rate hi
 UCSR0B = (1<<TXEN0);          // Transmit enable
}
void init_devices(void)        // данная функция сгенерирована Application Builder
{
 CLI();                        // disable all interrupts
 XDIV  = 0x00;                 // xtal divider
 XMCRA = 0x00;                 // external memory
 port_init();
 uart0_init();
 MCUCR = 0x00;
 EICRA = 0x00;                 // extended ext ints
 EICRB = 0x00;                 // extended ext ints
 EIMSK = 0x00;
 TIMSK = 0x00;                 // timer interrupt sources
 ETIMSK = 0x00;                // extended timer interrupt sources
 SEI();                        // re-enable interrupts
}
main ()                        // функция main
{
int k, k2, k3, k_max=16;       // объявление и инициализация переменных
int i;
```

```
init_devices();                          // инициализация МК
  for (k=1; k<k_max; k++)
    {
          k2=k*k;                                // вычисление квадратов целых чисел
          k3=k*k*k;                              // вычисление кубов целых чисел
          printf("%d  %d  %d\r", k, k2, k3);     // вывод на терминал результатов вычисления
            for (i=1; i<60000; i++)              // временная задержка
            {;}
}         }
```

4.5. Изучение работы внутреннего аналого-цифрового преобразователя

Задание 1. В среде Proteus спроектировать схему, содержащую микроконтроллер ATmega128 и источник напряжения, например, генератор сигналов SIGNAL GENERATOR из группы виртуальных инструментов (рис. 4.5). Подключить источник к выводу PF1/ADC1 (к 1-му каналу мультиплексора внутреннего АЦП).

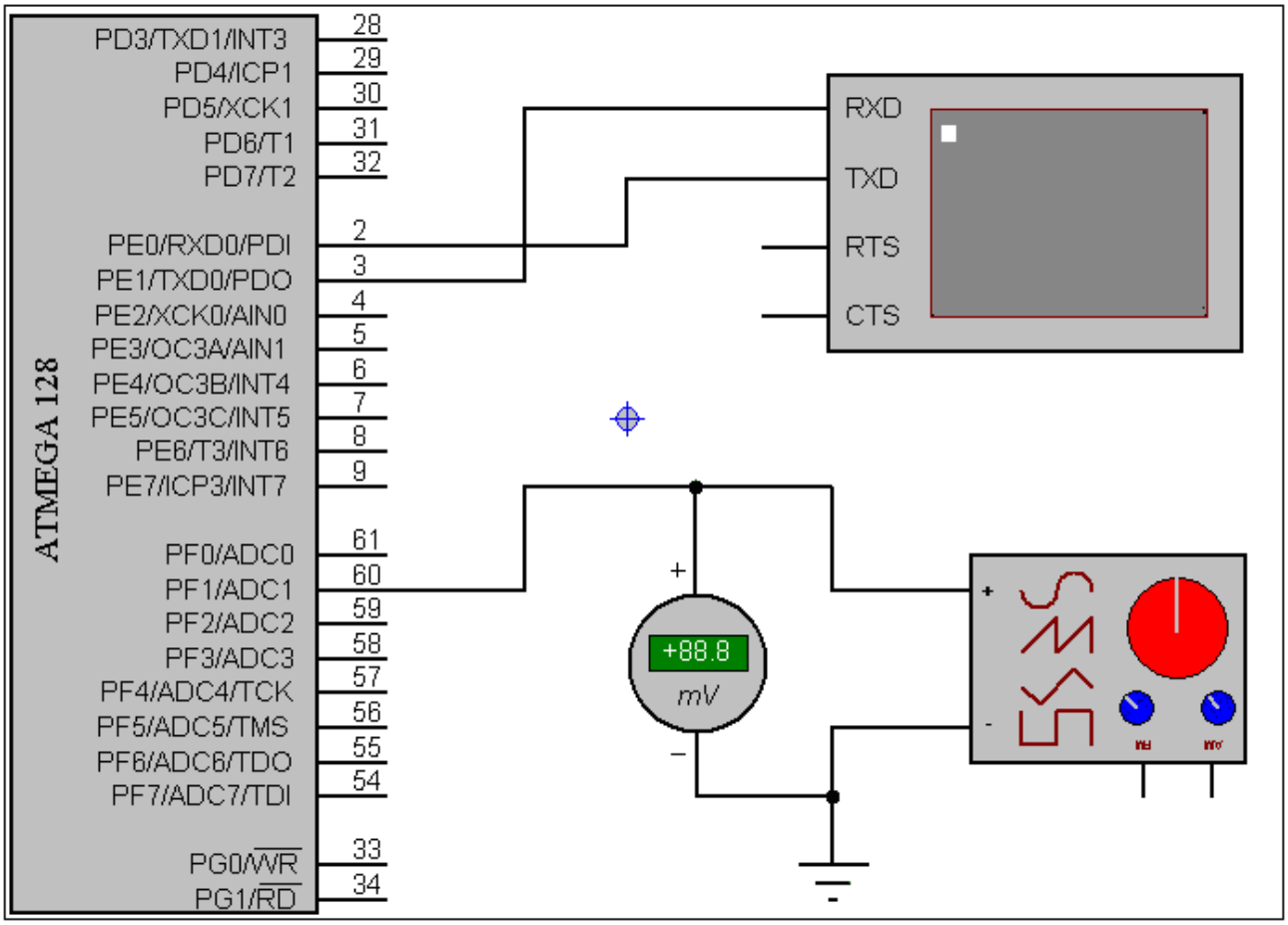

Рис. 4.5. Схема для изучения работы внутреннего АЦП

Задание 2. В среде ICCAVR разработать управляющую программу для микроконтроллера, с помощью которой внутренний АЦП будет производить

аналого-цифровое преобразование напряжения, поступающего от внешнего источника (генератора сигналов). Для этого осуществить следующие действия:

- с помощью **Applacation Builder** настроить выводы порта F на вход. Сгенерировать функцию инициализации АЦП, разрешающую работу АЦП в режиме одиночного преобразования. В качестве источника опорного напряжения установить внутренний ИОН на 2,56 В;

- разработать функцию, осуществляющую преобразование аналогового напряжения в цифровой код. Учесть, что старт преобразованию дает установка бита ADSC (регистр **ADCSRA**) в лог.1, который после завершения преобразования сбрасывается в лог.0;

- разработать основную функцию main (), в которой непрерывно производится преобразование аналогового напряжения в цифровой код. Результаты преобразования записывать в регистр данных какого-либо порта, например, порта С, предварительно настроив все его выводы на выход. Таким способом результат преобразования можно визуально индицировать в виде двоичного кода с помощью красных или синих квадратиков у выводов данного порта (красный – лог.1, синий – лог.0). Учесть, что функция main () должна содержать функцию инициализации устройства init_devices(), которую генерирует **Applacation Builder**. Не забыть установить нужный номер канала мультиплексора, к которому подключен источник внешнего сигнала.

Задание 3. В Proteus «прошить» в память микроконтроллера hex-файл (или cof-файл) управляющей программы и инициировать ее работу. Установить в генераторе сигналов в качестве выходного сигнала синусоидальный сигнал с частотой, равной нулю (постоянное напряжение).

Задание 4. Для контроля результата преобразования подключить к источнику сигнала виртуальный вольтметр. Убедиться, что результаты преобразования АЦП (двоичный код на выводах порта С) согласуются с показаниями вольтметра. Учесть, что с помощью 8-битного регистра данных порта можно визуально индицировать только младший байт результата, поэтому желательно на вход АЦП подавать небольшое по величине напряжение (менее 1000 мВ).

Задание 5. Подключить к выводам USART0 микроконтроллера виртуальный терминал. Добавить к управляющей программе функцию инициализации uart0_init() из программы, приведенной в п. 4.4 (скопировав ее в буфер и вставив затем в программу). Включить ее имя в функцию init_devices(). Доработать функцию main(), дополнив ее операторами, обеспечивающими вывод на терминал результатов преобразования АЦП (в милливольтах). Учесть, что единица младшего разряда АЦП при установленном внутреннем источнике опорного напряжения 2,56 В равна 25 мВ. Сравнить показания вольтметра с результатами, выводимыми на терминал.

Текст основной программы:

```
#include <iom128v.h>
#include <macros.h>
void port_init(void)
```

```
{
 PORTA = 0x00;
 DDRA  = 0x00;
……..                                  // порты B, F и G настраиваются аналогично порту A
 PORTC = 0x00;
 DDRC  = 0xFF;
 PORTD = 0x00;
 DDRD  = 0xFF;
 PORTE = 0x00;
 DDRE  = 0xFF;
 }
void uart0_init(void)
{
 UCSR0B = 0x00;
 UCSR0A = 0x00;
 UCSR0C = 0x06;                       // установка формата посылки (8 байт)
 UBRR0L = 0x0B;                       //установка скорости передачи 57600 бод (мл. байт)
 UBRR0H = 0x00;                       // установка скорости передачи 57600 бод (ст. байт)
 UCSR0B = (1<<TXEN0);                 // разрешение работы передатчика
}
void adc_init(void)                   // функцию генерирует Aplacation Builder
{
 ADCSRA = 0x00;                       // разрешение работы АЦП
 ADMUX = 0b11000000;                  // подключение внутреннего ИОН на 2,56 В
 ACSR  = 0x80;                        // аналоговый компаратор отключен
 ADCSRA = 0b11000110;                 // установка ADEN, ADSC в лог.1 (разрешение и
 }                                    // старт), а также ADPS2 и ADPS1 (делитель на 64)
int ReadADC(void)                     // функция чтения внутреннего АЦП
{    int data_ADC;
   ADCSRA |= (1<<ADSC);               // старт преобразованию установкой бита ADSC
        while (ADCSRA & (1<<ADSC));   // ожидание, пока закончится преобразование и
                                      // бит ADSC в регистре ADCSRA обнулится
   data_ADC = ADCL;                   // чтение младшего байта
        data_ADC += (int)ADCH << 8;   // добавление к результату старшего байта
        return data_ADC;              // передача результата на выход функции
}
void init_devices(void)               // функцию генерирует Aplacation Builder
{
 CLI();                               // disable all interrupts
 XDIV  = 0x00;                        // xtal divider
 XMCRA = 0x00;                        // external memory
 port_init();
 uart0_init();
 adc_init();
 MCUCR = 0x00;
 EICRA = 0x00;                        // extended ext ints
 EICRB = 0x00;                        // extended ext ints
 EIMSK = 0x00;
 TIMSK = 0x00;                        // timer interrupt sources
 ETIMSK = 0x00;                       // extended timer interrupt sources
```

```
 SEI();                                  // re-enable interrupts
}
main ()                                  // функция main
{
 int U,i,j;
 init_devices();
 ADMUX |= (1<<MUX0);                     // выбор канала измерения (канал №1).
 while (1)
        {
         U=ReadADC();                    // результат преобразования в единицах АЦП
         PORTC=U;                        // вывод данных в регистр порта С
         U=2.5*U;                        // результат преобразования в мВ
         printf("%d\r",U);               // вывод результатов на терминал
         for (i=1; i<60000; i++)         // временная задержка (около 1 с)
           {
           for (j=1; j<10; j++);
}        }  }
```

4.6. Изучение работы интерфейса SPI на примере взаимодействия с внешним АЦП

Задание 1. В Proteus спроектировать схему, содержащую микроконтроллер ATmega128 и 10-разрядный аналого-цифровой преобразователь MCP3001 с последовательным SPI-выходом (рис. 4.6). Подключить вывод CLK у MCP3001 к выводу PB1/SCK микроконтроллера, вывод DO – к PB3/MISO, вывод CS – к PE7 (вывод PE7 на рисунке не показан). К выводу VREF подключить источник напряжения +2.56 V (нажать на кнопку «Терминал» на левой панели инструментов и выбрать пункт POWER). Напряжение на вход АЦП подать с интерактивного потенциометра POT-LIN, который подключатся к источнику напряжения +1 V. Результаты аналого-цифрового преобразования выводить на виртуальный терминал. Для этого вывод RXD0 терминала подключить к выводу PE1/TXD0 микроконтроллера, а вывод TXD0 терминала – к выводу PE0/RXD0 микроконтроллера (выводы PE0 и PE1 на рисунке не показаны).

Задание 2. В среде ICCAVR разработать управляющую программу, с помощью которой микроконтроллер посредством интерфейса SPI будет получать данные в последовательном коде с выхода внешнего АЦП и передавать их на терминал. Для этого осуществить следующие действия:

- с помощью **Applacation Builder** настроить все используемые выводы микроконтроллера в зависимости от их назначения либо на вход, либо на выход. Сгенерировать функцию инициализации spi_init(), разрешающую работу SPI с передачей данных старшими разрядами вперед с тактовой частотой $f_0/4$. Сгенерировать функцию инициализации приемопередатчика uart0_init() и функцию инициализации микроконтроллера init_devices();

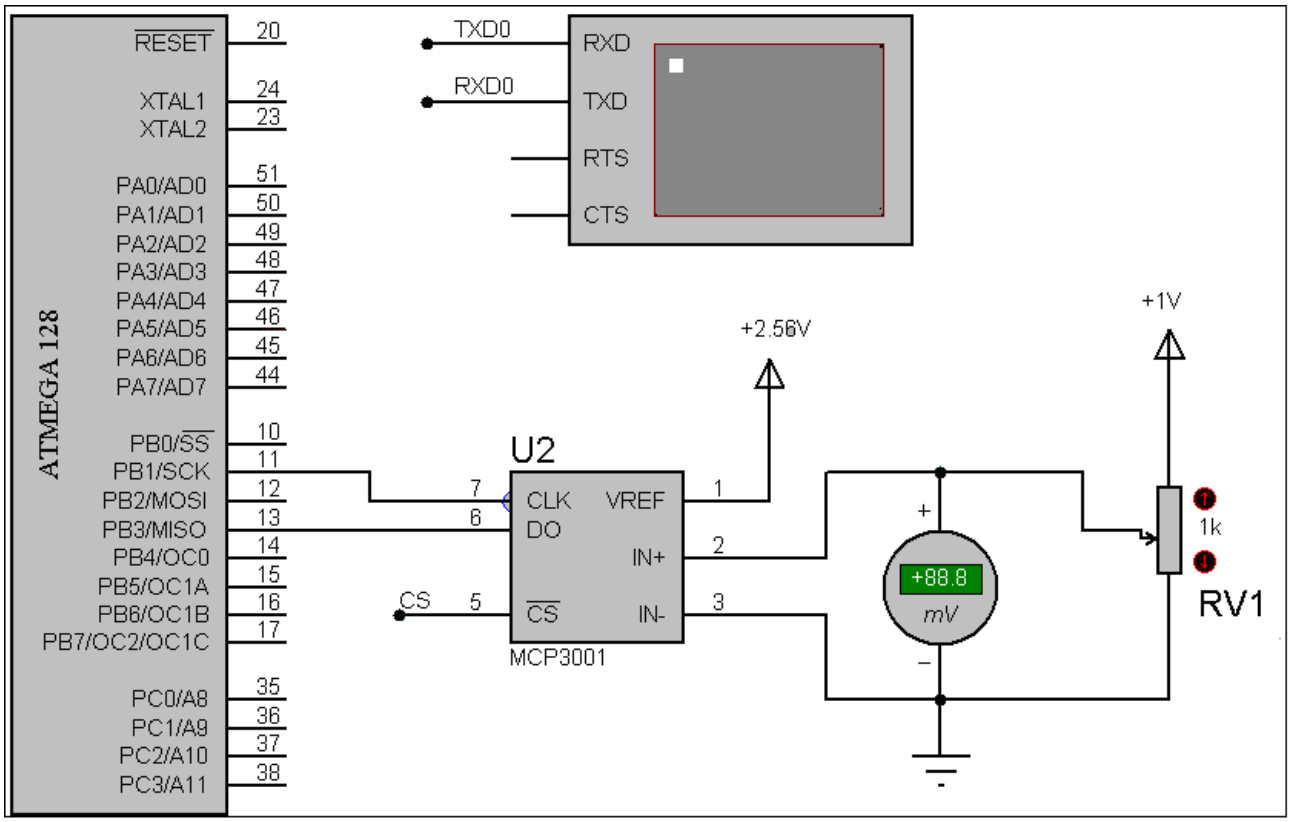

Рис. 4.6. Схема для изучения работы SPI-интерфейса

- сформировать C- и H-файлы для управления внешним АЦП и SPI-интерфейсом (тексты программ adc_ext и spi находятся в Приложении), сохранив их в папке с разрабатываемым проектом. Добавить все эти файлы в проект и, используя директиву **#include**, подключить к программе, содержащую функцию main();

- используя функцию ReadADC_ext() из файла adc_ext.c, разработать программу, осуществляющую преобразование аналогового напряжения в цифровой код и передачу его на виртуальный терминал, предварительно рассчитав значение единицы младшего разряда и преобразовав код в напряжение (в милливольтах). Для лучшего восприятия результатов моделирования производить пересылку данных на терминал с временной задержкой, реализованной программно, например, с помощью оператора **for**.

Задание 3. В Proteus установить режимы работы терминала (скорость, количество бит в посылке и т.д.) и установить для микроконтроллера нужные биты CKSEL Fuses и частоту синхронизации. «Прошить» управляющую программу (hex-файл или cof-файл) в память микроконтроллера и инициировать ее работу.

Задание 4. Изменяя положение движка потенциометра варьировать поступающее на вход АЦП напряжение и сравнивать результаты преобразования на терминале с результатами измерений на виртуальном вольтметре.

Текст основной программы:

```
#include <iom128v.h> #include <macros.h>
#include "adc_ext.h"
#include "spi.h"
void port_init(void)
{
 PORTA = 0x00;
 DDRA  = 0x00;
 PORTB = 0x00;
 DDRB  = 0b00000010;
 PORTC = 0x00;
 DDRC  = 0xFF;
 PORTD = 0x00;
 DDRD  = 0x00;
 PORTE = 0x00;
 DDRE  = 0xFE;
 PORTF = 0x00;
 DDRF  = 0xFF;
 PORTG = 0x00;
 DDRG  = 0x00;
}
void spi_init(void)
{
 SPCR = 0b01010000;          // разрешение SPI, передача ст. разрядами вперед, частота f0/4
 SPSR = 0x00;
}
void uart0_init(void)
{
 UCSR0B = 0x00;
 UCSR0A = 0x00;
 UCSR0C = 0x06;              / /установка формата посылки (8 байт)
 UBRR0L = 0x0B;              // установка скорости передачи 57600 бод (мл. байт)
 UBRR0H = 0x00;              // установка скорости передачи 57600 бод (ст. байт)
 UCSR0B = (1<<TXEN0);        // разрешение работы передатчика
}
void init_devices(void)      // функцию генерирует Aplacation Builder
{
 CLI();                      // disable all interrupts
 XDIV  = 0x00;               // xtal divider
 XMCRA = 0x00;               // external memory
 port_init();
 uart0_init();
 spi_init();
 MCUCR = 0x00;
 EICRA = 0x00;               // extended ext ints
 EICRB = 0x00;               // extended ext ints
 EIMSK = 0x00;
 TIMSK = 0x00;               // timer interrupt sources
 ETIMSK = 0x00;              // extended timer interrupt sources
```

```
 SEI();                          // re-enable interrupts
}
main ()                          // функция main
{
int U=0,i,j;
init_devices();
while (1)
  {
   U=ReadADC_ext();
   U=2.5*U;                      // для 10-битного АЦП при Uоп=2,56 В цена деления = 2,5  мВ
   printf("%d\r",U);
   for (i=1; i<60000; i++)       // временная задержка около 1 с
          {
          for (j=1; j<10; j++);
} }       }
```

4.7. Изучение работы 16-разрядного таймера/счетчика на примере генерации ШИМ-импульсов

Задание 1. В среде Proteus спроектировать схему, содержащую микроконтроллер ATmega128 и виртуальный осциллограф (рис. 4.7). Подключить один из входов осциллографа к выводу PB5/OC1A микроконтроллера. Данный вывод используется таймером/счетчиком T/C1 для генерации широтно-импульсно модулированных импульсов (ШИМ-импульсов) в режимах работы, в которых задействован регистр сравнения OC1A.

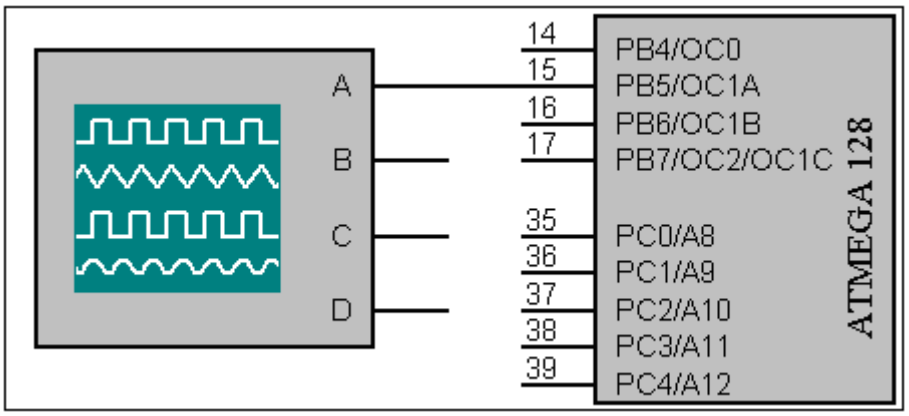

Рис. 4.7. Схема для изучения работы 16-разрядного таймера/счетчика

Задание 2. В среде ICCAVR разработать управляющую программу, с помощью которой микроконтроллер формирует на выводе PB5/OC1A ШИМ-импульсы с постоянным периодом следования и изменяющейся длительностью. Для этого осуществить следующие действия:

- с помощью **Applacation Builder** настроить частоту синхронизации на 11,059 МГц, вывод порта PB5 – на выход. Произвести настройки T/C1 на режим генерации ШИМ-импульсов (читатель, разумеется, может выбрать свои настройки): желаемое значение характеристик сигнала – 1 мс; предделитель час-

тоты – 1; режим вывода OC1A – сброс в состояние лог.0; разрешить прерывание при совпадении TCNT1 с OCR1A; выбрать один из режимов работы, например, номер 14 – режим быстрой ШИМ с вершиной счета, установленной содержимым регистра ICR1;

- сгенерировать функцию инициализации timer1_init() и шаблоны функций обработки прерываний по совпадению содержимого TCNT1 с регистром сравнения OCR1A (#pragma interrupt_handler timer1_compa_isr:13) и TCNT1 с ICR1 (#pragma interrupt_handler timer1_capt_isr:12). При необходимости можно скорректировать значения всех регистров, управляющих работой T/C1, «в ручную», т.е. не используя **Applacation Builder**;

- в функции обработки прерываний по совпадению с регистром сравнения производить модификацию содержимого OCR1A, например, инкрементируя его значение;

- разработать основную функцию main (), в которой с помощью оператора **while** реализована «бесконечная» петля. Предварительно произвести инициализацию устройства, используя функцию init_devices().

Задание 3. В среде Proteus «прошить» управляющую программу в память микроконтроллера, предварительно настроив его частоту синхронизации, и инициировать ее работу.

Задание 4. Произвести изменения значений регистров OCR1A и ICR1, наблюдая, как скажутся эти изменения на работе ШИМ-генератора.

Текст основной программы:

```
#include <iom128v.h>
#include <macros.h>
void port_init(void)
{
 PORTA = 0x00;
 DDRA  = 0x00;
 PORTB = 0x00;
 DDRB  = 0xFF;                  // выводы порта B настроены на выход
 PORTC = 0x00;
 DDRC  = 0xFF;                  // выводы порта C настроены на выход
……                              // порты D, E, F и G настраиваются аналогично порту A
}
void timer1_init(void)
{
 TCCR1B = 0x00;
 TCNT1H = 0x00;
 TCNT1L = 0x00;
 OCR1BH = 0x04;
 OCR1BL = 0x50;
 OCR1CH = 0x04;
 OCR1CL = 0x50;
 TCCR1A = (1<<COM1A1)|(1<<WGM11); //установка режима быстрой ШИМ
 TCCR1B = (1<<WGM13)|(1<<WGM12);
 ICR1H  = 0x2B;                         //формирование периода следования 1 мс
```

```
 ICR1L  = 0x33;
 OCR1A = 100;                                  //формирование импульса длительностью 70 мкс
}        //---------------------------------------------------------------------------
// Функция обработки прерывания в момент фронта импульса (по регистру захвата ICR1)
#pragma interrupt_handler timer1_capt_isr:12
void timer1_capt_isr(void)
{;}      //---------------------------------------------------------------------------
//Функция обработки прерывания в момент среза импульса (по регистру сравнения OCR1A)
#pragma interrupt_handler timer1_compa_isr:13
void timer1_compa_isr(void)
{
 OCR1A++;                    // увеличение длительности импульсов (примерно на 0,1 мкс)
 if (OCR1A>ICR1)             // проверка, не превысит ли длительность период следования
         {
         OCR1A=100;          // если превысит, то формирование ШИМ начнется с начала
         }
}        //---------------------------------------------------------------------------
void init_devices(void)
{
 CLI();                                        // disable all interrupts
 XDIV  = 0x00;                                 // xtal divider
 XMCRA = 0x00;                                 // external memory
 port_init();
 MCUCR = 0x00;
 EICRA = 0x00;                                 // extended ext ints
 EICRB = 0x00;                                 // extended ext ints
 EIMSK = 0x00;
 TIMSK = 0x30;                                 // timer interrupt sources
 ETIMSK = 0x00;                                // extended timer interrupt sources
 SEI();                                        // re-enable interrupts
 timer1_init();
}
main ()                                        // функция main
{
 init_devices();
 TCCR1B |= (1<<CS10);                          // старт таймера и запуск ШИМ
 while (1)                                     // бесконечная петля
    {;}
}
```

4.8. Изучение работы цифрового потенциометра

Задание 1. В Proteus спроектировать схему, содержащую микроконтроллер ATmega128 и цифровой потенциометр AD5290YRMZ10 с последовательным SPI-выходом (рис. 4.8). В цифровых потенциометрах с помощью микроконтроллера можно устанавливать нужные значения сопротивления. Диапазон и дискретность устанавливаемых в потенциометре сопротивлений зависят от марки потенциометра. Для AD5290YRMZ10 максимальное сопротивление равно 10 К; количество дискретных значений – 256; максимальное напряжение,

прикладываемое к потенциометру, может принимать значения в диапазоне от 0 до 30 В или от -15 В до +15 В.

Подключить вывод CLK у AD5290YRMZ10 к выводу PB1/SCK микроконтроллера, вывод SDI – к PB2/MOSI, вывод CS – к PE6 (вывод PE6 на рисунке не показан). Подключить вывод A потенциометра к источнику питания +15 V, а вывод B вместе с VSS – к общей шине GND. Снимаемое с выхода W потенциометра напряжение измерять с помощью виртуального вольтметра и контролировать по осциллографу. Дважды нажать левой кнопкой мыши на графическом изображении потенциометра, вызвав диалоговое окно «Правка компонента», выбрать в нем пункт «Скрытые пины» и в диалоговом окне «Скрытые пины питания» установить соединение пина GND с VSS, а пина VDD – с +15V.

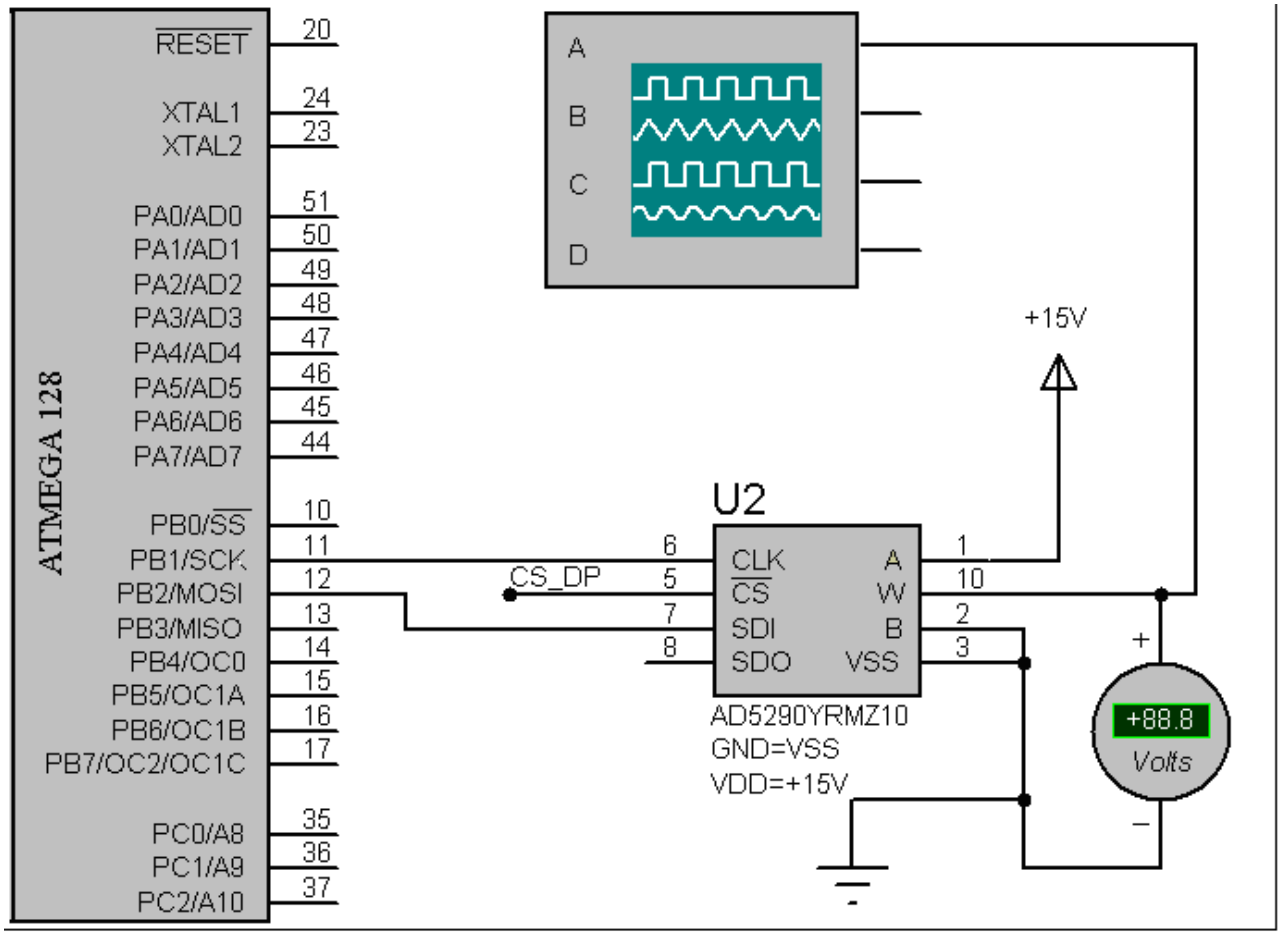

Рис. 4.8. Схема для изучения работы цифрового потенциометра

Задание 2. В среде ICCAVR разработать управляющую программу, с помощью которой микроконтроллер посредством интерфейса SPI будет передавать данные в последовательном коде на цифровой вход потенциометра, устанавливая тем самым различные сопротивления и изменяя напряжение, снимаемое с выхода потенциометра. В управляющей программе сформировать линей-

ный закон изменения сопротивления, позволяющий получить на выходе потенциометра пилообразное напряжение. Для этого осуществить следующие действия:

- с помощью **Applacation Builder** настроить все используемые выводы микроконтроллера в зависимости от их назначения либо на вход, либо на выход. Сгенерировать функцию инициализации spi_init(), разрешающую работу SPI с передачей данных старшими разрядами вперед с тактовой частотой $f_0/4$. Сгенерировать функцию инициализации микроконтроллера init_devices();

- разработать функцию для управления передачей байта данных от микроконтроллера к цифровому потенциометру (можно воспользоваться примерами, приведенными в технической документации на ATmega128). Скопировать в папку с разрабатываемым проектом C- и H-файлы программы Delays для формирования временных задержек (программа Delays используется в примере 3 и ее текст имеется в Приложении). Добавить эти файлы в проект и, используя директиву **#include,** подключить к программе, содержащую функцию main();

- используя функцию для управления передачей байта данных, разработать программу, осуществляющую циклическую передачу из микроконтроллера в потенциометр переменной, имеющей тип **char**. В цикле после каждой пересылки переменной ее значение должно увеличиваться на единицу. Между пересылками установить временную задержку около 2 миллисекунд.

Задание 3. В среде Proteus «прошить» управляющую программу в память микроконтроллера и инициировать ее работу. Наблюдать на виртуальном осциллографе форму выходного сигнала.

Текст основной программы:

```
#include <iom128v.h>
#include <macros.h>
#include "Delays.h"
void port_init(void)
{
 PORTA = 0x00;
 DDRA  = 0x00;
 PORTB = 0x00;
 DDRB  = 0b00000110;        // выводы порта PB1 и PB2 настроены на выход
 PORTC = 0x00;
 DDRC  = 0xFF;
…….                         // порты D, E, F и G настраиваются аналогично порту C
}
void spi_init(void)
{
 SPCR = 0b01010000;         //разрешение SPI, передача ст. разрядами вперед, частота f0/4
 SPSR = 0x00;               // удвоения скорости нет
}
void init_devices(void)
{
 CLI();                                 //disable all interrupts
 XDIV  = 0x00;                          //xtal divider
 XMCRA = 0x00;                          //external memory
```

```
 port_init();
 spi_init();
 MCUCR = 0x00;
 EICRA = 0x00;                          //extended ext ints
 EICRB = 0x00;                          //extended ext ints
 EIMSK = 0x00;
 TIMSK = 0x00;                          //timer interrupt sources
 ETIMSK = 0x00;                         //extended timer interrupt sources
 SEI();                                 //re-enable interrupts
}
void SPI_MasterTransmit(char cData)     // функция управления передачей байта данных
{
PORTE&=~(1<<PE6);                       //выбор цифрового потенциометра
asm("nop");                             //синхронизация перед чтением выводов порта B
SPDR = cData;                           //запуск передачи данных
while(!(SPSR & (1<<SPIF)));             //ожидание завершения передачи данных
PORTE|=(1<<PE6);                        //отключение выбора цифрового потенциометра
}
main ()
{
char a=0;
init_devices();
        while (1)
        {
                for (a; a<=255; a++)
                {
                SPI_MasterTransmit(a);
                timer0_delay(191, 1);           // задержка около 2 мс
        }       }
```

ПРИЛОЖЕНИЕ.
Тексты программ для формирования временных задержек и управления ЖК-индикатором, SPI-интерфейсом и внешним АЦП

Файл Delays.c

```
#include <iom128v.h>
#include <macros.h>
                                        /*      Типичные значения временных задержек:
                                        100 uSec (N_code=255-2=253, N_Clock_Select=1)
                                        1 mSec (N_code=255-32=223, N_Clock_Select=1)
                                        2 mSec (N_code=255-64=191, N_Clock_Select=1)
                                        5 mSec (N_code=255-162=93, N_Clock_Select=1)
                                        10 mSec (N_code=255-40=215, N_Clock_Select=2)
                                        50 mSec (N_code=255-200=55, N_Clock_Select=2)
                                        100 mSec (N_code=255-100=155, N_Clock_Select=3)
                                        500 mSec (N_code=255-127=128, N_Clock_Select=5)
                                        650 mSec (N_code=255-166=89, N_Clock_Select=5)
                                        750 mSec (N_code=255-191=64, N_Clock_Select=5)
                                        1  Sec  (N_code=255-31=224, N_Clock_Select=7)          */
void timer0_delay(unsigned char N_code, unsigned char N_Clock_Select)
{
 unsigned char temp = TIMSK;
 TCCR0 = 0x00;
 TIMSK = 0x00;
 ASSR  = (1<<AS0);
 while ((ASSR & (1<<TCN0UB)));
 TCNT0 = N_code;
 while ((ASSR & (1<<TCN0UB)));
 TIFR |= (1<<TOV0);
 if ((N_Clock_Select>0)&&(N_Clock_Select)<8)
   {
    TCCR0 = (N_Clock_Select<<CS00);
   }
 else
   {
    TCCR0 = 0x00;
   }
 while (!(TIFR & (1<<TOV0)));
 TIFR |= (1<<TOV0);
 TCCR0 = 0x00;
 TIMSK = temp;
}
void timer0_N_sec(unsigned char N_sec)
{
 while (N_sec)
          {
           timer0_delay(224,7);
           N_sec--;
          }
}
```

Файл Delays.h

```
void timer0_delay(unsigned char N_code, unsigned char N_Clock_Select);
void timer0_N_sec(unsigned char N_sec);
```

Файл LCD.c

```
#include <iom128v.h>
#include <macros.h>
#define BIT7 0x80
#define BIT6 0x40
#define BIT5 0x20
#define BIT4 0x10
#define BIT3 0x08
#define BIT2 0x04
#define BIT1 0x02
#define BIT0 0x01
#if  0
#define LCD_OP_PORT                     PORTC
#define LCD_IP_PORT                     PINC
#define LCD_DIR_PORT                    DDRC
#define LCD_EN_PORT                     PORTA
#define EN_BIT                          BIT7
#define LCD_RW_PORT                     PORTA
#define RW_BIT                          BIT6
#define LCD_RS_PORT                     PORTA
#define RS_BIT                          BIT5
#endif
volatile unsigned char * LCD_EN_PORT = &PORTA;
volatile unsigned char * LCD_DIR_PORT = &DDRC;
volatile unsigned char * LCD_IP_PORT = &PINC;
volatile unsigned char * LCD_OP_PORT = &PORTC;
volatile unsigned char * LCD_RS_PORT = &PORTA;
volatile unsigned char * LCD_RW_PORT = &PORTA;
char LCD_EN_BIT = BIT(7);
char LCD_RW_BIT = BIT(6);
char LCD_RS_BIT = BIT(5);
#define SET_LCD_E      *LCD_EN_PORT |= LCD_EN_BIT
#define CLEAR_LCD_E    *LCD_EN_PORT &= ~LCD_EN_BIT
#define SET_LCD_DATA   *LCD_RS_PORT |= LCD_RS_BIT
#define SET_LCD_CMD    *LCD_RS_PORT &= ~LCD_RS_BIT
#define SET_LCD_READ   *LCD_RW_PORT |= LCD_RW_BIT
#define SET_LCD_WRITE  *LCD_RW_PORT &= ~LCD_RW_BIT
#define LCD_ON                  0x0C
#define LCD_CURS_ON             0x0E
#define LCD_OFF                 0x08
#define LCD_HOME                0x02
#define LCD_CLEAR               0x01
#define LCD_NEW_LINE            0xC0
#define LCD_FUNCTION_SET        0x38
#define LCD_MODE_SET            0x06
unsigned char LCD_Busy ( void )
```

```
{
unsigned char temp;
CLEAR_LCD_E;                                    // Disable LCD
*LCD_DIR_PORT = 0x00;                           // Make I/O Port input
SET_LCD_READ;                                   // Set LCD to READ
SET_LCD_CMD;                                    // Set LCD to Command
SET_LCD_E;                                      // Enable LCD
asm("nop");                                     // Delay = 1 instruction = 91 ns, if
f=11.0592Mhz
temp = *LCD_IP_PORT;                            // Load data from port
CLEAR_LCD_E;                                    // Disable LCD
return temp;                                    // return busy flag
}
void LCD_WriteControl (unsigned char CMD)
{
        while (LCD_Busy()& 0X80);          // Test if LCD busy
        CLEAR_LCD_E;                            // Disable LCD
        SET_LCD_WRITE ;                         // Set LCD to write
        SET_LCD_CMD;                            // Set LCD to command
        *LCD_DIR_PORT = 0xFF;                   // LCD port output
        *LCD_OP_PORT = CMD;
        SET_LCD_E;                              // Enable LCD - Write data to LCD
        asm("nop");    CLEAR_LCD_E;             // Disable LCD
}
void LCD_WriteControl_init (unsigned char CMD)
{
        CLEAR_LCD_E;                             // Disable LCD
        SET_LCD_WRITE ;
        SET_LCD_CMD;                            // Set LCD to command
        *LCD_DIR_PORT = 0xFF;                   // LCD port output
        *LCD_OP_PORT = CMD;                     // Load data (Code of Command) to port
        SET_LCD_E;                              // Enable LCD - Write data to LCD
        asm("nop");                             // Delay = 1 instruction = 91 ns, if
f=11.0592Mhz
        asm("nop");
        CLEAR_LCD_E;                            // Disable LCD
}
void LCD_WriteData (unsigned char Data)
{       while (LCD_Busy() & 0X80);              // Test if LCD Busy
        CLEAR_LCD_E;                            // Disable LCD
        SET_LCD_WRITE ;                         // Set LCD to write
        SET_LCD_DATA;                           // Set LCD to data
        *LCD_DIR_PORT = 0xFF;                   // LCD port output
        *LCD_OP_PORT = Data;                    // Load data to port
        SET_LCD_E;                              // Write data to LCD
        asm("nop");                             // Delay=1 instruction = 91 ns, if
f=11.0592Mhz
        asm("nop");
        CLEAR_LCD_E;                            // Disable LCD
}
```

```
void Init_LCD_8bits(void)
{
      timer0_delay(175, 2);                        // waiting for 20 ms
      LCD_WriteControl_init (LCD_FUNCTION_SET);
      timer0_delay(93, 1);                         // waiting for 5 ms
      LCD_WriteControl_init (LCD_FUNCTION_SET);
      timer0_delay(253, 1);                        // waiting for 100 mks
      LCD_WriteControl_init (LCD_FUNCTION_SET);
      timer0_delay(254, 1);                        // waiting for 40 mks
      LCD_WriteControl_init (LCD_FUNCTION_SET);
      timer0_delay(254, 1);                        // waiting for 40 mks
      LCD_WriteControl (LCD_OFF);
      LCD_WriteControl (LCD_CLEAR);
      LCD_WriteControl (LCD_MODE_SET);
}
void LCD_Clear(void)
{ LCD_WriteControl(0x01); }
void LCD_Home(void)
{ LCD_WriteControl(0x02); }
void LCD_DisplayCharacter (unsigned char Char)
{ LCD_WriteData (Char); }
void LCD_Display_String (char row, char column ,void *ptr, char size)
 {
  char *string = ptr;
  LCD_Cursor (row, column);
  while (size--)
          {
          LCD_DisplayCharacter (*string);
           string++;
          }
 }
void LCD_DisplayString (char row, char column ,unsigned char *string)
{
          LCD_Cursor (row, column);
          while (*string)
                    LCD_DisplayCharacter (*string++);
}
void LCD_Cursor (char row, char column)
{
          switch (row) {
                    case 1: LCD_WriteControl (0x80 + column - 1); break;
                    case 2: LCD_WriteControl (0xc0 + column - 1); break;
                    case 3: LCD_WriteControl (0x94 + column - 1); break;
                    case 4: LCD_WriteControl (0xd4 + column - 1); break;
                    default: break; }
}
void LCD_Cursor_On (void)
{ LCD_WriteControl (LCD_CURS_ON); }
void LCD_Cursor_Off (void)
{ LCD_WriteControl (LCD_ON); }
```

```
void LCD_Display_Off (void)
{ LCD_WriteControl(LCD_OFF); }
void LCD_Display_On (void)
{ LCD_WriteControl(LCD_ON); }
```

Файл LCD.h

```
void Init_LCD_8bits(void);
void LCD_Display_Off(void);
void LCD_Display_On(void);
void LCD_Clear(void);
void LCD_Home(void);
void LCD_Cursor(char row, char column);
void LCD_Cursor_On(void);
void LCD_Cursor_Off(void);
void LCD_DisplayCharacter(unsigned char Char);
void LCD_DisplayString(char row, char column, unsigned char *string);
```

Файл spi.c

```
#include <iom128v.h>
#include "spi.h"
volatile char spiTransferComplete;
const TRUE=0xFF;
const FALSE=0x00;
void spiInit()
{
        int tmp;
        PORTB|=0x07;
        DDRB|=0x07;
        SPCR|=(1<<MSTR);
        SPCR&=~(1<<SPR0);
        SPCR&=~(1<<SPR1);
        SPCR&=~(1<<CPOL);
        SPCR&=~(1<<DORD);
        SPCR|=(1<<SPE);
        tmp=SPSR;
        spiTransferComplete = TRUE;
}
void spiSendByte(char data)
{
        while(!(SPSR & 128));
        spiTransferComplete = FALSE;
        SPDR=data;
}
char spiReciveByte(void)
{
int rx;
spiTransferComplete = FALSE;
SPDR=123;
while(!(SPSR & 128));
rx=SPDR;
```

```
return rx;
}
char spiTransferByte(char data)
{
  unsigned int tmp;
        spiTransferComplete = FALSE;
        SPDR=data;
                while(!(SPSR & 128));
                tmp=SPDR;
                spiTransferComplete = TRUE;
        return SPDR;
}
unsigned int spiTransferWord(int data)
{
        unsigned int rxData = 0;
        rxData = (spiTransferByte((data>>8) & 0x00FF))<<8;
        rxData |= (spiTransferByte(data & 0x00FF));
        return rxData;
}
unsigned int spiReciveWord(void)
{
 unsigned int rxData = 0;
 rxData=(spiReciveByte()<<8)|(spiReciveByte());
 return rxData;
}
```

Файл spi.h

```
#ifndef SPI_H
#define SPI_H
void spiInit(void);
void spiSendByte(char data);
char spiTransferByte(char data);
unsigned int spiTransferWord(int data);
unsigned int spiReciveWord(void);
#endif
```

Файл adc_ext.c

```
#include <iom128v.h>
unsigned int ReadADC_ext(void)
{
 unsigned int tmp=0;
 SPCR=(1<<MSTR);                // выбор режима работы "Master", скорость передачи fclk/4
  PORTE&=~(1<<PE7);             // выбор микросхемы АЦП
 asm("nop");                    // синхронизация перед считыванием выводов порта B
 while(PINB&(1<<PB3))           // ожидание появления лог.0 на выходе Dout
   {                            // формирование предварительных импульсов для АЦП
    PORTB|=(1<<PB1);
    PORTB&=~(1<<PB1);
    asm("nop");                 // синхронизация перед считыванием выводов порта B
   }
```

```
 PORTB|=(1<<PB1);
 PORTB&=~(1<<PB1);
 SPCR|=(1<<SPE);                  // разрешение работы модуля SPI при частоте fclk/4:
 SPSR|=(1<<SPI2X);                // включение бита SPI2X – бита удвоения скорости обмена
 SPDR=123;                        // запись в регистр SPDR произвольного числа
 while(!(SPSR & (1<<SPIF)));      // ожидание установки флага "Конец передачи" – бита
SPIF
 tmp=SPDR<<2;                     // запись ст. байта в результат преобразования
                                  // для 16-разрядного АЦП команда имеет вид:
                                  // tmp=SPDR<<8;
 SPDR=123;
while(!(SPSR & (1<<SPIF)));       // ожидание установки флага "Конец передачи" – бита
SPIF
tmp|=SPDR;                        // дополнение результата младшим байтом
 SPCR&=~(1<<SPE);                 // запрещение работы модуля SPI
PORTE|=(1<<PE7);                  // отключение сигнала CS – сигнала выбор микросхемы
АЦП"
return tmp;
}
```

Файл adc_ext.h

```
#ifndef SPI_H
#define SPI_H
void spiInit(void);
void spiSendByte(char data);
char spiTransferByte(char data);
unsigned int spiTransferWord(int data);
unsigned int spiReciveWord(void);
#endif
```

СПИСОК ЛИТЕРАТУРЫ

1. Техническое описание AVR-микроконтроллера ATmega128 с программируемой памятью 128 кбайт / http://usbsergdev.narod.ru/DOC/ATmega128rus.pdf

2. Евстифеев, А. В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL / А. В. Евстифеев. – М. : Додэка-21, 2005. – 558 с.

3. Новиков, Ю. В. Основы микропроцессорной техники / Ю. В. Новиков, П. К. Скоробогатов. – М. : Бином, 2006. – 357 с.

4. Баранов, В. Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы / В. Н. Баранов. – М.: Додека-21, 2004. – 288 с.

5. Голубцов, М. С. Микроконтроллеры AVR: от простого к сложному / М. С. Голубцов. – М. : СОЛОН-Пресс, 2003. – 288 с.

6. Трамперт, В. AVR-RISC микроконтроллеры / В. Трамперт. – Киев : МК-Пресс, 2006. – 464 с.

7. Шпак, Ю. А. Программирование на языке С для AVR и PIC микроконтроллеров / Ю. А. Шпак. – Киев : МК-Пресс, 2011. – 544 с.

8. Китаев, Ю. В. Основы программирования микроконтроллеров Atmega128 и 68hc908 / Ю. В. Китаев. – СПб. : СПбГУ ИТМО, 2007. – 107 с.

Учебное электронное издание

СМИРНОВ Виталий Иванович

ПРОЕКТИРОВАНИЕ И СХЕМОТЕХНИЧЕСКОЕ МОДЕЛИРОВАНИЕ МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ

Учебное пособие

Редактор М. В. Теленкова

Объем данных 1,97 Мб. ЭИ № 165.

Печатное издание
ЛР №020649 от 22.10.97.
Подписано в печать 28.10.2013. Формат 60×84/16.
Усл. печ. л. 6,98. Тираж 50 экз. Заказ 1002.

Ульяновский государственный технический университет
432027, Ульяновск, Северный Венец, 32.
ИПК «Венец» УлГТУ, 432027, Ульяновск, Северный Венец, 32.
Тел.: (8422) 778-113.
E-mail: venec@ulstu.ru
http://www.venec.ulstu.ru