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

Факультет информационной безопасности

Кафедра защиты информации

Д. В. Столер

ПРОГРАММИРОВАНИЕ ОДНОКРИСТАЛЬНЫХ МИКРОКОНТРОЛЛЕРОВ НА C/C++. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

В двух частях

Часть 1

ПОРТЫ ВВОДА/ВЫВОДА, ПРЕРЫВАНИЯ И ТАЙМЕРЫ/СЧЕТЧИКИ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальностей 1-45 01 02 «Инфокоммуникационные системы (по направлениям)»,1-98 01 02 «Защита информации в телекоммуникациях»

Минск БГУИР 2023

УДК 004.383-022.53(076.5)
ББК 32.971.32-04я73
С81

Р е ц е н з е н т ы:

кафедра телекоммуникационных систем
учреждения образования «Белорусская государственная академия связи»
(протокол №1 от 31.08.2021);

ведущий научный сотрудник научно-исследовательской лаборатории
факультета связи и автоматизированных систем управления войсками
учреждения образования «Военная академия Республики Беларусь»
кандидат технических наук, доцент А. В. Хижняк

Столер, Д. В.

С81 Программирование однокристальных микроконтроллеров на С/С++. Лабораторный практикум. В 2 ч. Ч. 1 : Порты ввода/вывода, прерывания и таймеры/счетчики : учеб.-метод. пособие / Д. В. Столер. – Минск : БГУИР, 2023. – 155 с. : ил.

ISBN 978-985-543-726-1 (ч. 1).

Предназначено для изучения основ программирования 8-разрядных однокристальных микроконтроллеров семейства AVR с использованием высокоуровневых языков С/С++. Рассмотрены правила конфигурирования и управления портами ввода/вывода, ключевые принципы обработки прерываний по различным аппаратным событиям и основные режимы работы таймеров/счетчиков. Также уделяется внимание программным инструментам разработки приложений и моделирования работы электронных устройств.

Содержит первые четыре лабораторные работы по учебной дисциплине «Микропроцессорные устройства и системы» и литературу для подготовки студентов. Каждая работа включает теоретические сведения с примерами написания программ, лабораторное задание с делением по вариантам, содержание отчета и контрольные вопросы.

УДК 004.383-022.53(076.5)
ББК 32.971.32-04я73

ISBN 978-985-543-726-1 (ч. 1)
ISBN 978-985-543-675-2

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА №1

ИЗУЧЕНИЕ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТКИ ATMEL STUDIO И СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ ЭЛЕКТРОННЫХ СХЕМ PROTEUS VSM

Цель работы: получить навыки работы в интегрированной среде разработки и системе автоматизированного проектирования электронных схем; создать проект в Atmel Studio, отследить этапы выполнения управляющей программы и проверить ее работоспособность с помощью виртуального микроконтроллера в системе Proteus VSM.

1 Теоретические сведения

Лабораторные работы по дисциплине «Микропроцессорные устройства и системы» в учебном семестре будут посвящены изучению архитектуры 8-разрядных микроконтроллеров (МК) семейства AVR (Advanced Virtual RISC) корпорации Atmel на примере модели ATmega16, написанию исполняемых программ на языке программирования C/C++ в интегрированной среде разработки Atmel Studio и проверке их работоспособности с помощью виртуального микроконтроллера в системе автоматизированного проектирования электронных схем Proteus VSM.

Микроконтроллер представляет собой функционально законченную микропроцессорную систему (МПС) обработки данных, которая реализована в виде одной большой интегральной микросхемы. Для того чтобы понять основополагающие принципы работы МК, сначала необходимо остановиться на МПС.

1.1 Микропроцессорная система

МПС – это функционально законченное устройство, выполненное на основе центрального процессора (ЦП), памяти и устройств ввода/вывода и ряда других вспомогательных устройств для выполнения заданных функций, задач. На рисунке 1.1 представлена обобщенная структурная схема МПС.

Структура МПС является магистрально-модульной. В такой структуре имеется группа шин (магистраль), к которым подключаются различные модули, обменивающиеся между собой информацией по одним и тем же шинам поочередно. Шина – это набор параллельных электрических проводников, по которым передается цифровой сигнал. Эти проводники называются линиями шины. В каждый момент времени по шине передается одно двоичное число, т. е. по каждой линии передается один разряд этого числа.

Вся совокупность информационных данных, которая передается в МПС, разделяется на три основные группы: адреса, данные и сигналы управления. В соответствии с этим выделяют три шины нижнего уровня:

- шина адреса (ША, Address Bus);

- шина данных (ШД, Data Bus);
- шина управления (ШУ, Control Bus).

Все вместе эти три шины образуют системную шину (СШ, System Bus).

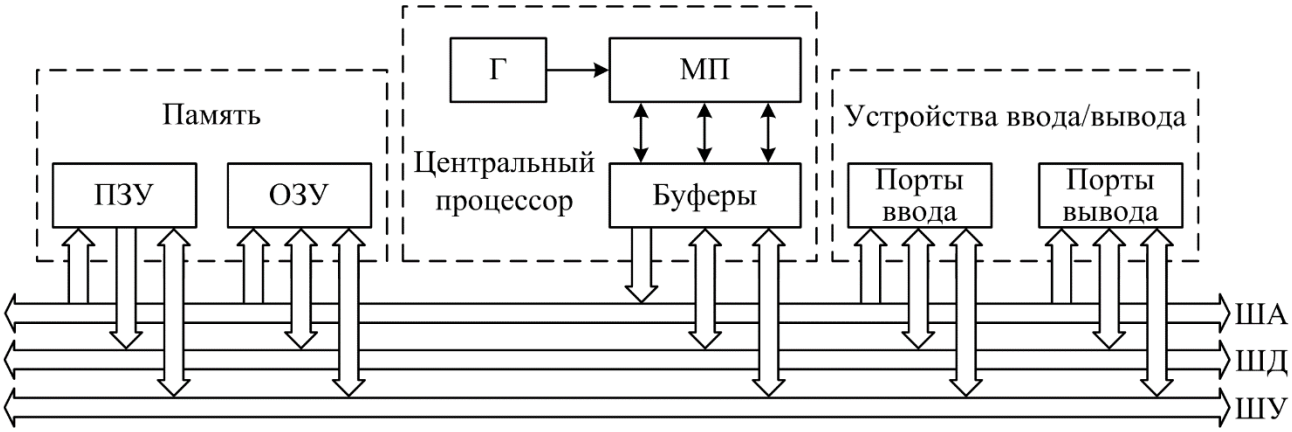


Рисунок 1.1 – Обобщенная структурная схема МПС

Шина адреса предназначена для однозначного определения адреса элемента МПС, например, ячейки памяти или устройства ввода/вывода. Обычно адрес задает ЦП, поэтому шина адреса чаще всего однонаправленная. Разрядность шины адреса может быть различной. От количества разрядов шины зависит то, какое количество ячеек памяти может адресовать ЦП. Объем адресуемой памяти определяется как 2^n байт, где n – разрядность шины адреса. Для адресации устройств ввода/вывода используется та же самая шина адреса, но т. к. обычно устройств ввода/вывода не так много, то используется не вся шина, а только несколько ее младших разрядов.

Шина данных служит для обмена данными между элементами МПС. Шина данных всегда двунаправленная. Разрядность шины определяется типом применяемого ЦП и всегда кратна 8, поэтому шина данных может быть 8-, 16-, 32-, 64-разрядной. Кратность 8 объясняется требованием, в соответствии с которым любой ЦП должен иметь возможность записать данные объемом в 1 байт в одну отдельную ячейку памяти или в отдельное устройство ввода/вывода, а также в обратном направлении прочитать этот байт данных.

Шина управления предназначена для управления работой элементов МПС. По ней передаются управляющие сигналы. Отдельные линии шины могут быть однонаправленными или двунаправленными.

Главным модулем МПС является **центральный процессор** (центральное процессорное устройство (ЦПУ, Central Processing Unit (CPU))), который считывает и выполняет команды управляющей программы, организует обращение к памяти, в нужных случаях инициирует работу устройств ввода/вывода. Основой модуля служит микропроцессор (МП). Кроме МП в ЦПУ обычно расположены генератор тактовых импульсов (Г) и буферы для сопряжения с системной шиной. Генератор обеспечивает тактирование МП. Буферы, выполненные на основе шинных формирователей, способствуют повышению нагрузочной спо-

собности выводов микросхемы МП и тем самым согласованию выводов МП с системной шиной.

Память МПС имеет две разновидности: постоянное запоминающее устройство (ПЗУ, Read Only Memory (ROM)) и оперативное запоминающее устройство (ОЗУ, Random Access Memory (RAM)). ПЗУ служит для хранения неизменяемых программ и констант. Оно является энергонезависимым и при выключении питания информацию не теряет. ОЗУ хранит оперативные данные (изменяемые программы, промежуточные результаты вычислений) и теряет свое содержимое вместе с отключением питания. Но ОЗУ в отличие от ПЗУ позволяет ЦПУ записывать данные, а не только читать их в процессе работы МПС.

С программной точки зрения память МПС представляет собой набор регистров (ячеек памяти), обычно 8-разрядных, предназначенных для хранения информации в двоичной форме. Каждая ячейка имеет уникальный адрес, что обеспечивает возможность доступа к ней.

Внешние устройства (ВУ) подключаются к МПС с помощью **устройств ввода/вывода**, реализующих определенные протоколы параллельного или последовательного обмена. Такими ВУ могут быть клавиатура, монитор, внешние запоминающие устройства, датчики, аналого-цифровые и цифроаналоговые преобразователи (АЦП, ЦАП), световые индикаторы, электродвигатели, реле и т. д. Устройства ввода/вывода в МПС чаще всего выполняются в виде портов ввода/вывода (Port Input/Output (Port I/O)). Порты могут быть двунаправленными, т. е. работать поочередно на ввод или вывод данных. ЦПУ работает с портами ввода/вывода практически так же, как и с ячейками памяти. Работа с портами сводится к тому, что процессор читает число из порта ввода или записывает число в порт вывода. С программной точки зрения порт ввода – это адрес устройства МПС, через которое из ВУ могут быть переданы данные на ШД. Соответственно порт вывода – это адрес устройства МПС, через которое данные с ШД передаются на ВУ. Порты ввода/вывода могут передавать данные в параллельном формате, тогда их называют параллельными портами ввода/вывода. Некоторые порты для сопряжения с ВУ могут использовать последовательный формат передачи данных, тогда их называют последовательными портами.

1.2 Однокристальный микроконтроллер

Микроконтроллер объединяет все основные элементы МПС: центральный процессор, память, устройства ввода/вывода, а также большой набор периферийных устройств, реализованных в виде модулей. МК предназначены для решения задач управления в различных отдельных устройствах и системах, встраиваемых в разнообразную аппаратуру, например, устройства промышленной автоматики, средства связи, измерительная и бытовая техника.

Может использоваться термин «однокристальный МК», который подчеркивает, что все устройство МК выполнено на одном кристалле (на одной полупроводниковой подложке) и размещается в одном корпусе микросхемы.

МК строятся по модульному принципу, когда все МК одного семейства содержат в себе одинаковый базовый функциональный блок и отличающийся изменяемый функциональный блок. Обобщенная структурная схема модульного МК представлена на рисунке 1.2.

Базовый функциональный блок принято называть процессорным ядром МК. Процессорное ядро обозначают именем семейства МК, основой которого оно является.

Процессорное ядро включает в себя:

- центральный процессор;
- внутреннюю контроллерную системную шину (ВКСШ), состоящую из шин адреса, данных и управления;
- схему тактирования МК, предназначенную для синхронизации работы центрального процессора, системной шины и периферийных модулей;
- схему управления режимами работы МК, например, режим микропотребления, экономичный режим, режим ожидания и т. д.

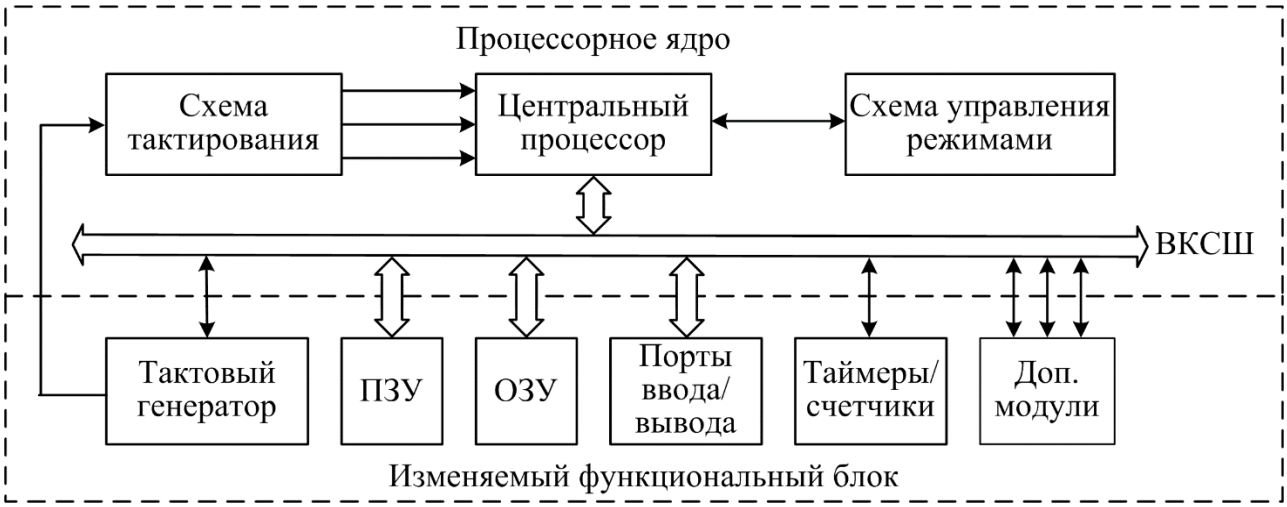


Рисунок 1.2 – Обобщенная структурная схема модульного микроконтроллера

Изменяемый функциональный блок включает в себя различные периферийные устройства, основными из которых являются:

- память различного типа и объема (ПЗУ и ОЗУ);
- порты ввода/вывода;
- тактовый генератор;
- таймеры/счетчики;
- дополнительные модули.

Дополнительные модули могут быть представлены следующими устройствами:

- процессоры событий;
- модуль обработки прерываний;
- модуль внутрисхемной отладки и программирования;
- контроллеры последовательного интерфейса различных типов;
- контроллеры напряжения питания и хода выполнения программы;

- контроллеры прямого доступа к памяти;
- контроллеры жидкокристаллических и светодиодных индикаторов;
- аналого-цифровой и цифроаналоговый преобразователи;
- аналоговый компаратор и т. д.

Каждый модуль имеет выводы для подключения его к внутренней контроллерной системной шине. Это позволяет на уровне функционального программирования новой модели МК подсоединить те или иные модули к шине процессорного ядра, создавая, таким образом, разнообразные по структуре микроконтроллеры в пределах одного семейства. Модули, объединенные в составе одного МК, размещаются на одном полупроводниковом кристалле.

МК характеризуются закрытой архитектурой, при которой линии внутренних шин адреса и данных отсутствуют на выводах корпуса МК. Поэтому при выборе МК для проектируемой системы следует убедиться, что управляющая программа сможет разместиться во внутренней памяти МК. В противном случае придется перейти к МК с большим объемом внутреннего ПЗУ. Для подобных случаев производители обычно предлагают ряд модификаций МК с одним и тем же набором периферийных модулей и различными объемами ПЗУ и ОЗУ.

Основным блоком ЦП однокристальных МК является арифметико-логическое устройство (АЛУ), которое выполняет арифметические и логические операции над входными данными. ЦП однокристальных МК может выполняться либо по CISC-архитектуре, либо по RISC-архитектуре.

Память в однокристальных МК может быть реализована либо по принстонской (фон Неймана) архитектуре, либо гарвардской архитектуре.

1.3 CISC- и RISC-архитектуры построения центрального процессора МПС

Центральный процессор МПС с точки зрения системы команд и способов адресации операндов может быть выполнен по одному из двух принципов построения:

- CISC-архитектура (Complex Instruction Set Computer), в которой реализуется полная система команд;
- RICS-архитектура (Reduced Instruction Set Computer), в которой реализуется сокращенная (упрощенная) система команд.

CISC-архитектура характеризуется большим набором разноформатных команд с использованием многочисленных способов адресации. Основной целью архитектуры было сокращение размера программ, что уменьшало требования к объему оперативной памяти. Для данной архитектуры характерно:

- большое количество команд, которые выполняют несколько и более функций;

- большое количество форматов команд различной разрядности (от 1 и более байт);

- большое количество способов адресации операндов;

- широкое использование команд обработки типа «регистр – память»;

- сравнительно небольшое число регистров общего назначения (8–16), которые наряду с ячейками памяти предназначены для хранения промежуточных результатов выполняемых операций, но обладают более скоростным доступом.

Многообразие команд и способов адресации позволяет программисту реализовать наиболее эффективные алгоритмы решения различных задач. Однако при этом существенно усложняется структура ЦП, особенно его устройства управления, что приводит к увеличению размеров и стоимости кристалла, снижению производительности МПС. В то же время многие команды и способы адресации используются достаточно редко. Кроме того, наличие в программе команд различного формата приводит к нерегулярности потока команд и сильно ограничивает эффективность их конвейерной обработки.

Эти недостатки обусловили необходимость разработки альтернативной архитектуры. RISC-архитектура обладает следующими особенностями:

- расширенное количество регистров общего назначения (от 32 и более);

- использование в командах обработки данных только регистровой адресации;

- отказ от аппаратной реализации сложных способов адресации;

- фиксированный формат команд (например, 4 байта);

- исключение из набора команд, реализующих редко используемые операции, а также команд, не вписывающихся в принятый формат.

Преимущественное использование регистровой адресации значительно повышает производительность ЦП. Фиксированный формат команд, отказ от сложных и редко используемых команд и способов адресации существенно упрощает устройство управления, сокращает объем используемой памяти, что позволяет уменьшить размер кристалла, снизить его стоимость и повысить тактовую частоту. Использование фиксированного формата команд обеспечивает также более эффективную работу конвейера ЦП, уменьшает число тактов простоя и ожидания, что дает дополнительный рост производительности.

Технология конвейеризации заключается в том, что во время выполнения текущей команды производится обращение к памяти, выборка из нее и дешифрация машинного кода следующей команды.

1.4 Принстонская и гарвардская архитектуры построения памяти

Важной архитектурной особенностью построения МПС является используемый вариант реализации памяти и организация выборки команд и данных. По этим признакам различают МПС с принстонской (фон Неймана) и гарвардской архитектурой.

Принстонская (фон Неймана или фон-неймановская) архитектура характеризуется общей оперативной памятью для хранения команд программы и данных (рисунок 1.3). Для обращения к этой памяти используется общая системная шина, по которой в ЦП поступают и команды, и данные.

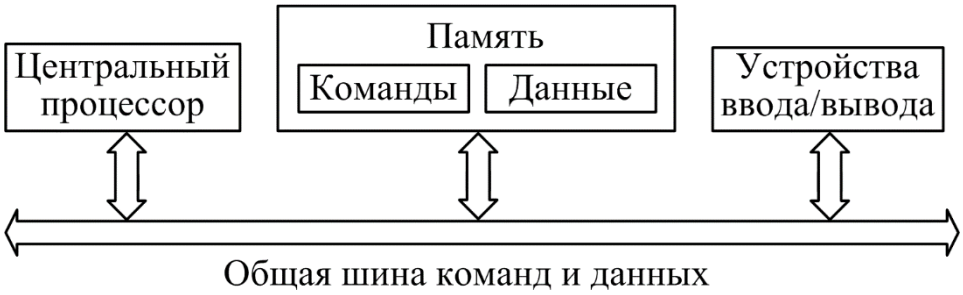


Рисунок 1.3 – Принстонская архитектура

Принстонская архитектура имеет ряд достоинств. Наличие общей памяти позволяет оперативно перераспределять ее объем для хранения отдельных массивов команд и данных в зависимости от решаемых задач. Обеспечивается возможность более эффективного использования имеющегося объема оперативной памяти в каждом конкретном случае применения МПС. Например, в некоторых случаях нужна большая и сложная программа, а данных в памяти надо хранить не слишком много. В других случаях, наоборот, требуется простая программа, но необходимы большие объемы хранимых данных (например, в системах сбора данных от многих объектов). Перераспределение памяти не вызывает никаких проблем, главное – чтобы программа и данные помещались в памяти. Как правило, в системах с такой архитектурой память бывает довольно большого объема. Это позволяет решать самые сложные задачи. Использование общей шины для передачи команд и данных значительно упрощает отладку, тестирование и текущий контроль функционирования системы, повышает ее надежность.

Однако ей присущи и существенные недостатки. Дело в том, что при единственной шине команд и данных ЦП вынужден по одной шине принимать данные (из памяти или устройств ввода/вывода) и передавать данные (в память или устройства ввода/вывода), а также читать команды из памяти. Естественно, одновременно эти пересылки данных по шине происходить не могут, они должны производиться по очереди. При этом общая шина становится «узким местом», которое ограничивает производительность системы.

Эти недостатки отсутствуют в гарвардской архитектуре, которая характеризуется физическим разделением памяти команд и памяти данных. Обмен ЦП с каждым из двух типов памяти происходит по своей системной шине. Гарвардская архитектура представлена на рисунке 1.4.

Каждая память соединяется с ЦП отдельной шиной, что позволяет одновременно с чтением/записью данных при выполнении текущей команды производить выборку следующей команды. Благодаря такому разделению потоков

команд и данных и совмещению операций их выборки и исполнения реализуется более высокая производительность, чем при использовании принстонской архитектуры.

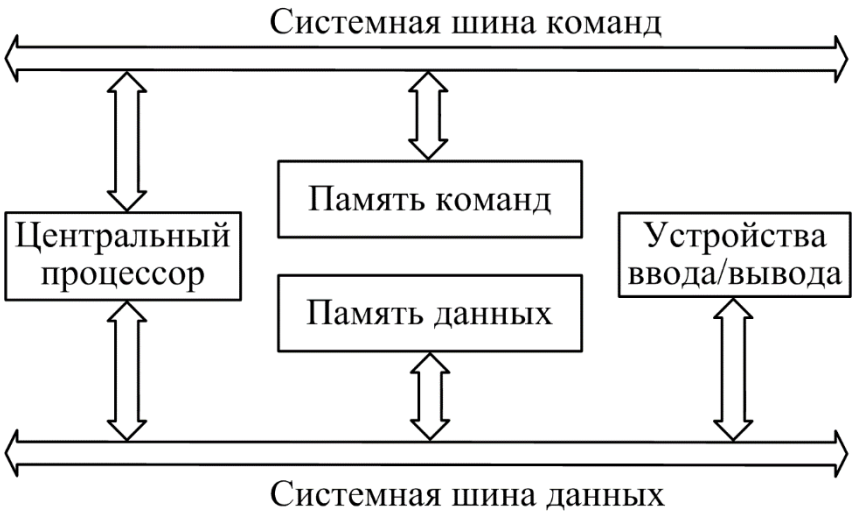


Рисунок 1.4 – Гарвардская архитектура

Недостатки гарвардской архитектуры связаны с необходимостью использования большего числа шин, усложнения структуры процессора. Кроме того, существенный недостаток – это фиксированный объем памяти, выделенной для команд и данных, назначение которой не может оперативно перераспределяться в соответствии с требованиями решаемой задачи. Поэтому приходится применять память большего объема, коэффициент использования которой при решении разнообразных задач оказывается более низким, чем в системах с принстонской архитектурой.

Однако развитие микро- и наноэлектронных технологий позволило в значительной степени преодолеть указанные недостатки, поэтому гарвардская архитектура широко применяется во внутренней структуре современных ЦП, где используется отдельная кэш-память для хранения команд и данных. В то же время во внешней структуре большинства МПС реализуются принципы принстонской архитектуры.

Проще всего преимущества двухшинной архитектуры реализуются внутри одной микросхемы. В этом случае можно также существенно уменьшить влияние недостатков этой архитектуры. Поэтому гарвардская архитектура получила преимущественное распространение в однокристальных МК, управляющая программа которых обычно хранится в отдельном ПЗУ. В МК программы обычно небольшие, но зато необходимо максимальное быстродействие при заданной тактовой частоте.

1.5 Микроконтроллеры семейства AVR

Семейство микроконтроллеров AVR было создано в 1996 г. американской компанией Atmel Corporation (далее – Atmel). В основе архитектуры микроконтроллеров была использована идея двух студентов Норвежского университета науки и технологий Альфа Богена (Alf-Egil Bogen) и Вегарда Воллена (Vegard Wollen), которые предложили выпускать новый 8-разрядный RISC-микроконтроллер и снабдить его перепрограммируемым ПЗУ (flash-памятью) для программ на одном полупроводниковом кристалле с вычислительным ядром. В конце 1996 г. был выпущен опытный микроконтроллер AT90S1200, а во второй половине 1997 г. компания Atmel приступила к серийному производству нового семейства микроконтроллеров, их рекламной и технической поддержке. Новое ядро было запатентовано и получило название AVR. Существует несколько трактовок данной аббревиатуры. По одной версии название происходит от первых букв имен разработчиков – A и V, и первой буквы аббревиатуры архитектуры RISC, которая лежит в основе семейства микроконтроллеров. По другой версии аббревиатура может быть расшифрована как Advanced Virtual RISC (передовая фактическая RISC-архитектура).

В январе 2016 г. компания Atmel была приобретена компанией Microchip Technology Inc. (далее – Microchip), которая является производителем микроконтроллеров семейства PIC, поэтому на текущий момент времени разработкой новых моделей МК и вопросами технической поддержки МК семейства AVR занимается компания Microchip, а на корпусах выпускаемых моделей МК размещается товарный знак Microchip.

Значительная часть микроконтроллеров AVR является 8-разрядными, что означает наличие 8-разрядной шины данных и соответственно способность МК принимать, обрабатывать и передавать 8-разрядные данные, т. е. объемом 1 байт. В основе архитектуры лежат регистры общего назначения, которые также являются 8-разрядными и используются для хранения исходных данных выполняемой операции и ее результата. ЦП берет данные из двух входных рабочих регистров, выполняет арифметическую или логическую операцию и сохраняет результат в выходном регистре. Также существуют и 32-разрядные микроконтроллеры AVR32.

ЦП микроконтроллеров AVR строится по RISC-архитектуре, а память организуется по гарвардской архитектуре. Данные характеристики делают возможным использование технологии конвейеризации, благодаря которой большинство команд ЦП микроконтроллера выполняет за один период тактового сигнала.

Компанией Atmel был разработан обширный набор 8-разрядных микроконтроллеров AVR, разбитых на несколько подсемейств в соответствии с их характеристиками и областью применения. Рассмотрим эти подсемейства:

1) tiny – микроконтроллеры, которые характеризуются небольшим объемом памяти (до 8 Кбайт), небольшим количеством выводов, что позволяет снизить потребление энергии и размеры корпуса, подходят для самых простых задач;

2) mega – это более распространенное подсемейство, микроконтроллеры которого имеют большой объем встроенной памяти (до 256 Кбайт), множество модулей встроенной периферии и предназначены для задач средней и высокой сложности;

3) xmega – микроконтроллеры, которые используются в сложных коммерческих задачах, требующих большого объема памяти, высокой производительности и низкую потребляемую мощность.

1.6 Микроконтроллер ATmega16

Микроконтроллер ATmega16 является 8-разрядным, относится к семейству AVR, подсемейству mega, изготовлен по малопотребляющей КМОП-технологии, строится по гарвардской и RISC архитектурам, обладает технологией конвейеризации.

Полная маркировка изучаемой модели МК:

$$\underbrace{AT}_{1}\ \underbrace{mega}_{2}\ \underbrace{16}_{3}\ \underbrace{A}_{4}\ -\ \underbrace{16}_{5}\ \underbrace{P}_{6}\ \underbrace{U}_{7}\ /\ \underbrace{DIP40}_{8}.$$

Рассмотрим обозначения маркировки, которые условно сведены в восемь групп:

1 – разработчик – компания Atmel.

2 – подсемейство микроконтроллеров AVR.

3 – объем встроенной flash-памяти (Кбайт). При этом должно выполняться условие 2^n, где n – натуральные числа. Существуют модели МК, где будет указано трехзначное число, например, 169. В этом случае 9 обозначает модификацию МК, а объем flash-памяти будет оставаться 16 Кбайт. Чтобы узнать, в чем заключается данная модификация, необходимо обратиться к технической спецификации (Data sheet), которую можно скачать на официальном сайте компании Microchip (microchip.com). Для этого необходимо на главной веб-странице сайта последовательно пройти по следующим пунктам: **TOOLS AND RESOURCES → Documentation → Data Sheets**, затем в поле для поиска с поясняющим названием **Search Documents** ввести название требуемой модели МК, после чего в нижней части веб-страницы появится список файлов.

4 – вид энергопотребления: L – версии МК, работающие на пониженном напряжении питания (2,7 В); V – версии МК, работающие на низком напряжении питания (1,8 В); A и P – версии МК, имеющие низкий потребляемый ток за счет применения технологии picoPower.

Энергопотребление влияет на величину используемой максимальной тактовой частоты (сравнительная характеристика представлена в таблице 1.1).

Таблица 1.1 – Зависимость тактовой частоты от вида энергопотребления

Модель МК	Рабочее напряжение, В	Тактовая частота, МГц
ATmega16	4,5–5,5	0–16
ATmega16L	2,7–5,5	0–8
ATmega16A	2,7–5,5	0–16

5 – предельно допустимая тактовая частота в мегагерцах.

6 – тип корпуса для защиты от внешних воздействий, например: P – корпус DIP (Dual In-line Package). Представляет собой прямоугольный корпус с двумя рядами контактов, расположенными на длинных сторонах. Контакты имеют большие размеры, их можно легко запаять, вставить в макетную плату. Корпус может быть выполнен из пластика (PDIP) или керамики (CDIP). S – корпус SOIC (Small-Outline Integrated Circuit); A – корпус TQFP (Thin Quad Flat Pack); M – корпус MLF (Micro Lead Frame). Корпусы SOIC и TQFP имеют компактные размеры и предназначены для поверхностного монтажа. Контакты и расстояние между ними небольшие, поэтому главным образом используются в промышленном производстве электронных устройств.

7 – температурный диапазон (–40...+85 °C) и тип лужения выводов: U – бессвинцовый припой; I – свинцовый.

8 – количество выводов и тип корпуса, которые дополнительно могут указываться через слеш.

Так как в лабораторных работах проверка работоспособности программы будет осуществляться с помощью виртуального микроконтроллера в программе Proteus ISIS, где вид энергопотребления и тип корпуса не имеют принципиального значения, то определяющей остается только первая часть маркировки, а именно ATmega16, которая и будет в дальнейшем использована при упоминании этой модели в среде разработки Atmel Studio и программе моделирования Proteus ISIS.

Технические характеристики МК ATmega16A следующие:

- flash-память программ объемом 16 Кбайт (число циклов стирания/записи не менее 10 000);
- оперативная память (статическое ОЗУ) объемом 1 Кбайт;
- память данных на основе электрически стираемого перепрограммируемого ПЗУ (ЭСППЗУ, Electrically Erasable Programmable Read-Only Memory (EEPROM)) объемом 512 байт (число циклов стирания/записи не менее 100 000);
- возможность защиты от чтения и модификации памяти программ и данных;
- возможность программирования непосредственно в системе через последовательные интерфейсы SPI и JTAG;
- возможность внутрисхемного программирования (In-System Programming (ISP)) встроенной загрузочной программой;
- возможность внутрисхемной отладки в соответствии со стандартом IEEE 1149.1 (JTAG);
- разнообразные способы тактирования (синхронизации): встроенный RC-генератор с внутренней или внешней времязадающей RC-цепочкой, встроенный генератор с внешним кварцевым или пьезокерамическим резонатором, внешний сигнал тактирования;
- шесть режимов снижения энергопотребления: режим холостого хода (Idle), режим снижения шумов АЦП (ADC Noise Reduction), экономичный ре-

жим (Power-save), режим микропотребления (Power-down), режим ожидания (Standby) и расширенный режим ожидания (Extended Standby);

- наличие детектора пониженного напряжения питания (Brown-Out Detection (BOD));
- возможность программного выбора тактовой частоты;
- статическая архитектура (минимальная тактовая частота может быть равна нулю);
- АЛУ подключено непосредственно к 32 регистрам общего назначения;
- векторная система прерываний, поддержка очереди прерываний;
- наличие аппаратного умножителя;
- 131 высокопроизводительная команда, многие из которых выполняются за один период тактового сигнала;
- производительность порядка 16 MIPS (Million instructions per second) при тактовой частоте 16 МГц;
- программное конфигурирование и выбор портов ввода/вывода;
- выводы могут быть запрограммированы как входные или как выходные независимо друг от друга;
- входные буферы с триггером Шмитта на всех выводах;
- на входах портов ввода/вывода имеются индивидуально отключаемые внутренние подтягивающие (к напряжению питания) резисторы сопротивлением 20–50 кОм;
- рабочее напряжение – 2,7–5,5 В;
- тактовая частота – 0–16 МГц.

МК ATmega16A имеет богатый набор **встроенных периферийных устройств**:

- два 8-разрядных таймера/счетчика TC0 и TC2 с отдельными предварительными делителями и режимами сравнения. TC2 может работать в качестве часов реального времени (в асинхронном режиме) с отдельным внешним тактовым генератором;
- один 16-разрядный таймер/счетчик TC1 с отдельным предварительным делителем, режимом сравнения и режимом захвата;
- сторожевой таймер;
- два одноканальных генератора 8-разрядного сигнала с широтно-импульсной модуляцией (ШИМ) (один из режимов работы 8-разрядных таймеров/счетчиков);
- один двухканальный генератор 16-разрядного ШИМ-сигнала (один из режимов работы 16-разрядного таймера/счетчика);
- аналоговый компаратор;
- 8-канальный 10-разрядный АЦП последовательного приближения, имеющий несимметричные и дифференциальные входы;
- последовательный синхронный интерфейс SPI (Serial Peripheral Interface) с поддержкой режимов ведущий/ведомый (Master/Slave);

- полнодуплексный универсальный синхронный/асинхронный приемопередатчик последовательного типа USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter);

- байт-ориентированный двухпроводной последовательный интерфейс TWI (Two Wire Interface).

Электрические характеристики МК ATmega16A представлены в таблицах 1.2 и 1.3.

Таблица 1.2 – Предельно допустимые параметры

Параметр	Значение
Рабочая температура	–55…+125 °С
Температура хранения	–65…+150 °С
Напряжение на любом выводе относительно вывода GND (кроме вывода сброса $\overline{\text{RESET}}$)	–0,5…V_{CC} + 0,5 В
Напряжение на выводе $\overline{\text{RESET}}$ относительно вывода GND	–0,5…+13,0 В
Максимальное рабочее напряжение питания V_{CC}	6,0 В
Максимальный постоянный ток через линию ввода/вывода	40,0 мА
Максимальный постоянный ток через выводы V_{CC} и GND: - в корпусе PDIP - в корпусе TQFP/MLF	 200,0 мА 400,0 мА

Выход за предельно допустимые параметры может вызвать необратимое повреждение микроконтроллера. Вышепредставленные характеристики указывают на перегрузочные способности МК, не следует их использовать как рабочие характеристики. Работа в условиях, близких к предельно допустимым параметрам, может повлиять на надежность работы микроконтроллера.

Таблица 1.3 – Статические характеристики при $T_{окр.ср}$, равной –40…+85 °С, и V_{CC}, равном 2,7–5 В, если не указаны другие условия измерения

Обозначение	Параметр	Условия измерения	Минимальное	Номинальное	Максимальное	Единица измерения
1	2	3	4	5	6	7
V_{IL}	Входное напряжение низкого уровня (Input Low Voltage)	Кроме выводов XTAL1 и $\overline{\text{RESET}}$	–0,5	–	$0{,}2V_{CC}$[1]	В
V_{IH}	Входное напряжение высокого уровня (Input High Voltage)	Кроме выводов XTAL1 и $\overline{\text{RESET}}$	$0{,}6V_{CC}$[2]	–	V_{CC} + 0,5	В
V_{IL1}	Входное напряжение низкого уровня	Вывод XTAL1, выбрана внешняя синхронизация	–0,5	–	$0{,}1V_{CC}$[1]	В

Продолжение таблицы 1.3

1	2	3	4	5	6	7
V_{IH1}	Входное напряжение высокого уровня	Вывод XTAL1, выбрана внешняя синхронизация	$0{,}7V_{CC}$[2]	–	V_{CC} + 0,5	В
V_{IL2}	Входное напряжение низкого уровня	Вывод сброса $\overline{\text{RESET}}$	–0,5	–	$0{,}2V_{CC}$[1]	В
V_{IH2}	Входное напряжение высокого уровня	Вывод сброса $\overline{\text{RESET}}$	$0{,}9V_{CC}$[2]	–	V_{CC} + 0,5	В
V_{OL}	Выходное напряжение низкого уровня[3] (порты A, B, C, D) (Output Low Voltage)	I_{OL} = 20 мА, V_{CC} = 5 В	–	–	0,7	В
		I_{OL} = 10 мА, V_{CC} = 3 В	–	–	0,5	
V_{OH}	Выходное напряжение высокого уровня[4] (порты A, B, C, D) (Output High Voltage)	I_{OL} = –20 мА, V_{CC} = 5 В	4,2	–	–	В
		I_{OL} = –10 мА, V_{CC} = 3 В	2,2	–	–	
I_{IL}	Входной ток утечки через линию ввода/вывода (Input Leakage Current I/O Pin)	V_{CC} = 5,5 В, на выводе – логический 0 (абсолютная величина)	–	–	1	мкА
I_{IH}	Входной ток утечки через линию ввода/вывода (Input Leakage Current I/ O Pin)	V_{CC} = 5,5 В, на выводе – логическая 1 (абсолютная величина)	–	–	1	мкА
R_{RST}	Сопротивление подтягивающего резистора в цепи сброса (Reset Pull-up Resistor)	–	30	60	85	кОм
R_{PU}	Сопротивление подтягивающего резистора на линиях ввода/вывода (I/O Pin Pull-up Resistor)	–	20	–	50	кОм
I_{CC}	Потребляемый ток (Power Supply Current)	Рабочий режим (Active):				
		f = 1 МГц, V_{CC} = 3 В	–	0,6	–	мА
		f = 4 МГц, V_{CC} = 3 В	–	1,9	5	мА
		f = 8 МГц, V_{CC} = 5 В	–	7	15	мА
		Режим холостого хода (Idle Mode):				
		f = 1 МГц, V_{CC} = 3 В	–	0,2	–	мА
		f = 4 МГц, V_{CC} = 3 В	–	0,6	2	мА
		f = 8 МГц, V_{CC} = 5 В	–	2,7	7	мА
		Режим микропотребления[5] (Power-down Mode):				
		сторожевой таймер включен, V_{CC} = 3 В	–	< 8	15	мкА
		сторожевой таймер выключен, V_{CC} = 3 В	–	< 1	4	мкА
V_{ACIO}	Смещение входного напряжения аналогового компаратора (Analog Comparator Input Offset Voltage)	V_{CC} = 5 В $V_{IN} = V_{CC} / 2$	–	–	40	мВ
I_{ACLK}	Входной ток утечки аналогового компаратора (Analog Comparator Input Leakage Current)	V_{CC} = 5 В $V_{IN} = V_{CC} / 2$	–50	–	50	нА

Продолжение таблицы 1.3

1	2	3	4	5	6	7
t_{ACPD}	Задержка распространения сигнала в аналоговом компараторе (Analog Comparator Propagation Delay)	V_{CC} = 2,7 В V_{IN} = 4,0 В	–	750 500	–	нс

Примечания

[1] «Максимальное» означает наибольшее значение напряжения, приложенное к выводу, которое гарантированно распознается как логический 0 (лог. 0).

[2] «Мин» означает наименьшее значение напряжения, приложенное к выводу, которое гарантированно распознается как логическая 1 (лог. 1).

[3] Несмотря на то что каждая линия ввода/вывода может пропускать больший ток, чем в условиях измерения (20 мА при питании V_{CC} = 5 В; 10 мА при питании V_{CC} = 3 В) в статическом состоянии (не переходном), необходимо учитывать следующее:

а) в корпусе PDIP:

- суммарные выходные токи низкого уровня I_{OL} для линий ввода/вывода всех портов не должны превышать 200 мА;
- суммарные выходные токи низкого уровня I_{OL} для линий ввода/вывода порта A не должны превышать 100 мА;
- суммарные выходные токи низкого уровня I_{OL} для линий ввода/вывода портов B, C, D и вывода XTAL2 не должны превышать 100 мА;

б) в корпусах TQFP/MLF:

- суммарные выходные токи низкого уровня I_{OL} для линий ввода/вывода всех портов не должны превышать 400 мА;
- суммарные выходные токи низкого уровня I_{OL} для линий ввода/вывода порта A не должны превышать 100 мА;
- суммарные выходные токи низкого уровня I_{OL} для линий ввода/вывода порта B не должны превышать 100 мА;
- суммарные выходные токи низкого уровня I_{OL} для линий ввода/вывода порта C не должны превышать 100 мА;
- суммарные выходные токи низкого уровня I_{OL} для линий ввода/вывода порта D и вывода XTAL2 не должны превышать 100 мА.

Если выходной ток низкого уровня I_{OL} превышает условия измерения, то выходное напряжение низкого уровня V_{OL} может также увеличиться. Не гарантируется, что выводы не будут пропускать ток, превышающий указанный в условиях измерения.

[4] Несмотря на то что каждая линия ввода/вывода может быть источником более большого тока, чем в условиях измерения (20 мА при питании V_{CC} = 5 В;

10 мА при питании V_{CC} = 3 В) в статическом состоянии (не переходном), необходимо учитывать следующее:

а) в корпусе PDIP:

- суммарные выходные токи высокого уровня I_{OH} для линий ввода/вывода всех портов не должны превышать 200 мА;
- суммарные выходные токи высокого уровня I_{OH} для линий ввода/вывода порта A не должны превышать 100 мА;
- суммарные выходные токи высокого уровня I_{OH} для линий ввода/вывода портов B, C, D и вывода XTAL2 не должны превышать 100 мА;

б) в корпусах TQFP/MLF:

- суммарные выходные токи высокого уровня I_{OH} для линий ввода/вывода всех портов не должны превышать 400 мА;
- суммарные выходные токи высокого уровня I_{OH} для линий ввода/вывода порта A не должны превышать 100 мА;
- суммарные выходные токи высокого уровня I_{OH} для линий ввода/вывода порта B не должны превышать 100 мА;
- суммарные выходные токи высокого уровня I_{OH} для линий ввода/вывода порта C не должны превышать 100 мА;
- суммарные выходные токи высокого уровня I_{OH} для линий ввода/вывода порта D и вывода XTAL2 не должны превышать 100 мА.

[5] Минимальное напряжение питания V_{CC} для режима микропотребления составляет 2,5 В.

Упрощенная структурная схема МК ATmega16 показана на рисунке 1.5, на которой обозначены следующие выводы: V_{CC} – вывод для подключения источника питания МК; GND – общий вывод (земля); AVCC – вывод для подключения источника питания АЦП; AREF – вход опорного напряжения для АЦП; XTAL1 – вход инвертирующего усилителя внутреннего тактового генератора и вход внешнего источника тактового сигнала; XTAL2 – выход инвертирующего усилителя внутреннего тактового генератора; $\overline{\text{RESET}}$ – вход сигнала сброса (перезагрузки МК); PA0–PA7 – выводы порта A; PB0–PB7 – выводы порта B; PC0–PC7 – выводы порта C; PD0–PD7 – выводы порта D.

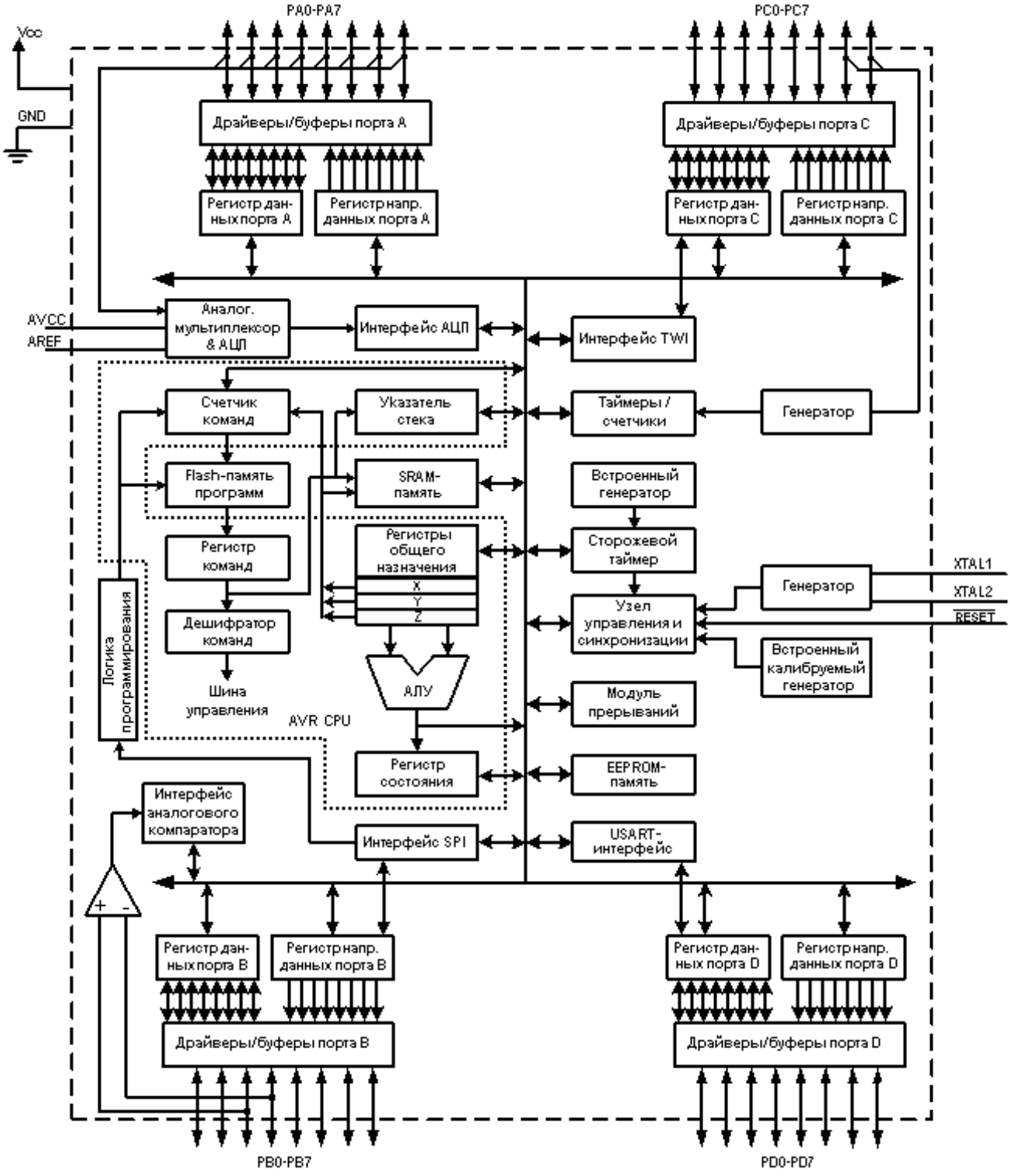
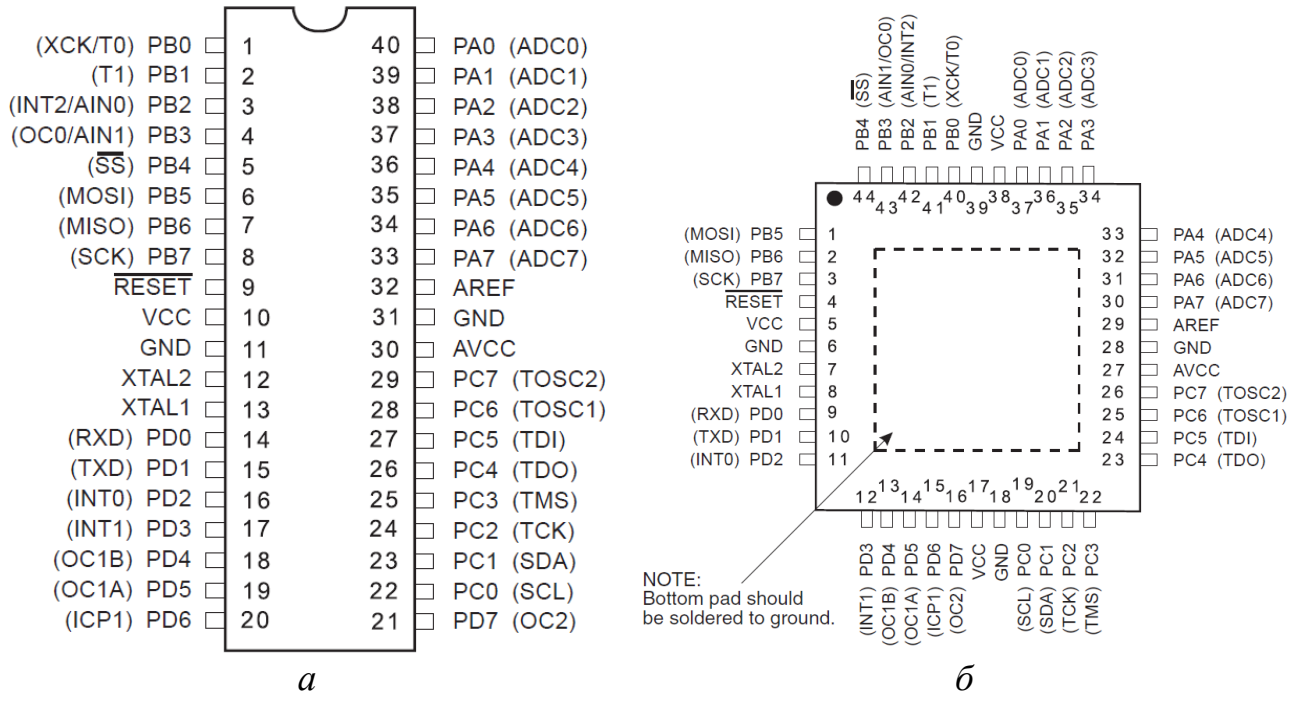


Рисунок 1.5 – Упрощенная структурная схема микроконтроллера ATmega16

Расположение выводов (цоколевка) МК ATmega16 в корпусах PDIP и TQFP/MLF показано на рисунке 1.6.



а *б*

а – корпус PDIP; *б* – корпус TQFP/MLF

Рисунок 1.6 – Расположение выводов микроконтроллера ATmega16

Микроконтроллер ATmega16 имеет четыре 8-разрядных параллельных порта A, B, C и D. Помимо функций ввода/вывода данных все порты имеют альтернативные функции (условные обозначения указаны в скобках на рисунке 1.6), которые используются различными периферийными устройствами МК. При этом возможны две ситуации. В одних случаях требуется самостоятельно задавать конфигурацию вывода, а в других вывод конфигурируется автоматически при включении соответствующего периферийного устройства. Описание всех выводов МК ATmega16 приведено в таблице 1.4, в которой использованы следующие обозначения типов вывода: I – вход; O – выход; I/O – вход/выход; P – вывод питания.

Таблица 1.4 – Описание выводов микроконтроллера ATmega16

Обозначение вывода	Номер вывода		Тип вывода	Описание
	PDIP	TQFP/MLF		
1	2	3	4	5
RESET	9	4	I	Вход сигнала сброса
VCC	10	5, 17, 38	P	Вывод источника питания
GND	11, 31	6, 18, 28, 39	P	Общий вывод
XTAL1	13	8	I	Вход тактового генератора
XTAL2	12	7	O	Выход тактового генератора

Продолжение таблицы 1.4

1	2	3	4	5
AVCC	30	27	P	Вывод источника питания АЦП
AREF	32	29	I	Вход опорного напряжения для АЦП
Порт A. 8-разрядный двунаправленный порт ввода/вывода с внутренними подтягивающими резисторами				
PA0 (ADC0)	40	37	I/O	0-я линия порта A (вход АЦП)
PA1 (ADC1)	39	36	I/O	1-я линия порта A (вход АЦП)
PA2 (ADC2)	38	35	I/O	2-я линия порта A (вход АЦП)
PA3 (ADC3)	37	34	I/O	3-я линия порта A (вход АЦП)
PA4 (ADC4)	36	33	I/O	4-я линия порта A (вход АЦП)
PA5 (ADC5)	35	32	I/O	5-я линия порта A (вход АЦП)
PA6 (ADC6)	34	31	I/O	6-я линия порта A (вход АЦП)
PA7 (ADC7)	33	30	I/O	7-я линия порта A (вход АЦП)
Порт B. 8-разрядный двунаправленный порт ввода/вывода с внутренними подтягивающими резисторами				
PB0 (T0/XCK)	1	40	I/O	0-я линия порта B (вход внешнего тактового сигнала таймера/счетчика TC0 / вход/выход внешнего тактового сигнала интерфейса USART)
PB1 (T1)	2	41	I/O	1-я линия порта B (вход внешнего тактового сигнала таймера/счетчика TC1)
PB2 (AIN0/INT2)	3	42	I/O	2-я линия порта B (неинвертирующий вход компаратора / вход внешнего прерывания 2)
PB3 (AIN1/OC0)	4	43	I/O	3-я линия порта B (инвертирующий вход компаратора / выход таймера/счетчика TC0)
PB4 ($\overline{SS}$)	5	44	I/O	4-я линия порта B (выбор ведомого устройства на шине интерфейса SPI)
PB5 (MOSI)	6	1	I/O	5-я линия порта B (выход (ведущего) или вход (ведомого) передачи данных интерфейса SPI)
PB6 (MISO)	7	2	I/O	6-я линия порта B (вход (ведущего) или выход (ведомого) передачи данных интерфейса SPI)
PB7 (SCK)	8	3	I/O	7-я линия порта B (выход (ведущего) или вход (ведомого) тактового сигнала интерфейса SPI)
Порт C. 8-разрядный двунаправленный порт ввода/вывода с внутренними подтягивающими резисторами				
PC0 (SCL)	22	19	I/O	0-я линия порта C (вход/выход тактового сигнала интерфейса TWI)
PC1 (SDA)	23	20	I/O	1-я линия порта C (вход/выход данных TWI)
PC2 (TCK)	24	21	I/O	2-я линия порта C (тактовый сигнал JTAG)
PC3 (TMS)	25	22	I/O	3-я линия порта C (выбор режима JTAG)
PC4 (TDO)	26	23	I/O	4-я линия порта C (выход данных JTAG)
PC5 (TDI)	27	24	I/O	5-я линия порта C (вход данных JTAG)
PC6 (TOSC1)	28	25	I/O	6-я линия порта C (вывод для подключения резонатора к таймеру/счетчику TC2)
PC7 (TOSC2)	29	26	I/O	7-я линия порта C (вывод для подключения резонатора к таймеру/счетчику TC2)

Продолжение таблицы 1.4

1	2	3	4	5
Порт D. 8-разрядный двунаправленный порт ввода/вывода с внутренними подтягивающими резисторами				
PD0 (RXD)	14	9	I/O	0-я линия порта D (вход данных USART)
PD1 (TXD)	15	10	I/O	1-я линия порта D (выход данных USART)
PD2 (INT0)	16	11	I/O	2-я линия порта D (вход внешнего прерывания 0)
PD3 (INT1)	17	12	I/O	3-я линия порта D (вход внешнего прерывания 1)
PD4 (OC1B)	18	13	I/O	4-я линия порта D (выход B таймера/счетчика TC1)
PD5 (OC1A)	19	14	I/O	4-я линия порта D (выход A таймера/счетчика TC1)
PD6 (ICP1)	20	15	I/O	6-я линия порта D (вход захвата таймера/счетчика TC1)
PD7 (OC2)	21	16	I/O	7-я линия порта D (выход таймера/счетчика TC2)

1.7 Среда разработки Atmel Studio

Средством разработки программного обеспечения для микроконтроллеров различных семейств являются интегрированные среды разработки (IDE, Integrated Development Environment). Среда разработки включает в себя основные компоненты:

- текстовый редактор кода;
- препроцессор, компилятор и/или интерпретатор;
- компоновщик;
- отладчик.

Для разработки и отладки программ для 8- и 32-разрядных микроконтроллеров семейства AVR и 32-разрядных микроконтроллеров семейства ARM компания Atmel предложила использовать Atmel Studio – бесплатную проприетарную интегрированную среду разработки, основанную на Visual Studio.

В ноябре 2020 г. компания Microchip выпустила новую версию среды разработки Atmel Studio под новым названием Microchip Studio for AVR and SAM Devices, которую можно скачать на сайте microchip.com. Для этого необходимо на главной веб-странице сайта последовательно пройти по следующим пунктам: **TOOLS AND RESOURCES → Develop → Microchip Studio for AVR® and SAM Devices**. Затем в самой нижней части загруженной веб-страницы во вкладке **Downloads** будут ссылки для скачивания автономного установщика и его веб-версии. Для скачивания предыдущих версий программы необходимо пройти по следующим пунктам: **TOOLS AND RESOURCES → Develop → AVR® and SAM MCU Downloads Archive**.

Описание минимального набора функций Atmel Studio, необходимых для выполнения лабораторных работ, представлено в пунктах 1.7.1–1.7.4.

1.7.1 Новый проект в Atmel Studio

После запуска среды разработки Atmel Studio для создания нового проекта можно использовать один из следующих способов:

1) на стартовой странице (**Start Page**), представленной на рисунке 1.7, выбрать пункт **New Project…**;

2) нажать кнопку с пиктограммой  на панели инструментов;

3) последовательно пройти по пунктам меню **File → New → Project...**.

Затем появится диалоговое окно мастера создания проектов, в котором необходимо задать параметры будущего проекта (рисунок 1.8):

1) выбрать язык программирования, например, **C/C++**;

2) выбрать тип проекта, например, **GCC C Executable Project** (исполняемый проект на языке C с поддержкой компилятором GCC);

3) указать название проекта в поле **Name**;

4) указать местоположение проекта в поле **Location**, используя кнопку **Browse…**;

5) указать название решения в поле **Solution name** либо оставить без изменения, тогда оно будет автоматически повторять название проекта;

6) необходимо убедиться, что установлен флажок напротив строки **Create directory for solution** (создать каталог для решения).

Далее после подтверждения в открывшемся окне **Device Selection** (рисунок 1.9) необходимо в выпадающем списке выбрать семейство, например, **ATmega**, и модель требуемого микроконтроллера, например, **ATmega16**. В правой части окна будет приведен список устройств, работающих с этим микроконтроллером, а также документация.

После подтверждения выбора микроконтроллера появится основное окно Atmel Studio (рисунок 1.10), в центральной части которого находится окно текстового редактора кода (вкладка **main.c**). В нем производится написание, редактирование и отладка программы. На начальном этапе в редакторе кода будет представлен листинг программы, построенный на основе шаблона для выбранного типа проектов. Каждая строка пронумерована. Эти номера установлены лишь для ссылки на используемые команды языка программирования в соответствующих строках программы.

Для включения нумерации строк кода необходимо пройти по пунктам меню **Tools → Options…**, затем в раскрывающемся списке **Text Editor** выбрать пункт **All Languages** и установить флажок на опции **Line numbers**. Также здесь можно установить опции **Word wrap** (переносить по словам) и **Show visual glyphs for word wrap** (показывать графические метки в местах переноса слов). Выставить подходящий тип и размер шрифта текста в окне редактора кода можно в этом же окне **Options**, выбрав в раскрывающемся списке **Environment** пункт **Font and Colors**.

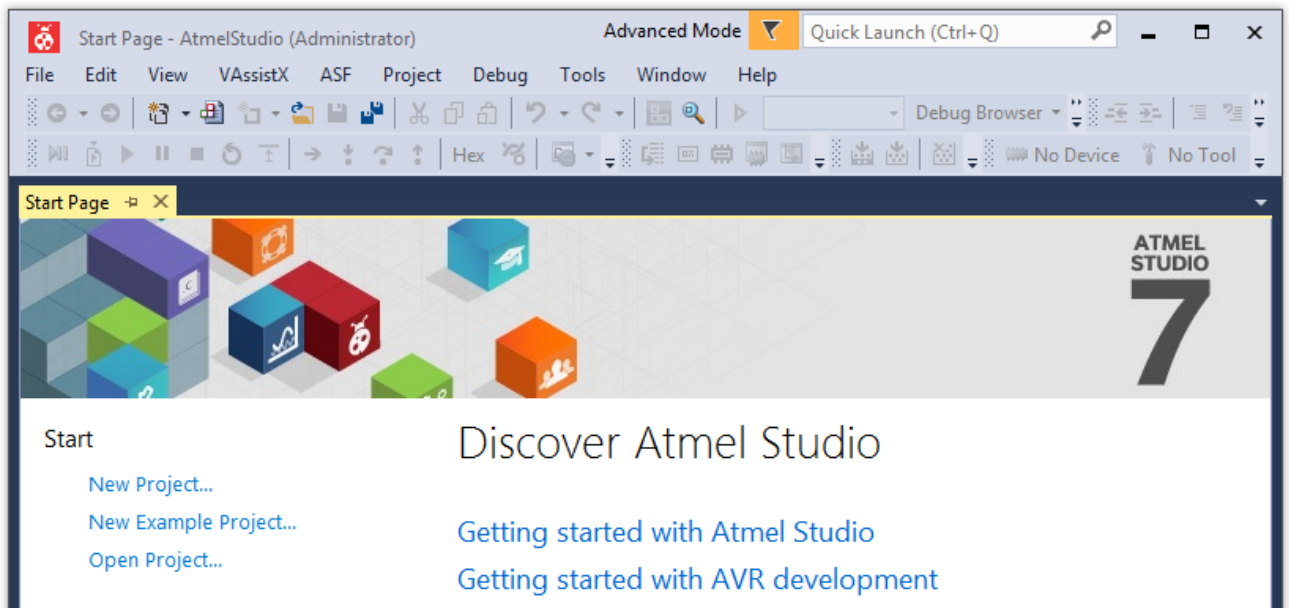


Рисунок 1.7 – Создание нового проекта

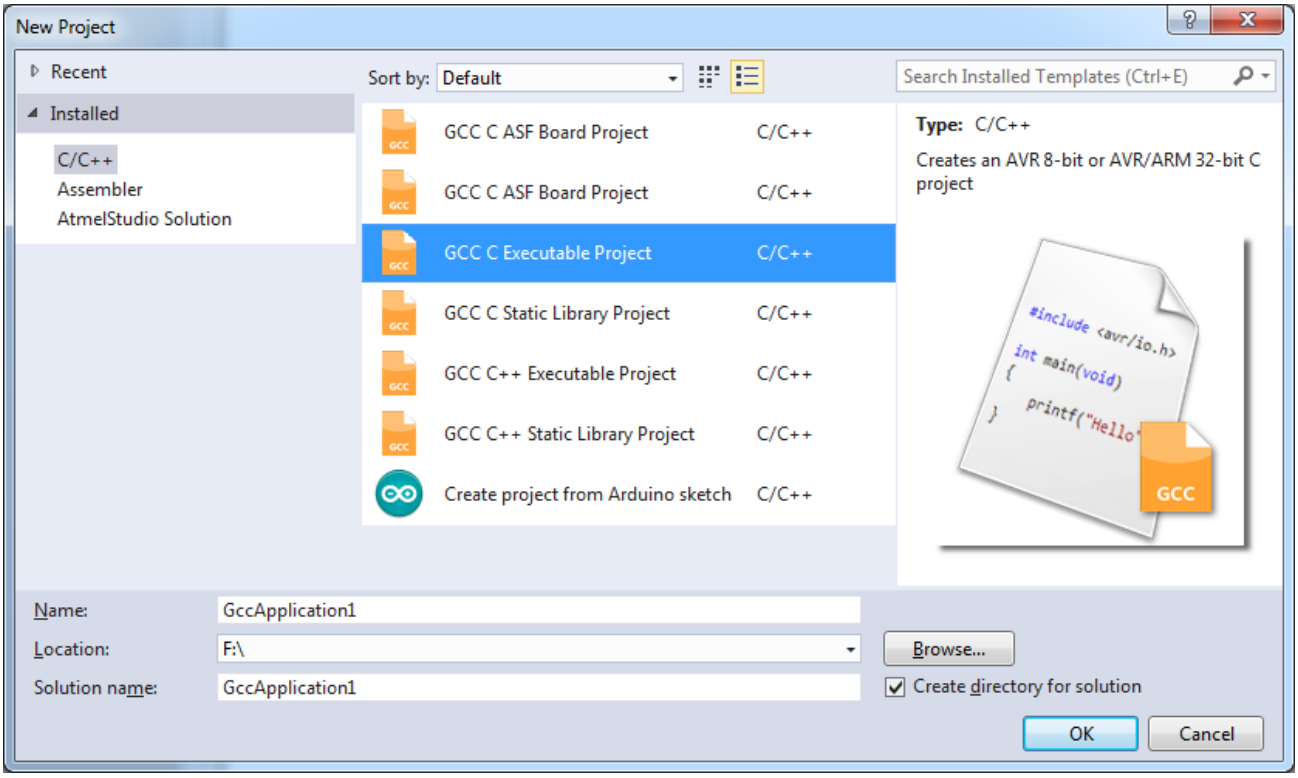


Рисунок 1.8 – Окно мастера создания проектов

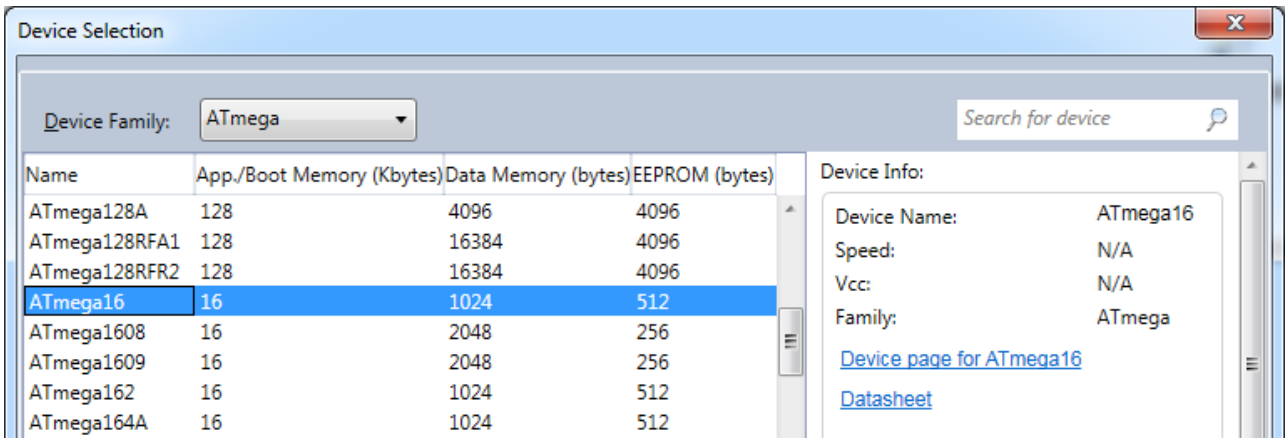


Рисунок 1.9 – Окно выбора модели микроконтроллера

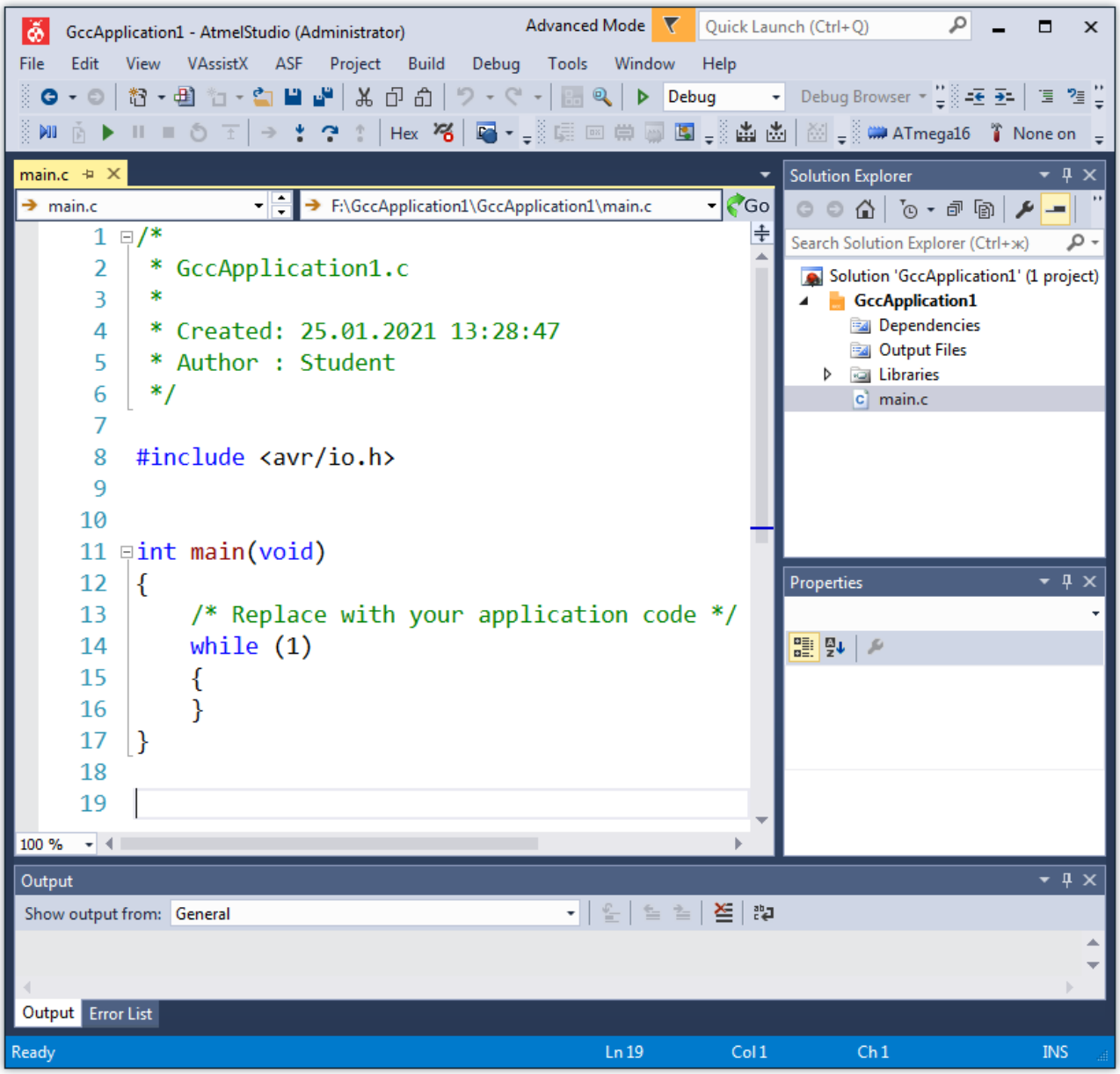


Рисунок 1.10 – Основное окно Atmel Studio

В соответствии с рисунком 1.10 листинг программы состоит из следующих элементов:

1 **В строках 1–6** находятся комментарии, которые содержат название созданного проекта, дату создания и имя автора (по имени учетной записи, в которой была запущена Atmel Studio). При компиляции комментарии игнорируются. Могут быть использованы два вида комментариев: однострочные, которые начинаются двойным слешем (//) и заканчиваются символом перехода на новую строку, и многострочные, которые начинаются открывающей парой символов – сочетанием слеша и звездочки (/*) и заканчиваются закрывающей парой символов */. Часто символы комментариев используются, когда нужно исключить часть кода из программы, не удаляя его при этом.

2 **В строке 8** находится директива препроцессора, которая подключает стандартный заголовочный файл *io.h*. При обработке этого файла препроцессор включает в программу заголовочный файл микроконтроллера, который был выбран в окне **Device Selection** (см. рисунок 1.9). Для каждой модели микроконтроллера есть свой заголовочный файл, например, для ATmega16 – *iom16.h*, для ATmega8 – *iom8.h*. Использование данных заголовочных файлов связано с тем, что в исходном тексте программы могут быть использованы имена регистров общего назначения и регистров ввода/вывода МК, имена отдельных разрядов этих регистров, адреса всех регистров МК, адреса и имена векторов прерываний и другие идентификаторы, необходимые для понимания компилятором GCC.

Угловые скобки указывают компилятору, что подключаемые файлы находятся в стандартном системном каталоге среды разработки с именем avr. Могут быть использованы двойные кавычки, которые будут указывать компилятору начинать поиск с директории, в которой хранятся файлы текущего проекта. Как правило, это папка проекта, в которой расположен файл с исходным текстом программы *main.c*.

Кроме директив препроцессора в данной части программы могут размещаться определение типов пользователя (typedef), объявление прототипов функций, определение глобальных переменных.

3 **В строках 11–17** находится главная (основная) функция. С нее начинается выполнение программы и ее назначение – управлять работой всей программы (проекта). В случае микроконтроллеров главная функция состоит из двух частей:

а) *однократно выполняемые команды*, например, настройка портов ввода/вывода, инициализация модулей встроенной периферии и внешних подключаемых устройств. На рисунке 1.10 – это строка 13, т. е. все команды между первой открывающей фигурной скобкой и оператором цикла ***while***;

б) *многократно (циклически) выполняемые команды*, например, вызов функций обработки данных, поступающих на или с устройств встроенной и внешней периферии. Данные команды размещаются между фигурными скобками (строки 15, 16) оператора цикла **while**, у которого логическое выражение всегда принимает значение единицы (истина). Этот бесконечный цикл является

обязательной составляющей главной функции на языке программирования C/C++, выполняемой микроконтроллером. Довольно часто этот цикл пустой и служит только для того, чтобы не дать выйти за пределы адресного пространства ПЗУ, в котором находится требуемый для исполнения машинный код управляющей программы.

Следует обратить внимание на один специфичный момент. Обычная программа для персонального компьютера чаще всего выполняется под управлением операционной системы (ОС), и потому функция *main()* завершается командой возврата *return 0;*, которая сообщает ОС об успешном выполнении программы. Однако в микроконтроллере нет операционной системы – с самого начала управление всеми ресурсами отдается единственной управляющей программе. Стандартное завершение не имеет смысла – возвращать код некуда, поэтому в конце главной функции отсутствует команда возврата. И, естественно, функция *main()* в программе для МК не может принимать никаких аргументов – им просто неоткуда взяться.

1.7.2 Панели инструментов Atmel Studio, отладка и сборка проекта

В основном окне Atmel Studio могут быть отображены параметры различных панелей инструментов (Toolbar Options). Основные панели инструментов представлены на рисунке 1.11 и заключаются в следующем:

1 – **Standard Toolbar Options** (стандартная панель инструментов) содержит некоторые инструменты пунктов меню File, Edit, View и Debug;

2 – **Debug Toolbar Options** (отладка) содержит главные инструменты отладки;

3 – **Atmel Debugger Toolbar Options** (отладчик Atmel) содержит инструменты для просмотра содержимого окон дизассемблера, регистров, памяти, процессора и окна ввода/вывода микроконтроллера;

4 – **Build Toolbar Options** (сборка) содержит инструменты для создания (сборки) решения и проекта;

5 – **Device and Debugger Toolbar Options** (устройство и отладчик) позволяет выбрать микроконтроллер и инструмент для отладки и прошивки;

6 – **Text Editor Toolbar Options** (текстовый редактор) содержит инструменты редактирования исходного текста программы.

Набор инструментов каждой панели может быть настроен пользователем.

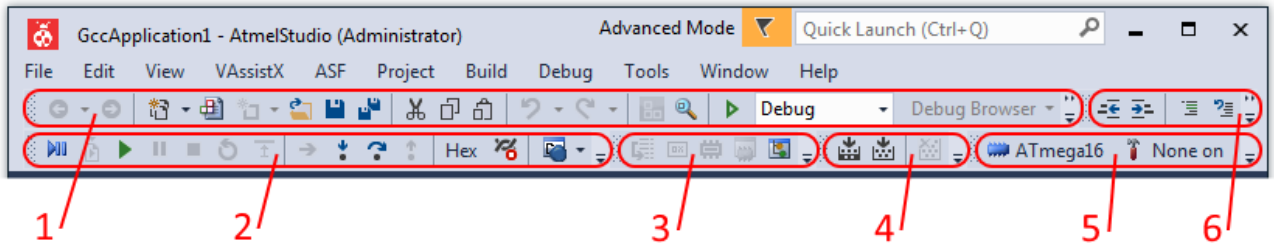


Рисунок 1.11 – Основные панели инструментов Atmel Studio

Отладку проекта необходимо начать с выбора инструмента для отладки (панель инструментов **Device and Debugger Toolbar Options**). По умолчанию выставлено значение **None on** или **No Tool**. Щелчок левой кнопки мыши на пиктограмме None on вызывает появление вкладки свойств проекта (рисунок 1.12), в выпадающем списке **Selected debugger/programmer** необходимо выбрать инструмент для отладки, например, **Simulator** – программный эмулятор отладки, встроенный в Atmel Studio.

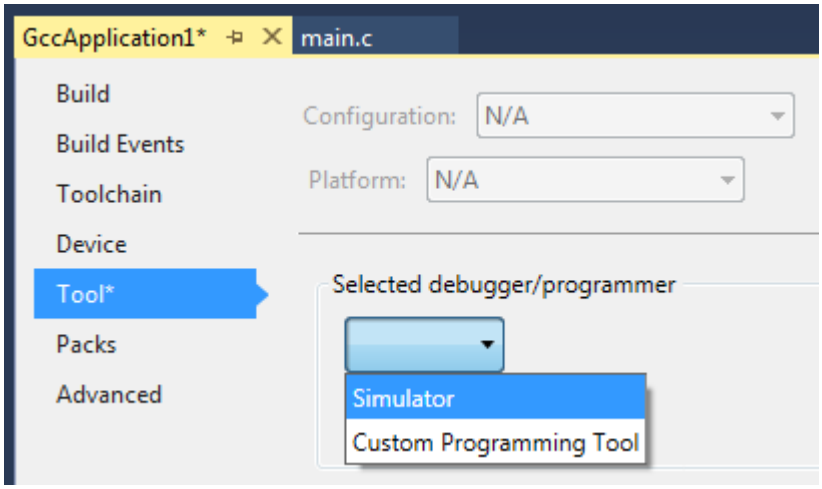


Рисунок 1.12 – Выбор отладчика

Для выбора другой модели микроконтроллера необходимо нажать пиктограмму с названием модели микроконтроллера **ATmega16**. Затем нажать **Change Device…** и выбрать новую модель микроконтроллера.

После того как инструмент для отладки был выбран, становится возможным начать процедуру отладки. Возможны два режима отладки:

1) пошаговый;

2) непрерывный.

Отличие этих режимов состоит в том, что в непрерывном режиме не отображаются текущие изменения содержимого окон отладчика Atmel. Непрерывный режим можно использовать, когда требуется выполнить какую-либо часть программного кода (без многократного нажатия кнопки выполнения одной команды **Step Into**) с остановкой на следующей ближайшей точке останова и переходом в пошаговый режим.

Для запуска отладки в пошаговом режиме необходимо либо на панели инструментов **Debug Toolbar Options** нажать кнопку **Start Debugging and Break** (пиктограмма), либо нажать кнопку **Step Into** (пиктограмма), либо в меню **Debug** выбрать пункт **Start Debugging and Break**. После запуска в окне текстового редактора кода появится указатель отладчика в виде желтой стрелки (рисунок 1.13), показывающий на отдельную строку в тексте программы, которая будет выполнена на следующем шаге, т. е. после нажатия кнопки **Step Into** или функциональной клавиши **F11**.

Для запуска отладки в непрерывном режиме необходимо либо на панели инструментов **Debug Toolbar Options** нажать кнопку **Start Debugging** (пиктограмма), либо в меню **Debug** выбрать пункт **Start Debugging**.

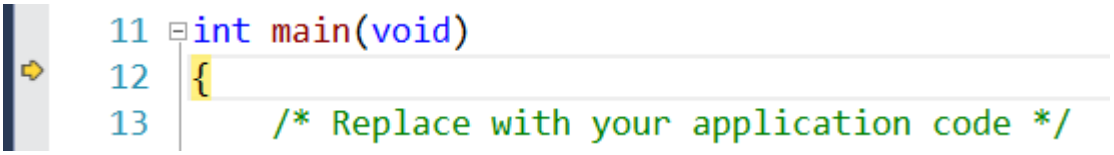


Рисунок 1.13 – Указатель отладчика в пошаговом режиме

В меню **Debug** находятся все инструменты стандартного отладчика и отладчика Atmel. Кроме вышерассмотренных инструментов в меню **Debug** в пошаговом режиме отладки (рисунок 1.14) также можно выделить следующие:

- **Windows** – содержит большинство инструментов, характерных для IDE Visual Studio, а также нескольких специальных инструментов отладчика Atmel;
- **Stop Debugging** – остановка и закрытие отладки, возвращение в режим разработки;
- **Start Without Debugging** – создает проект без запуска отладки;
- **Disable debugWire and Close** – опция доступна только при использовании интерфейса debugWire;
- **Continue** – продолжение отладки в непрерывном режиме;
- **Restart** – перезапуск отладки с перегрузкой исполняемой программы (после использования запускается *непрерывный режим отладки*);
- **Break All** – приостановка отладки в непрерывном режиме с переходом в *пошаговый*;
- **Quick Watch…** – быстрый расчет значения переменной, заранее выделенной или введенной в самом диалоговом окне **Quick Watch**, из которого переменную можно добавить в окно для последующего наблюдения **Watch 1**;
- **Step Into** (**F11**) – выполнение текущей строки исходного текста программы. Если встречается функция, то будет выполнен переход на ее первую строку;
- **Step Over** – выполнение текущей строки программы, не заходя в вызываемые функции, если они встречаются в тексте программы, при этом функция будет выполнена;
- **Step Out** – позволяет закончить выполнение текущей функции в непрерывном режиме отладки и сразу после завершения этой функции вернуться в пошаговый режим в основной программе;
- **Run To Cursor** – выполнение программы до текущей позиции курсора в непрерывном режиме с переходом в пошаговый режим на данной строке кода;
- **Reset** – перезапуск отладки с перегрузкой исполняемой программы (после использования запускается *пошаговый режим отладки*);
- **Toggle Breakpoint** – включает или выключает точку останова в месте расположения курсора;

- **New Breakpoint** – создает новую точку останова в месте расположения курсора;

- **Delete All Breakpoints** – очищает все точки останова, включая неактивные;

- **Clear All DataTips** – очищает все установленные подсказки по данным DataTips, которые позволяют получить информацию о значении переменной прямо в окне редактора кода. Для этого необходимо выделить переменную в коде программы и, щелкнув правой кнопкой мыши, выбрать пункт **Pin To Source**. После чего рядом с выделенной переменной появится небольшое окно, в котором будет прописано название переменной и ее текущее значение. В процессе выполнения программы в этом окне будет отображаться актуальное значение переменной;

- **Export DataTips…** – сохраняет подсказки по данным DataTips в XML-файл;

- **Import DataTips…** – загружает подсказки DataTips из XML-файла;

- **Options...** – задаются различные параметры среды Atmel Studio. Диалоговое окно **Options** можно также открыть, выбрав пункт меню **Tools → Options…**;

- **<*Имя_проекта*> Properties…** – задаются различные свойства текущего проекта. Вкладку свойств проекта можно также открыть, выбрав пункт меню **Project → <*Имя_проекта*> Properties…**.

Создание (сборка) решения или текущего проекта выполняется с помощью меню **Build**, показанного на рисунке 1.15 и состоящего из инструментов:

- **Build Solution/<*Имя_проекта*>** – выполняет сборку решения/проекта, в процессе которой компилируются только те файлы решения/проекта, в которых были изменения;

- **Rebuild Solution/<*Имя_проекта*>** – выполняет сборку решения/проекта на основании компиляции всех файлов решения/проекта;

- **Clean Solution/<*Имя_проекта*>** – все файлы, созданные в результате сборки, будут удалены;

- **BatchBuild...** – позволяет выполнить сборку для нескольких конфигураций;

- **Configuration Manager…** – позволяет создать конфигурации с различными параметрами для решения/проекта;

- **Compile** – выполняет компиляцию содержимого текущего окна.

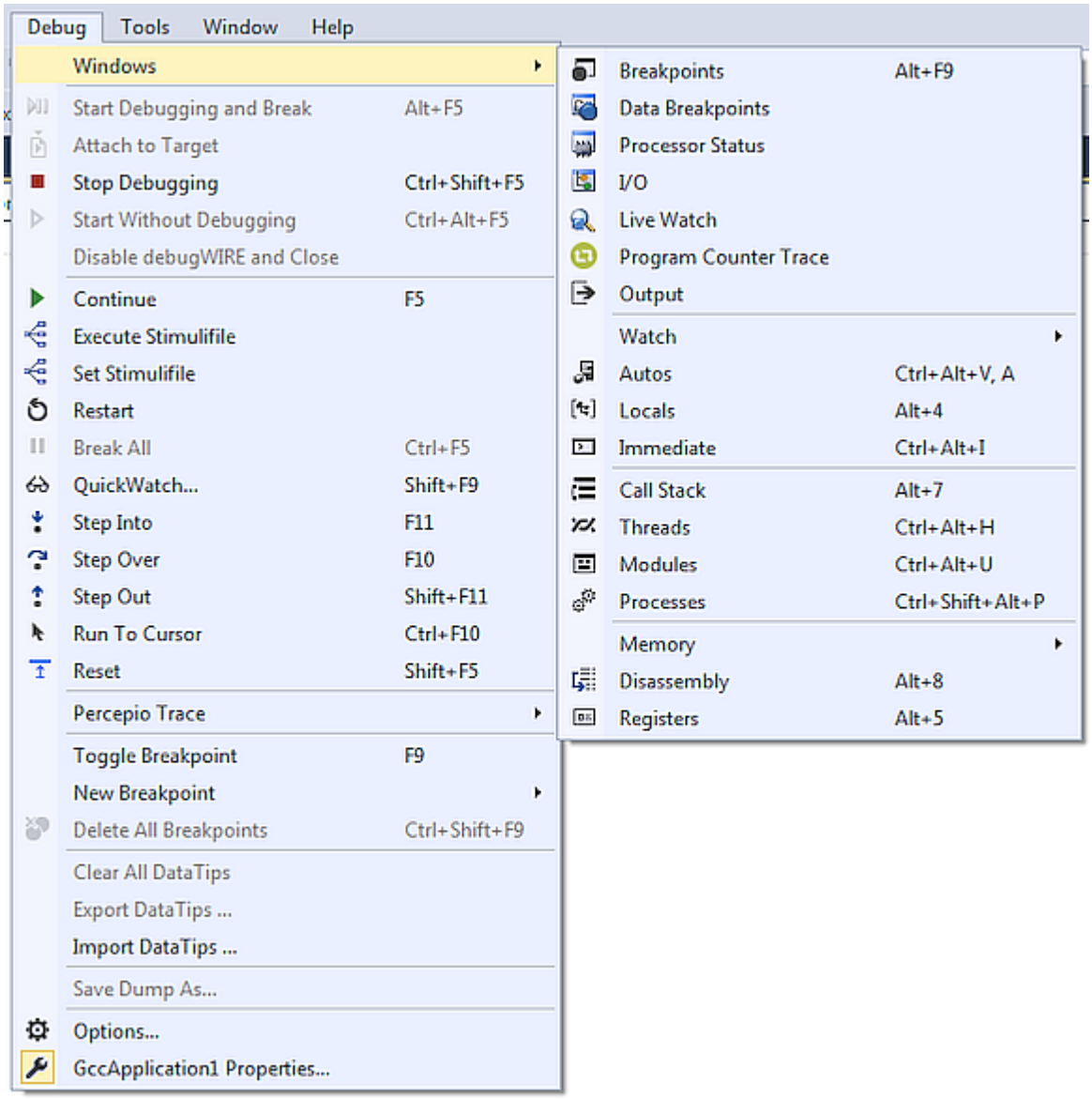


Рисунок 1.14 – Инструменты меню **Debug** в пошаговом режиме отладки

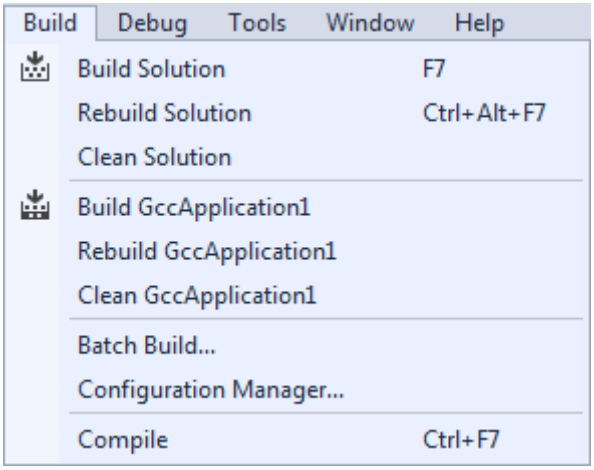


Рисунок 1.15 – Инструменты меню **Build**

Для микроконтроллеров сборка решения/проекта позволяет сформировать hex-файл или файл прошивки, который используется в дальнейшем для проверки работоспособности программы и конечного устройства с помощью виртуального микроконтроллера в программе Proteus ISIS или с помощью реального микроконтроллера в результате прошивки его flash-памяти программатором. Файл прошивки (*.hex) находится в папке проекта Atmel Studio в директории Debug.

Сборку решения проще выполнять, используя «горячую» клавишу. Чтобы задать клавишу, например, функциональную клавишу **F7** (см. рисунок 1.15), необходимо пройти по пунктам меню **Tools → Options…**. Затем в раскрывающемся списке **Environment** необходимо выбрать пункт **Keyboard**. В поле **Show commands containing** ввести название **Build.BuildSolution**, затем в поле **Press shortcut keys** нажать на клавиатуре **F7** и подтвердить выбор, нажав кнопку **Assign**.

1.7.3 Окна отладчика Atmel

К окнам отладчика Atmel относятся окно процессора **Processor Status**, окно регистров **Registers**, окно дампов памяти **Memory**, окно ввода/вывода **I/O**, окно дизассемблера **Disassembly**. Рассмотрим их подробнее.

Окно процессора **Processor Status** показывает основные регистры центрального процессора микроконтроллера (рисунок 1.16):

- **Program Counter** – программный счетчик (счетчик команд) содержит адрес ячейки flash-памяти МК, в которой находится машинный код следующей исполняемой команды программы;
- **Stack Pointer** – указатель стека содержит адрес вершины стека. Стек – это область ОЗУ, используемая исполняемой программой для хранения промежуточных результатов вычислений, адресов возврата из функций;
- **X Register**, **Y Register**, **Z Register** – индексные регистры используются для косвенной адресации при обращениях к ОЗУ и flash-памяти МК;
- **Status Register** – регистр состояния (статуса, флагов) содержит восемь флагов, характеризующих результат выполнения арифметических и логических команд, а также команд передачи управления и команд, работающих с разрядами регистров;
- **Cycle Counter** – счетчик тактов показывает количество периодов тактового сигнала, прошедших с начала отладки;
- **Frequency** – тактовая частота микроконтроллера;
- **Stop Watch** – время остановки показывает время, прошедшее с запуска отладки, и рассчитывается исходя из величины тактовой частоты и количества прошедших тактов;
- **Registers** – регистры общего назначения МК. Они дублируют содержимое отдельного окна регистров **Registers** (рисунок 1.17).

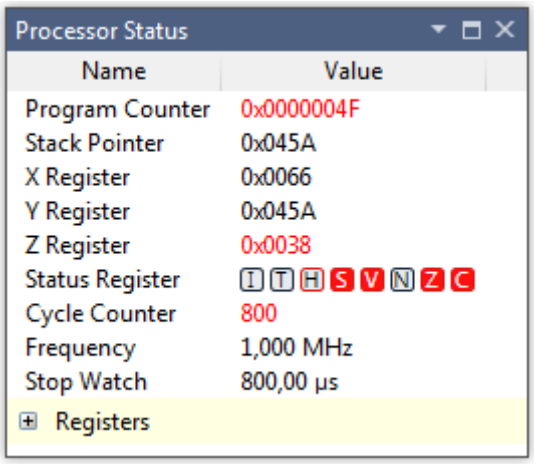


Рисунок 1.16 – Окно процессора **Processor Status**

Значения регистров окна процессора **Processor Status** могут отображаться в шестнадцатеричной, десятичной и двоичной системах счисления. Для того чтобы изменить систему, необходимо щелкнуть правой кнопкой мыши на значении и затем из контекстного меню выбрать **Display as Binary/Hexadecimal/Decimal**.

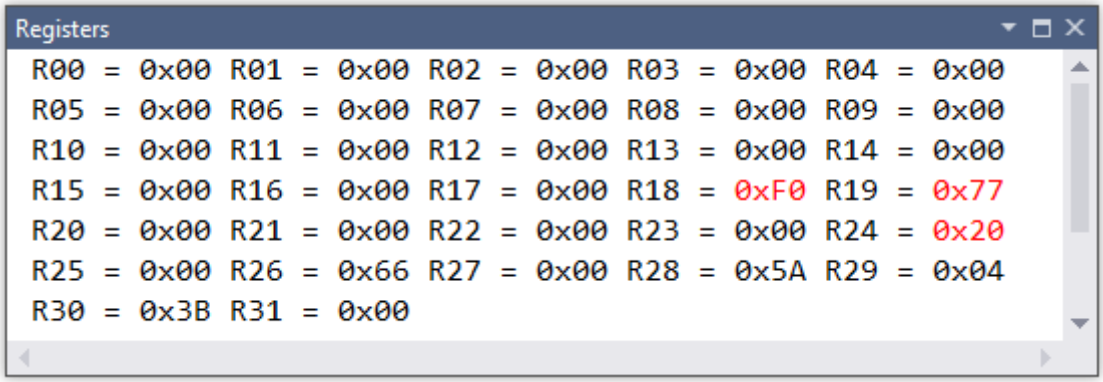


Рисунок 1.17 – Окно регистров **Registers**

В окне дампов памяти **Memory** (рисунок 1.18) отображается содержимое всех видов памяти микроконтроллера (flash, ОЗУ, ЭСППЗУ, регистры). Числовые значения с префиксом `prog` показывают адрес первой ячейки памяти в этой же строке. Адрес каждой следующей ячейки памяти в строке находится прибавлением единицы к адресу предыдущей ячейки, например, на рисунке 1.18 выделенная ячейка памяти будет иметь адрес 0x007B (в шестнадцатеричной системе счисления).

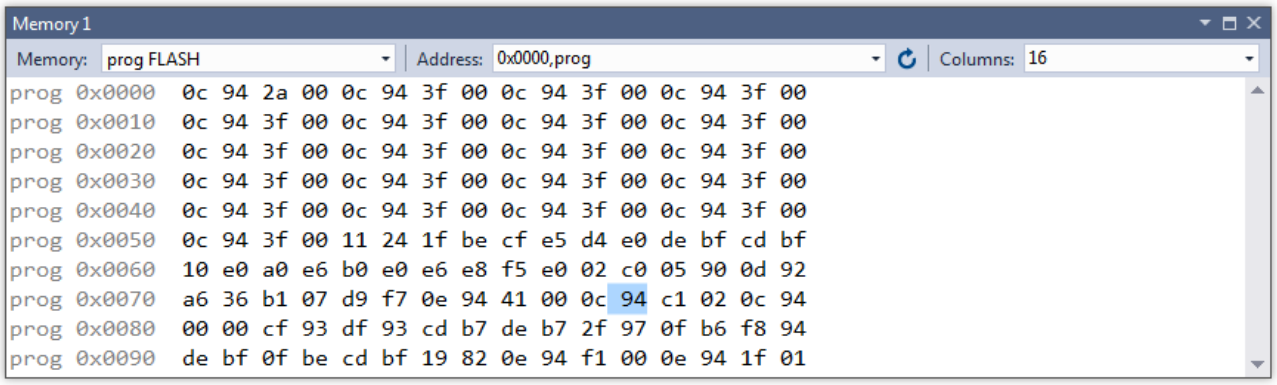


Рисунок 1.18 – Окно дампов памяти **Memory** (flash-память МК)

В поле **Address** можно указать адрес конкретной ячейки памяти для просмотра. В поле **Columns** можно настроить количество столбцов ячеек памяти в окне.

Окно ввода/вывода **I/O** показывает содержимое портов ввода/вывода и устройств встроенной периферии МК (рисунок 1.19).

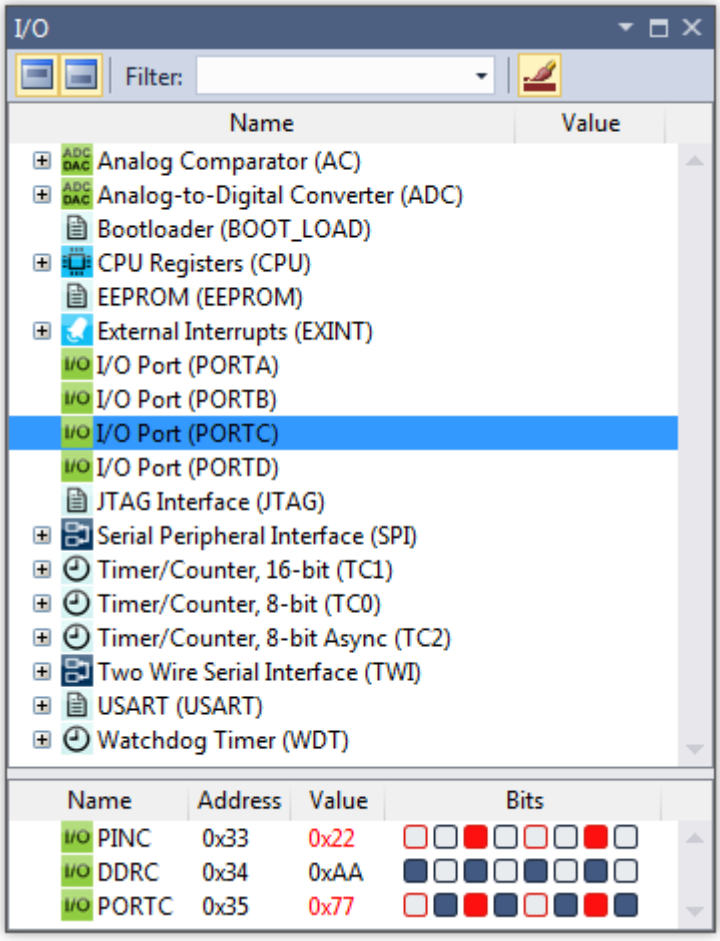


Рисунок 1.19 – Окно ввода/вывода **I/O**

В пошаговом режиме отладки значения регистров (отдельных разрядов регистров) окон процессора и ввода/вывода могут быть изменены вручную путем щелчка левой кнопки мыши на значении регистра (разряда). Некоторые разряды не могут быть изменены, они находятся в режиме «только чтение», некоторые разряды могут быть в режиме «только запись». Когда разряд установлен, он немедленно считывается отладчиком, таким образом, отображаются актуальные значения. Разряд, установленный в лог. 1, отображается как черный квадрат, установленный в лог. 0, – как белый квадрат. Когда регистр меняет свое начальное значение, оно отображается красным цветом. Если происходит переход разряда регистра из лог. 0 в лог. 1, то цвет квадрата меняется с белого на красный. Если происходит переход из лог. 1 в лог. 0, то цвет квадрата меняется с черного на белый с красной рамкой.

В окне дизассемблера **Disassembly** выводится дизассемблированный код программы, т. е. восстановленный до команд AVR-ассемблера (рисунок 1.20). При этом в окне дизассемблера можно выбрать отображаемую информацию (**Viewing Options**): показывать машинный код для каждой команды ассемблера; показывать операторы C/C++, что позволяет увидеть, какими ассемблерными командами реализован тот или иной оператор (зависит от уровня оптимизации); показывать номера строк кода; показывать адреса flash-памяти МК, по которым размещаются машинные коды команд.

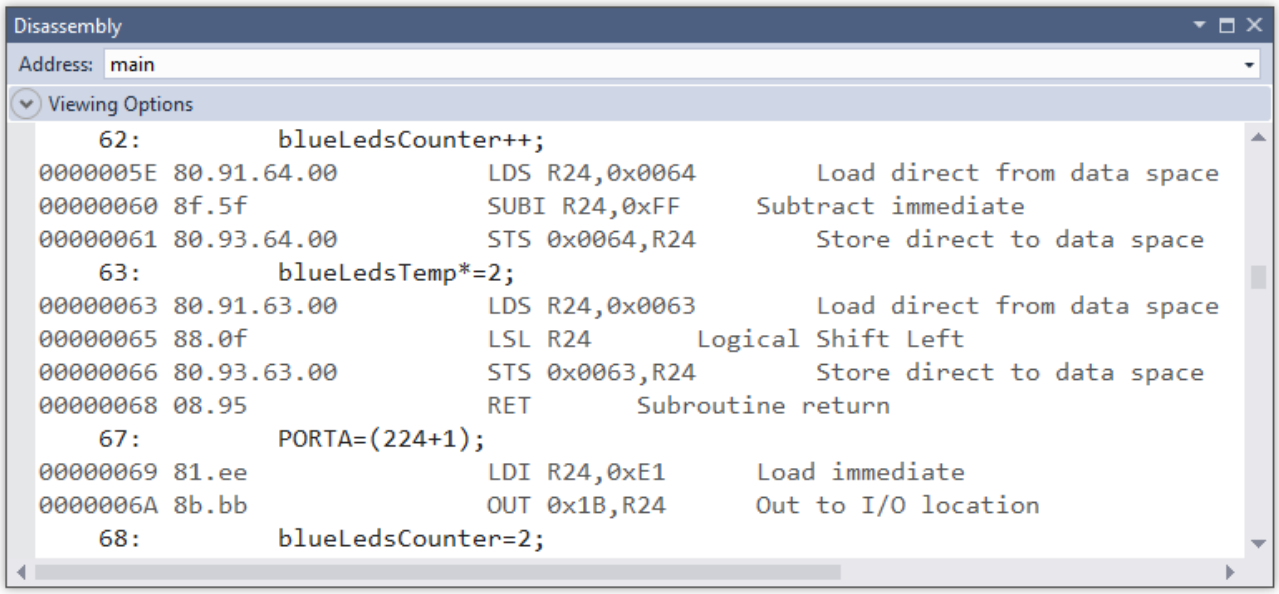


Рисунок 1.20 – Окно дизассемблера **Disassembly**

1.7.4 Вспомогательные отладочные окна

Рассмотрим некоторые вспомогательные отладочные окна: окно точек останова **Breakpoints** и окно списка ошибок **Error List**.

Точка останова является пометкой, которую можно установить в то место исходного текста программы, где непрерывный режим отладки должен быть

приостановлен. Установив несколько точек останова, можно следить за ходом выполнения определенных частей программы. Возможно создание условных точек останова. Эти точки приостанавливают отладку, только если выполнено некоторое условие, например, переменная имеет определенное значение. Установить точку останова можно, щелкнув левой кнопкой мыши на серой области справа от кода. Все функции, связанные с точками останова, доступны с помощью контекстного меню, которое появляется при щелчке правой кнопкой мыши на маркере точки останова (рисунок 1.21).

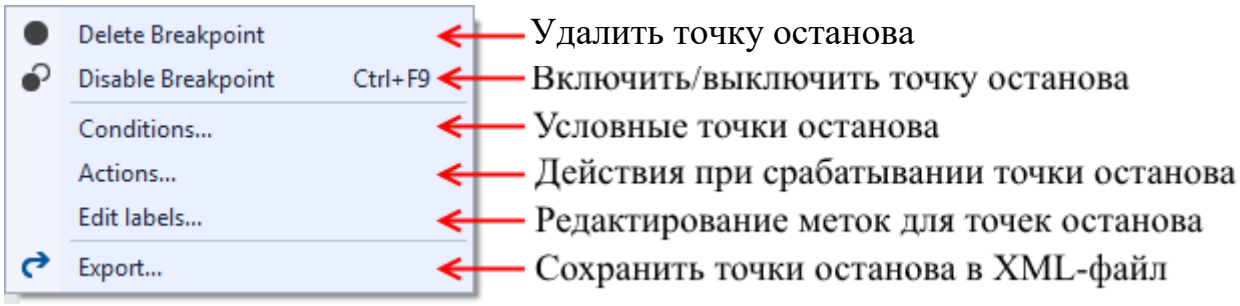


Рисунок 1.21 – Управление точками останова

Для создания условной точки останова необходимо задать условие (пункт **Conditions…**). Условие может быть переменной, например «i», или выражением, например «i > 30» (рисунок 1.22). При выбранной опции **Is true** точка останова вступит в силу, если условие верно (для вышеприведенного примера, если «i» стало больше 30). Если выбрана опция **When changed**, точка останова вступит в силу при условии, что значение выражения изменилось с момента последнего прохождения точки останова. Опция **Hit Count** позволяет приостановить выполнение тогда, когда данная строка кода выполнится определенное количество раз. Пункт **Actions…** позволяет настроить вывод сообщения в окне **Output** при срабатывании точки останова.

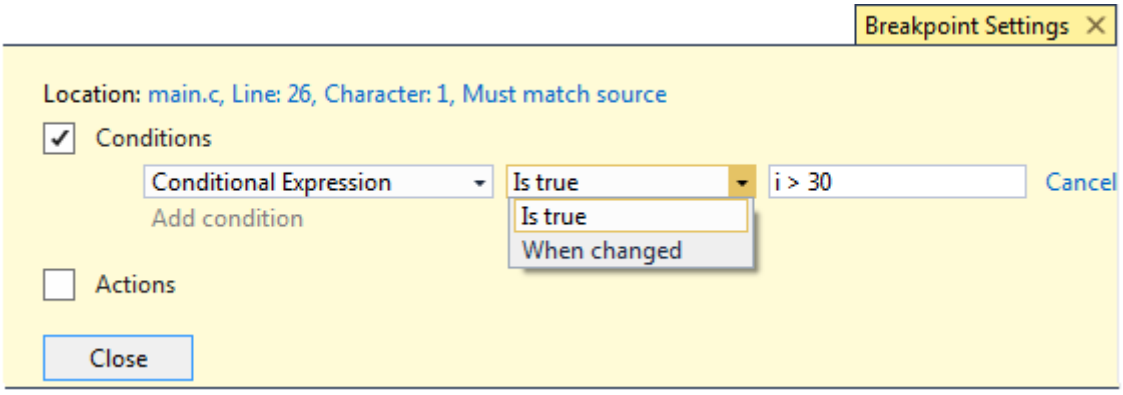


Рисунок 1.22 – Настройки условных точек останова

Информация обо всех точках останова, используемых в программе, и их статусе отображается в окне точек останова **Breakpoints** (рисунок 1.23).

Если в процессе компиляции обнаруживается ошибка в коде, выводится соответствующее сообщение в списке ошибок **Error List** (рисунок 1.24). Если дважды щелкнуть на строке, содержащей описание ошибки, будет показана строка исходного текста программы, в которой эта ошибка обнаружена. Часто единственная ошибка приводит к появлению множества сообщений. Компилятор генерирует несколько сообщений, чтобы дать больше информации о проблемах, которые он обнаружил.

Рисунок 1.23 – Окно точек останова **Breakpoints**

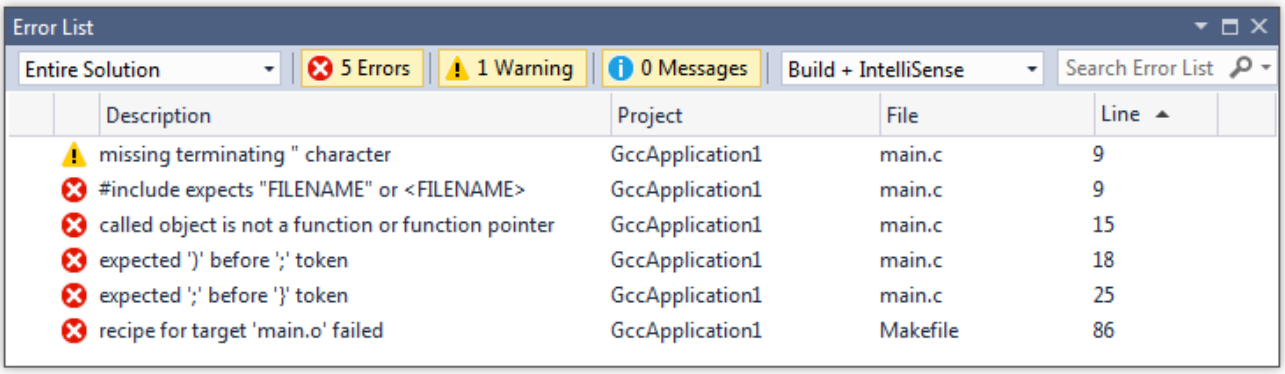


Рисунок 1.24 – Окно списка ошибок **Error List**

1.8 Система автоматизированного проектирования электронных схем Proteus VSM

Моделирование работы различных устройств на основе микроконтроллера ATmega16 и программного обеспечения, написанного в среде Atmel Studio, может проводиться с помощью системы Proteus Virtual System Modelling (Proteus VSM).

Proteus VSM представляет собой коммерческий пакет программ класса САПР, разработанный английской компанией Labcenter Electronics, и состоит из программы синтеза, моделирования и отладки электронных схем в режиме реального времени (Proteus VSM Simulator (ISIS)) и программы разработки печатных плат (Proteus PCB Design (ARES)). В лабораторных работах для провер-

ки работоспособности программ и схем на основе микроконтроллера ATmega16 будет использована только программа Proteus ISIS.

Последнюю версию системы проектирования можно приобрести на официальном сайте компании labcenter.com. Разработчик предлагает возможность скачать бесплатную демонстрационную версию. Она обладает всеми функциями и возможностями платного пакета, использование программы не ограничено по времени. Однако позволяет проводить моделирование только для встроенных примеров демо-проектов.

Основное окно программы Proteus ISIS версии 7.10 представлено на рисунке 1.25.

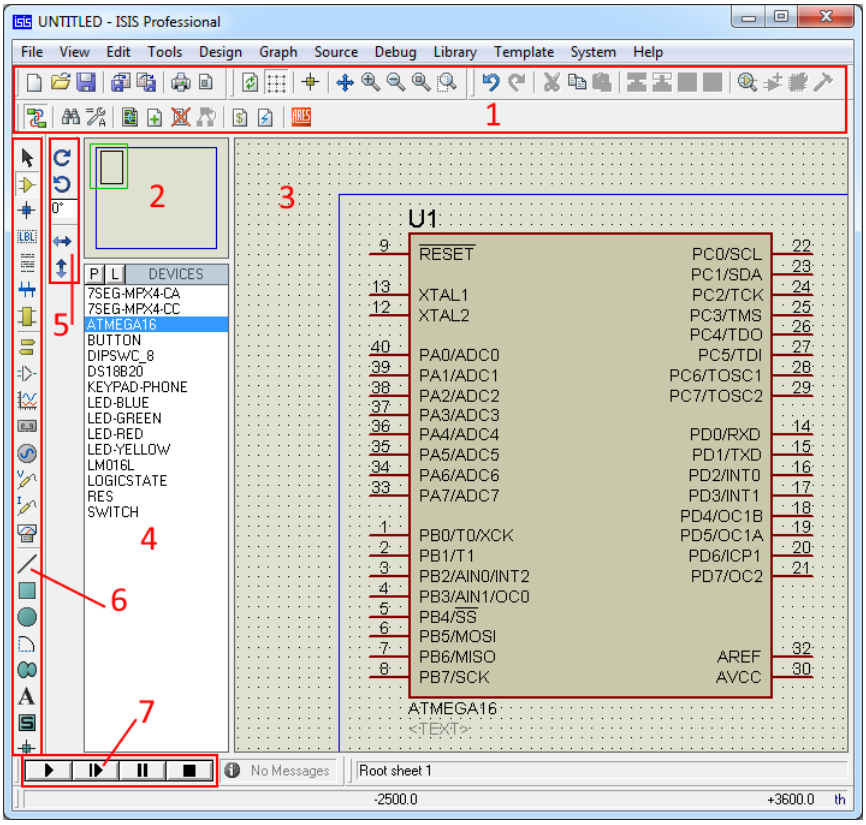


Рисунок 1.25 – Основное окно программы Proteus ISIS

Основное окно программы Proteus ISIS состоит из следующих элементов:

1 – панель базовых и специальных инструментов;

2 – окно обзора, которое позволяет оперативно перемещаться по схеме проекта, предоставляет краткий обзор схемы (отображается только область, обозначенная синим прямоугольником; зеленый прямоугольник обозначает видимую область окна редактирования схемы), а также позволяет просмотреть выбранный компонент;

3 – окно редактирования схемы, предназначенное для размещения и соединения компонентов схемы;

4 – окно выбора объектов, которое содержит список компонентов, необходимых для размещения в окне редактирования (в частности, оно запоминает

использованные компоненты из библиотеки (кнопка **P** (**Pick new object(s) from the libraries**)), чтобы в дальнейшем их можно было быстро применить);

5 – панель ориентации компонентов схемы в окне редактирования;

6 – панель выбора режимов редактора (разделена на три части: основные режимы, приспособления, 2D-графика);

7 – панель управления моделированием, на которой размещено четыре кнопки: **Play** (пуск), **Step** (выполнение одного шага моделирования), **Pause** (пауза) и **Stop** (стоп).

Чтобы открыть проект в программе Proteus ISIS, необходимо в меню **File** выбрать пункт **Open Design** (открыть проект) либо использовать кнопку с аналогичным названием на панели инструментов и проследовать по требуемому пути. Для выполнения лабораторных работ были заранее созданы схемы. Необходимо выбрать схему в соответствии с выполняемой работой.

Для загрузки в модель микроконтроллера управляющей программы (предварительно созданной и отлаженной в Atmel Studio) необходимо открыть окно редактирования свойств микроконтроллера (рисунок 1.26) двойным щелчком левой кнопки мыши на микроконтроллере.

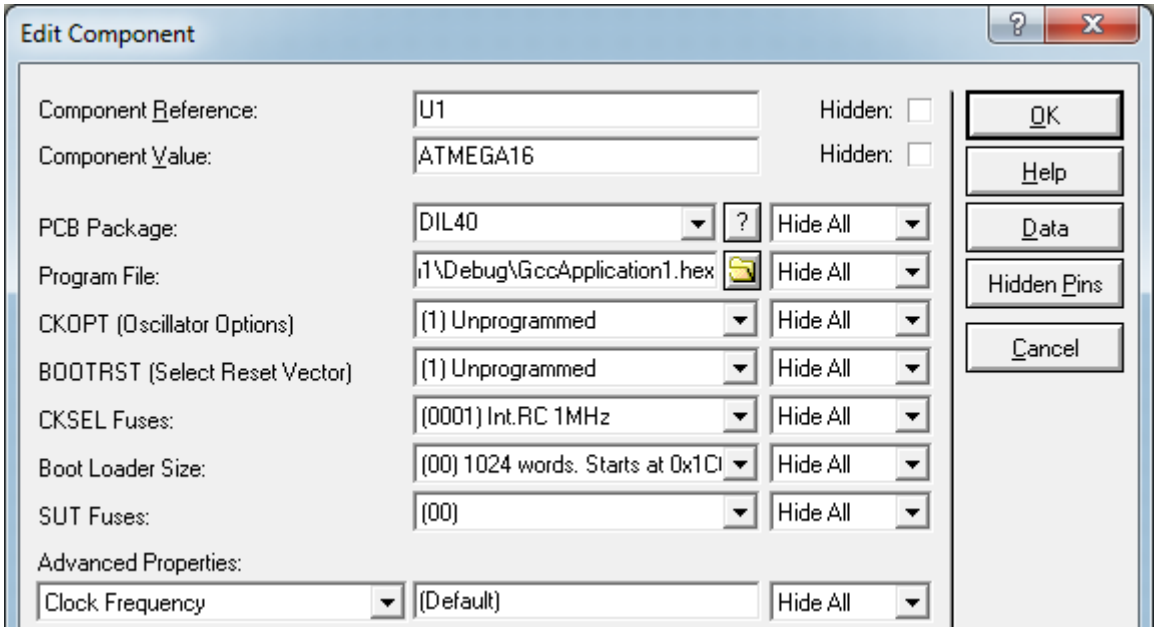


Рисунок 1.26 – Окно редактирования свойств микроконтроллера

В поле **Program File** (файл программы) необходимо указать путь к файлу прошивки (*.hex), который находится в папке проекта Atmel Studio в директории Debug.

В поле **CKSEL Fuses** (фьюзы выбора источника тактирования) из выпадающего списка выбирается тип и частота источника тактового сигнала, который будет использован для работы микроконтроллера. Если необходимо использовать нерегламентированную частоту работы, то вначале в списке выбирается внешний тактовый сигнал **(0000) Ext.Clock**, затем в списке расширенных

свойств **Advanced Properties** выбирается пункт тактовой частоты **Clock Frequency** и в поле вместо значения по умолчанию **Default** указывают требуемое значение частоты в герцах.

2 Порядок выполнения работы

Задание 1

1.1 Запустите среду разработки Atmel Studio.

1.2 Создайте проект в соответствии с инструкциями, представленными в пункте 1.7.1 теоретических сведений. Модель микроконтроллера – ATmega16. В качестве названия проекта укажите имя, состоящее из фамилий студентов бригады на английском языке и номера лабораторной работы. Для сохранения проекта создайте папку с номером учебной группы на указанном преподавателем локальном диске компьютера.

1.3 В окне текстового редактора кода полностью удалите листинг программы, созданный по умолчанию, и затем наберите исходный текст программы из листинга, представленного в приложении А, кроме номеров строк, которые необходимы только для ссылки на соответствующие строки программы.

1.4 Выберите пункт меню **Project**, затем **<*Имя_проекта*> Properties…**. Откроется вкладка свойств проекта. Перейдите к разделу **Toolchain**, затем в раскрывающемся списке **AVR/GNU C Compiler** выберите **Optimazation**, затем в выпадающем списке уровня оптимизации **Optimazation Level** выберите **None (-O0)**.

Компилятор GCC для AVR предоставляет несколько уровней оптимизации: None (-O0), Optimize (-O1), Optimize more (-O2), Optimize most (-O3) и Optimize for size (-Os). На каждом уровне предоставляются разные опции оптимизации, за исключением уровня -O0, который означает отсутствие оптимизации.

Отключение оптимизации -O0 обычно используется для отладки по исходному тексту программы, чтобы можно было нормально оценить алгоритм работы программы и просмотреть значение переменных на каждом шаге.

1.5 Выполните отладку в пошаговом и непрерывном режимах. Предварительно необходимо ознакомиться с информацией, представленной в пунктах 1.7.2 и 1.7.3.

Перед запуском отладчика **обязательно** выделите строку кода 27 (функция *_delay_ms(500)*) и закомментируйте ее, используя кнопку **Comment out the selected lines** (пиктограмма ) на панели инструментов.

Обратите внимание на доступность содержимого окон, которые появляются после запуска отладки в пошаговом и непрерывном режимах, а именно окно дампов памяти **Memory** (память **prog FLASH** и память **data IRAM**), окно регистров **Registers**, окно процессора **Processor Status**, окно ввода/вывода **I/O**.

Для выполнения команд программы в пошаговом режиме нажмите клавишу **F11** или кнопку **Step Into** (пиктограмма ) на панели инструментов.

1.5.1 При выполнении отладки в пошаговом режиме, **когда желтая стрелка отладчика будет указывать**:

- на строку 17: откройте дамп flash-памяти в окне **Memory** (в выпадающем списке окна должен быть выбран пункт **prog FLASH**). Затем нажмите кнопку **Memory Toolbar Options** (пиктограмма). В строке **Columns** из списка выберите 8 (отобразить ячейки памяти в восемь столбцов). Затем сделайте скриншот этого окна для отчета. В строках 19 и 20 отладчик выполнит вызов функций инициализации (начальной настройки) портов ввода/вывода и таймера/счетчика TC0 микроконтроллера, которые будут рассмотрены в следующих лабораторных работах;
- на строку 42: выберите в окне ввода/вывода **I/O** строку с названием I/O Port (PORTA). Обратите внимание, как изменяются разряды регистров PINA, DDRA, PORTA при выполнении команд в строках 42, 43;
- на строку 44: выберите в окне ввода/вывода **I/O** строку с названием I/O Port (PORTB). Обратите внимание, как изменяются разряды регистров PINB, DDRB, PORTB при выполнении команд в строках 44, 45;
- на строку 46: выберите в окне ввода/вывода **I/O** строку с названием I/O Port (PORTC). Обратите внимание, как изменяются разряды регистров PINC, DDRC, PORTC при выполнении команд в строках 46, 47;
- на строку 48: выберите в окне ввода/вывода **I/O** строку с названием I/O Port (PORTD). Обратите внимание, как изменяются разряды регистров PIND, DDRD, PORTD при выполнении команд в строках 48, 49;
- на строку 53: выберите в окне ввода/вывода **I/O** строку с названием Timer/Counter, 8-bit (TC0). Обратите внимание, как изменяются разряды регистров этого 8-разрядного таймера/счетчика при выполнении команд в строках 53–55.

В строках 23–26 отладчик выполнит вызов функций, которые в зависимости от своих команд будут изменять значения в соответствующих регистрах портов A, B, C и D. Используя пошаговый режим отладки, определите, с какими портами ввода/вывода связана каждая из функций и какие значения будут записаны в регистры.

Закомментированную строку 27 отладчик пропустит, после чего проверит логическое условие в двух операторах выбора ***if***. И в зависимости от их истинности выполнит соответствующие команды. Далее в соответствии с бесконечным циклом оператора ***while*** выполнение команд начнется снова со строки 23.

1.5.2 Выйдите из режима пошаговой отладки, нажав кнопку **Stop Debugging** (пиктограмма). Перед выполнением отладки в непрерывном режиме необходимо установить точки останова на шесть вызываемых функций в главной функции (рисунок 1.27). Затем запустите отладку в непрерывном режиме (кнопка **Start Debugging** (пиктограмма) или клавиша **F5**). Обратите внимание, что при каждом повторном нажатии этой кнопки будет полностью выполняться соответствующая функция с остановкой на следующей точке останова и переходом в пошаговый режим отладки. Так после выполнения первой вызываемой функции *ports_init()* будут «сразу» установлены начальные

значения регистров PIN, DDR, PORT всех четырех портов ввода/вывода в окне ввода/вывода **I/O**. Пройдите через каждую точку останова по несколько раз, наблюдая за изменениями в соответствующих регистрах портов A, B, C и D.

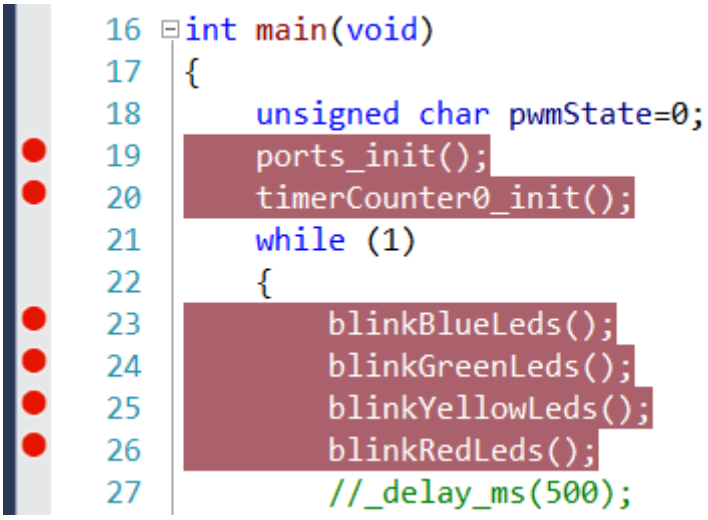


Рисунок 1.27 – Расположение точек останова в тексте программы

1.5.3 Выйдите из режима отладки. Выключите первые пять точек останова с помощью опции **Disable Breakpoint**. Затем задайте в настройках шестой точки останова условие срабатывания **redLedsCounter == value**, в котором вместо value нужно подставить значение в соответствии с заданным вариантом в таблице 1.5 по примеру рисунка 1.28. Запустите отладку в непрерывном режиме. После срабатывания этой условной точки останова сделайте скриншоты следующих окон для отчета:

- окно ввода/вывода **I/O** сделайте таким образом, чтобы были видны значения в регистрах PIND, DDRD, PORTD;
- окно процессора **Processor Status** без списка регистров R00–R31;
- окно регистров **Registers**;
- окно дампа памяти **Memory**, а именно память **data IRAM**.

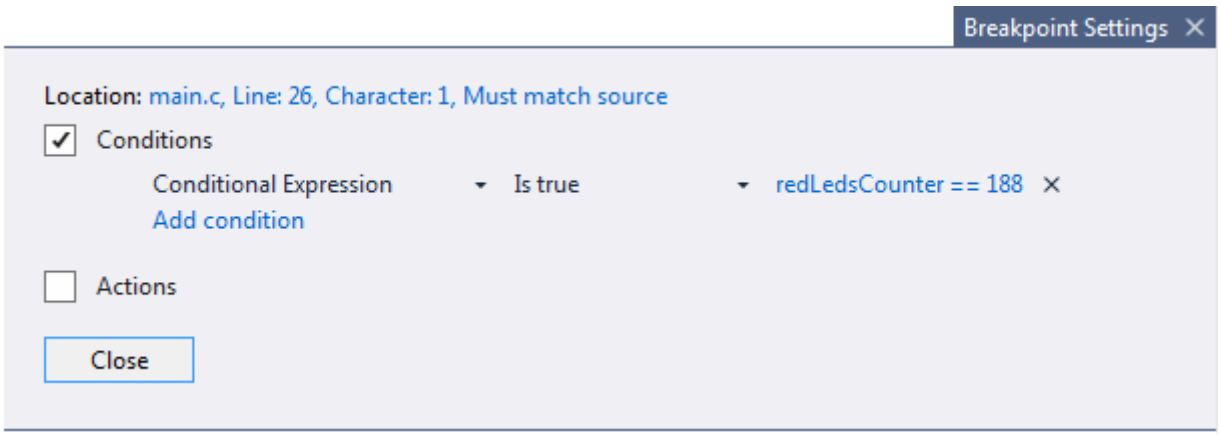


Рисунок 1.28 – Настройки условной точки останова для строки 26 программы

Таблица 1.5 – Исходные данные для выполнения пункта 1.5.3 задания

Номер варианта	Значение value для условия
1	18
2	35
3	52
4	69
5	86
6	103
7	120
8	137
9	154
10	171

Задание 2

2.1 Выйдите из режима отладки. Выделите строку кода 27 (функция *_delay_ms(500)*) и раскомментируйте ее, используя кнопку **Uncomment the selected lines** (пиктограмма) на панели инструментов.

Включите первый уровень оптимизации Optimize (-O1).

Выполните сборку решения, нажав функциональную клавишу **F7**. В результате последние изменения в коде программы и настройки будут учтены, и сформируется конечный файл прошивки.

2.2 Запустите программу Proteus ISIS. Откройте исследуемую схему устройства, которая находится в указанной преподавателем директории компьютера и показана в приложении Б. Затем сохраните эту схему в вашу папку проекта Atmel Studio с таким же названием, как и у вашего проекта Atmel Studio.

2.3 Откройте окно редактирования свойств микроконтроллера ATmega16 (Edit Component) двойным щелчком левой кнопки мыши на микроконтроллере. Затем в поле **Program File** (файл программы) укажите путь к вашему файлу прошивки (*.hex), который находится в папке вашего проекта Atmel Studio в директории Debug.

В поле **CKSEL Fuses** (фьюзы выбора источника тактирования) выберите в качестве источника внутренний калиброванный RC-генератор с тактовой частотой 1 МГц, т. е. необходимо из выпадающего списка выбрать строку **(0001) Int.RC 1MHz**.

Затем запустите моделирование, нажав кнопку **Play** (**Run the simulation**) в нижнем левом углу окна программы. Обратите внимание на выполнение программы микроконтроллером. По замыканию ключа SA1 запускается формирование ШИМ-сигнала со скважностью 3. Размыкание ключа останавливает формирование сигнала. Наблюдать ШИМ-сигнал можно в окне осциллографа OSC1 программы Proteus ISIS. Чтобы открыть это окно, необходимо после запуска моделирования в меню **Debug** в конце списка выбрать пункт **Digital Oscilloscope - OSC1**.

3 Содержание отчета

3.1 Титульный лист.

3.2 Цель работы.

3.3 Исследуемая схема устройства в Proteus ISIS (см. приложение Б).

3.4 Исходные данные для выполнения пункта 1.5.3 задания.

3.5 Листинг программы, функционирование которой проверялось в программе Proteus ISIS.

3.6 Скриншоты окон отладчика Atmel в соответствии с заданием 1.

4 Контрольные вопросы

4.1 Как осуществляется преобразование числа из десятичной системы счисления в двоичную и шестнадцатеричную системы и наоборот?

4.2 Из каких функциональных блоков состоит микропроцессорная система, охарактеризуйте их?

4.3 Что такое шина? Какие шины нижнего уровня известны?

4.4 Что собой представляет однокристальный микроконтроллер?

4.5 Каков модульный принцип построения микроконтроллеров?

4.6 Каковы архитектуры построения центрального процессора микропроцессорной системы?

4.7 Какие архитектуры построения памяти микропроцессорной системы известны и чем они отличаются друг от друга?

4.8 Что понимается под технологией конвейеризации?

4.9 К какому семейству относится изучаемый микроконтроллер? Расшифруйте и переведите название семейства.

4.10 Какая модель микроконтроллера используется в лабораторной работе? Поясните маркировку. Какую разрядность имеет данный микроконтроллер?

4.11 Каковы основные технические характеристики изучаемого микроконтроллера?

4.12 Каков состав встроенных периферийных устройств изучаемого микроконтроллера?

4.13 Какими предельно допустимыми параметрами обладает изучаемый микроконтроллер?

4.14 Расшифруйте и переведите аббревиатуру IDE. Какие основные компоненты входят в ее состав? Какая IDE используется в лабораторной работе?

4.15 Опишите основные элементы шаблонного листинга программы, возникающего в окне текстового редактора кода используемой среды разработки сразу же после создания проекта на языке программирования C/C++.

4.16 Для чего необходим отладчик? Как называется встроенный инструмент отладки и какие режимы отладки имеются в используемой среде разработки? Как эти режимы влияют на доступность отладочных окон?

4.17 Перечислите окна отладчика Atmel. Какая информация представлена в каждом из этих окон?

4.18 Для чего используются точки останова и каким образом их можно установить? В чем заключаются условные точки останова?

4.19 Какая программа синтеза, моделирования и отладки электронных схем в режиме реального времени используется в лабораторной работе?

ЛАБОРАТОРНАЯ РАБОТА №2

ИЗУЧЕНИЕ ПОРТОВ ВВОДА/ВЫВОДА МИКРОКОНТРОЛЛЕРА ATMEGA16

Цель работы: получить навыки программного конфигурирования и управления портами ввода/вывода микроконтроллера ATmega16 при использовании простейших внешних подключаемых устройств ввода (тактовые кнопки) и устройств вывода (светодиоды).

1 Теоретические сведения

Микроконтроллер ATmega16 является 8-разрядным, относится к семейству AVR, подсемейству mega, изготовлен по малопотребляющей КМОП-технологии, строится по гарвардской и RISC архитектурам, обладает технологией конвейеризации.

1.1 Организация памяти микроконтроллера ATmega16

В соответствии с гарвардской архитектурой память МК ATmega16 разделена на две физически непересекающиеся области – память программ (Program Memory) и память данных (Data Memory). Каждая память обладает своим адресным пространством, своими способами адресации и шинами доступа. Такая структура позволяет центральному процессору работать одновременно как с памятью программ, так и с памятью данных, что существенно увеличивает производительность. Карта памяти МК ATmega16 представлена на рисунке 2.1, где префикс 0x означает шестнадцатеричную систему счисления.

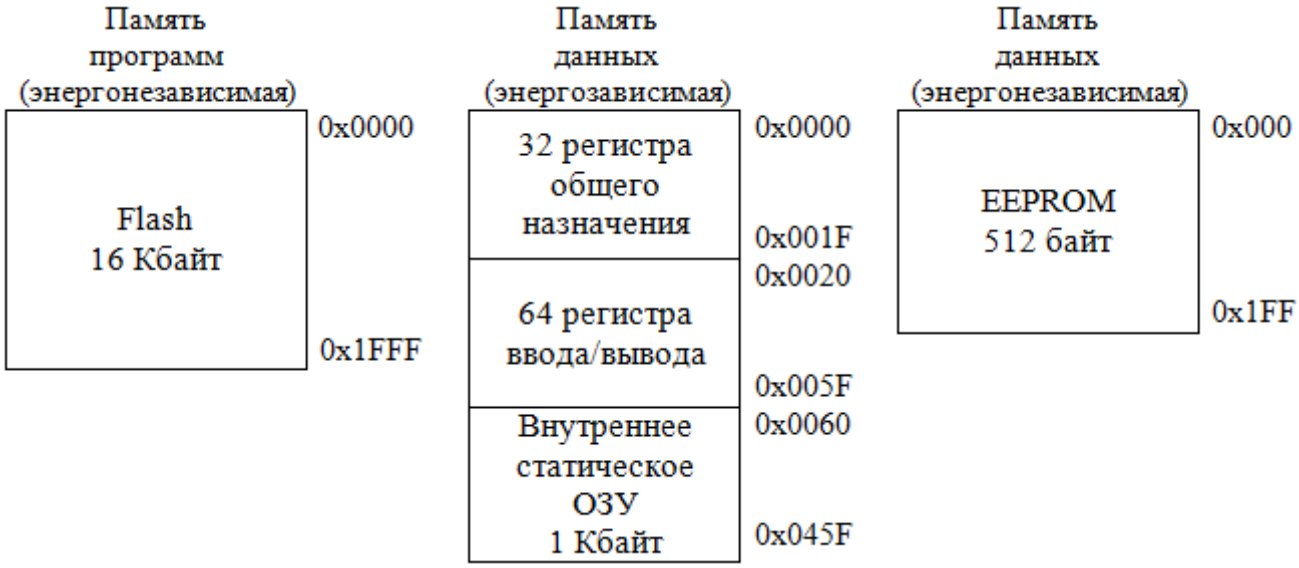


Рисунок 2.1 – Карта памяти микроконтроллера ATmega16

Память программ предназначена для хранения команд, управляющих работой микроконтроллера, и таблиц констант, не меняющихся во время работы программы. Физическим носителем памяти программ является энергонезависимое постоянное запоминающее устройство (Read Only Memory (ROM)), которое выполнено по flash-технологии. Память программ размещается только в кристалле.

Объем памяти МК ATmega16 составляет 8×1024 16-разрядных слов. Данную запись можно пояснить следующим образом. Каждая команда программы, которую выполняет центральный процессор МК, кодируется некоторым числом, называемым машинным кодом. В зависимости от используемого способа адресации команды микроконтроллеров семейства AVR представляются двумя либо четырьмя байтами, при этом каждый байт размещается в одной ячейке памяти программ. Для уменьшения времени обращения и считывания таких команд используется один адрес для каждой пары байтов, т. е. используется так называемая адресация по словам (т. к. одно слово соответствует 16 разрядам или двум байтам). Именно поэтому последняя ячейка памяти программ на рисунке 2.1 имеет адрес в шестнадцатеричной системе счисления 0x1FFF или в десятичной – 8191, а с учетом нулевой ячейки памяти получится 8192 адреса, каждый из которых используется для обращения к отдельной выполняемой 2-байтовой команде программы. Если команда 4-байтовая, то для хранения кода команды будут отведены четыре ячейки памяти, а для обращения к ним будут использованы два адреса. Следует также отметить, что т. к. каждый байт команды и данных размещается в отдельной ячейке памяти программ, то шина адреса, которая используются для обращения к памяти программ, является 14-разрядной ($2^{14} = 16384$).

Память программ МК AVR логически разделена на две части: область прикладной программы (Application Section) и область загрузчика (Boot Loader Section). В последней может располагаться специальная программа (загрузчик), позволяющая микроконтроллеру самостоятельно управлять загрузкой и выгрузкой прикладных программ, используя любой доступный интерфейс передачи данных (USART, SPI, TWI). Загрузчик может модифицировать, удалять и собственный код. Если же возможность самопрограммирования у микроконтроллера не используется, прикладная программа может располагаться и в области загрузчика. Запись машинных кодов команд в память программ осуществляется при помощи специальных устройств – программаторов.

Всегда после подачи напряжения питания на МК или сигнала сброса в процессе работы МК выполнение команд начинается с адреса 0x0000 памяти программ и последовательно продолжается в направлении последнего адреса 0x1FFF. Как правило, начиная с адреса 0x0002, в памяти программ располагается таблица векторов прерываний, которые представляют собой 4-байтовые команды абсолютного перехода к процедурам обработки прерываний и будут подробнее рассмотрены в следующих лабораторных работах. Чтобы исключить непреднамеренное выполнение векторов прерываний, по нулевому адресу (0x0000) памяти программ всегда должна находиться 4-байтовая команда пере-

хода (вектор сброса) к инициализационной части исполняемой прикладной программы. Однако положение таблицы векторов прерываний может быть изменено. Они могут размещаться не только в начале памяти программ, но и в области загрузчика. В этом случае прикладная программа может начинаться с нулевой ячейки памяти.

Память данных МК ATmega16 предназначена для хранения промежуточных результатов выполняемых команд, использует байтовую адресацию (т. е. один адрес соответствует одной ячейке памяти объемом в один байт), имеет 11-разрядную шину адреса (адрес последней ячейки памяти 0x045F или 0b10001011111, т. е. используется не более 11 разрядов) и представлена двумя типами: энергозависимая и энергонезависимая памяти (см. рисунок 2.1).

Энергозависимая память данных разделена на две части:

1) регистровая память;

2) оперативное запоминающее устройство (Random Access Memory (RAM)).

Регистровая память включает 32 регистра общего назначения (РОН) и 64 регистра ввода/вывода (РВВ). Все регистры являются 8-разрядными.

Регистры общего назначения объединены в так называемый регистровый файл. Каждый РОН имеет свое имя: R0–R31. Все машинные арифметико-логические команды микроконтроллеров семейства AVR, а также некоторые команды передачи управления и работы с разрядами регистров построены таким образом, что обязательно используют РОН. Команды в качестве операндов используют либо содержимое двух разных РОН, либо содержимое РОН и константу, либо содержимое одного РОН. Результат вычислений также помещается в один из РОН. Регистры общего назначения используются также в командах передачи данных. Копировать данные можно из одного РОН в другой, из РОН в ячейку памяти МК и в обратном направлении. Копирование данных возможно также между РОН и регистрами ввода/вывода. Регистры общего назначения делятся на три группы:

1) младшие регистры R0–R15, с которыми не могут работать некоторые машинные команды, например, непосредственная запись константы в регистр;

2) старшие регистры R16–R31 – полноценные регистры, работающие со всеми командами без исключения;

3) индексные регистры R26–R31.

Шесть последних регистров из старшей группы используются как индексные регистры (регистры-указатели) для косвенной адресации при обращениях к ОЗУ и flash-памяти МК. Они попарно объединяются, образуя 16-разрядные регистры: X (R26:R27), Y (R28:R29) и Z (R30:R31). При косвенной адресации обращение направлено к ячейке памяти (чтение/запись), адрес которой должен находиться в одном из этих регистров. При этом для работы с ОЗУ может использоваться любой из регистров-указателей, а для работы с flash-памятью – только регистр Z. Также стоит отметить: т. к. адрес большинства ячеек памяти МК является 16-разрядным и соответственно состоит из младшего и старшего байтов, то в соответствующих регистровых парах реги-

стры R26, R28 и R30 используются для хранения младшего байта адреса, а регистры R27, R29 и R31 – для хранения старшего байта.

Регистры ввода/вывода являются регистрами специальных функций (Special Function Registers (SFR)), которые можно разделить на две группы:

1) служебные регистры микроконтроллера;

2) регистры, относящиеся к конкретным устройствам встроенной периферии.

Список регистров ввода/вывода микроконтроллера ATmega16 представлен в таблице 2.1, где знак $ означает шестнадцатеричную систему счисления.

Для обращения к РВВ могут быть использованы как адреса общего адресного пространства энергозависимой памяти данных ($20–$5F), так и адреса своего адресного пространства ($00–$3F). Разница заключается в том, что чтение/запись РВВ при использовании своего адресного пространства осуществляется с помощью 2-байтовых машинных команд, которые к тому же выполняются за один период тактового сигнала, а при использовании общего адресного пространства памяти данных чтение/запись РВВ будет уже осуществляться с помощью 4-байтовых машинных команд, которые выполняются за два периода тактового сигнала и используются, главным образом, при обращении к ОЗУ.

Оперативное запоминающее устройство МК ATmega16 имеет объем 1 Кбайт и используется для хранения значений переменных в том случае, когда для выполнения прикладной программы не хватает регистров общего назначения. ОЗУ является внутренней памятью МК ATmega16, который в отличие от других моделей семейства AVR не имеет возможности подключения внешней памяти в качестве ОЗУ.

ОЗУ представляет собой статическую память (СОЗУ, Static RAM (SRAM)), преимущество использования которой заключается в том, что позволяет поддерживать данные без частой регенерации, необходимой в динамической памяти (Dynamic RAM (DRAM)). По этой причине статическая память может работать на очень низких частотах тактирования с полной остановкой синхроимпульсов, что позволяет МК ATmega16 использовать тактовые сигналы с низкой частотой до 0 Гц. Однако чтение/запись данных остается возможным только пока есть питание, поэтому СОЗУ является энергозависимым типом памяти.

Для микроконтроллеров семейства AVR, имеющих внутреннюю оперативную память, характерно наличие стека, который используется при выполнении большинства прикладных программ. **Стек** представляет собой область ОЗУ, используемую исполняемой программой для хранения промежуточных результатов вычислений, адресов возврата из функций. В МК ATmega16 стек реализован программно. Он начинается от максимального адреса ОЗУ (0x45F) и взаимодействие с ним осуществляется по принципу LIFO (Last In – First Out, «последним пришел – первым вышел»). По мере заполнения стека его граница смещается к младшим адресам ОЗУ микроконтроллера, т. е. стек растет вверх ОЗУ, а адрес в указателе стека (Stack Pointer (SP)) уменьшается. Предельное минимальное значение адреса в указателе стека для МК ATmega16 составляет 0x60,

поскольку дальше начинается область регистров специальных функций. Для инициализации стека используется пара регистров ввода/вывода SPH и SPL (см. таблицу 2.1).

В качестве **энергонезависимой памяти данных** в МК ATmega16 выступает электрически стираемое перепрограммируемое ПЗУ (ЭСППЗУ, Electrically Erasable Programmable Read-Only Memory (EEPROM)) объемом 512 байт. Память EEPROM используют для долговременного хранения различной информации, которая может изменяться в процессе функционирования готового устройства (калибровочные константы, серийные номера, ключи). Эта память находится в отдельном адресном пространстве, а доступ к ней (чтение/запись) осуществляется с помощью определенных регистров ввода/вывода (см. таблицу 2.1), поэтому память можно отнести и к устройствам встроенной периферии. Цикл записи в ЭСППЗУ длится миллисекунды.

При написании прикладной программы на языке C/C++ имена регистров ввода/вывода необходимо использовать в исходном тексте для конфигурирования параметров МК и используемых устройств встроенной периферии. Чтение/запись РВВ будет выполняться с использованием оператора присваивания «=». Аналогично и для ЭСППЗУ, в котором данные и адреса используемых ячеек памяти необходимо записывать в соответствующие РВВ, отведенные для работы с ЭСППЗУ. Регистры общего назначения непосредственно не используются в тексте программы на языке C/C++ (исключением являются ассемблерные вставки). Оптимизатор компилятора на основании алгоритма (зависит от выбранного уровня оптимизации) самостоятельно выбирает, какие регистры общего назначения и ячейки памяти СОЗУ выделить для использования в машинных командах, которые соответствуют заменяемым операторам C/C++ в процессе компиляции в объектный код, но без нарушения рассмотренных выше принципов работы с РОН и стеком.

Таблица 2.1 – Регистры ввода/вывода микроконтроллера ATmega16

Название регистра	Адрес	Функция
1	2	3
SREG	$3F ($5F)	Регистр состояния (флагов)
SPH	$3E ($5E)	Указатель стека, старший байт
SPL	$3D ($5D)	Указатель стека, младший байт
OCR0	$3C ($5C)	Регистр сравнения таймера/счетчика TC0
GICR	$3B ($5B)	Общий регистр управления прерываниями
GIFR	$3A ($5A)	Регистр флагов внешних прерываний
TIMSK	$39 ($59)	Регистр масок прерываний таймеров/счетчиков
TIFR	$38 ($58)	Регистр флагов прерываний таймеров/счетчиков
SPMCR	$37 ($57)	Регистр управления записью в память программ
TWCR	$36 ($56)	Регистр управления интерфейса SPI
MCUCR	$35 ($55)	Общий регистр управления микроконтроллера
MCUCSR	$34 ($54)	Регистр управления и состояния микроконтроллера

Продолжение таблицы 2.1

1	2	3
TCCR0	$33 ($53)	Регистр управления таймера/счетчика TC0
TCNT0	$32 ($52)	Счетный регистр таймера/счетчика TC0
OSCCAL	$31 ($51)	Регистр калибровки тактового генератора
OCDR		Регистр внутрисхемной отладки
SFIOR	$30 ($50)	Регистр специальных функций ввода/вывода
TCCR1A	$2F ($4F)	Регистр управления A таймера/счетчика TC1
TCCR1B	$2E ($4E)	Регистр управления B таймера/счетчика TC1
TCNT1H	$2D ($4D)	Счетный регистр таймера/счетчика TC1, старший байт
TCNT1L	$2C ($4C)	Счетный регистр таймера/счетчика TC1, младший байт
OCR1AH	$2B ($4B)	Регистр сравнения A таймера/счетчика TC1, старший байт
OCR1AL	$2A ($4A)	Регистр сравнения A таймера/счетчика TC1, младший байт
OCR1BH	$29 ($49)	Регистр сравнения B таймера/счетчика TC1, старший байт
OCR1BL	$28 ($48)	Регистр сравнения B таймера/счетчика TC1, младший байт
ICR1H	$27 ($47)	Регистр захвата таймера/счетчика TC1, старший байт
ICR1L	$26 ($46)	Регистр захвата таймера/счетчика TC1, младший байт
TCCR2	$25 ($45)	Регистр управления таймера/счетчика TC2
TCNT2	$24 ($44)	Счетный регистр таймера/счетчика TC2
OCR2	$23 ($43)	Регистр сравнения таймера/счетчика TC2
ASSR	$22 ($42)	Регистр состояния асинхронного режима таймера/счетчика TC2
WDTCR	$21 ($41)	Регистр управления сторожевого таймера
UBRRH	$20 ($40)	Регистр скорости обмена интерфейса USART, старший байт
UCSRC		Регистр управления и состояния C интерфейса USART
EEARH	$1F ($3F)	Регистр адреса ЭСППЗУ (EEPROM), старший байт
EARL	$1E ($3E)	Регистр адреса ЭСППЗУ (EEPROM), младший байт
EEDR	$1D ($3D)	Регистр данных ЭСППЗУ (EEPROM)
EECR	$1C ($3C)	Регистр управления ЭСППЗУ (EEPROM)
PORTA	$1B ($3B)	Регистр вывода данных через порт A
DDRA	$1A ($3A)	Регистр направления передаваемых данных через порт A
PINA	$19 ($39)	Регистр ввода данных через порт A
PORTB	$18 ($38)	Регистр вывода данных через порт B
DDRB	$17 ($37)	Регистр направления передаваемых данных через порт B
PINB	$16 ($36)	Регистр ввода данных через порт B
PORTC	$15 ($35)	Регистр вывода данных через порт C
DDRC	$14 ($34)	Регистр направления передаваемых данных через порт C
PINC	$13 ($33)	Регистр ввода данных через порт C
PORTD	$12 ($32)	Регистр вывода данных через порт D
DDRD	$11 ($31)	Регистр направления передаваемых данных через порт D
PIND	$10 ($30)	Регистр ввода данных через порт D
SPDR	$0F ($2F)	Регистр данных интерфейса SPI
SPSR	$0E ($2E)	Регистр состояния интерфейса SPI
SPCR	$0D ($2D)	Регистр управления интерфейса SPI
UDR	$0C ($2C)	Регистр данных интерфейса USART
UCSRA	$0B ($2B)	Регистр управления и состояния A интерфейса USART
UCSRB	$0A ($2A)	Регистр управления и состояния B интерфейса USART
UBRRL	$09 ($29)	Регистр скорости обмена интерфейса USART, младший байт
ACSR	$08 ($28)	Регистр управления и состояния аналогового компаратора
ADMUX	$07 ($27)	Регистр управления мультиплексором АЦП

Продолжение таблицы 2.1

1	2	3
ADCSRA	$06 ($26)	Регистр управления и состояния А АЦП
ADCH	$05 ($25)	Регистр данных АЦП, старший байт
ADCL	$04 ($24)	Регистр данных АЦП, младший байт
TWDR	$03 ($23)	Регистр данных интерфейса TWI
TWAR	$02 ($22)	Регистр адреса интерфейса TWI
TWSR	$01 ($21)	Регистр состояния интерфейса TWI
TWBR	$00 ($20)	Регистр скорости обмена данными через интерфейс TWI

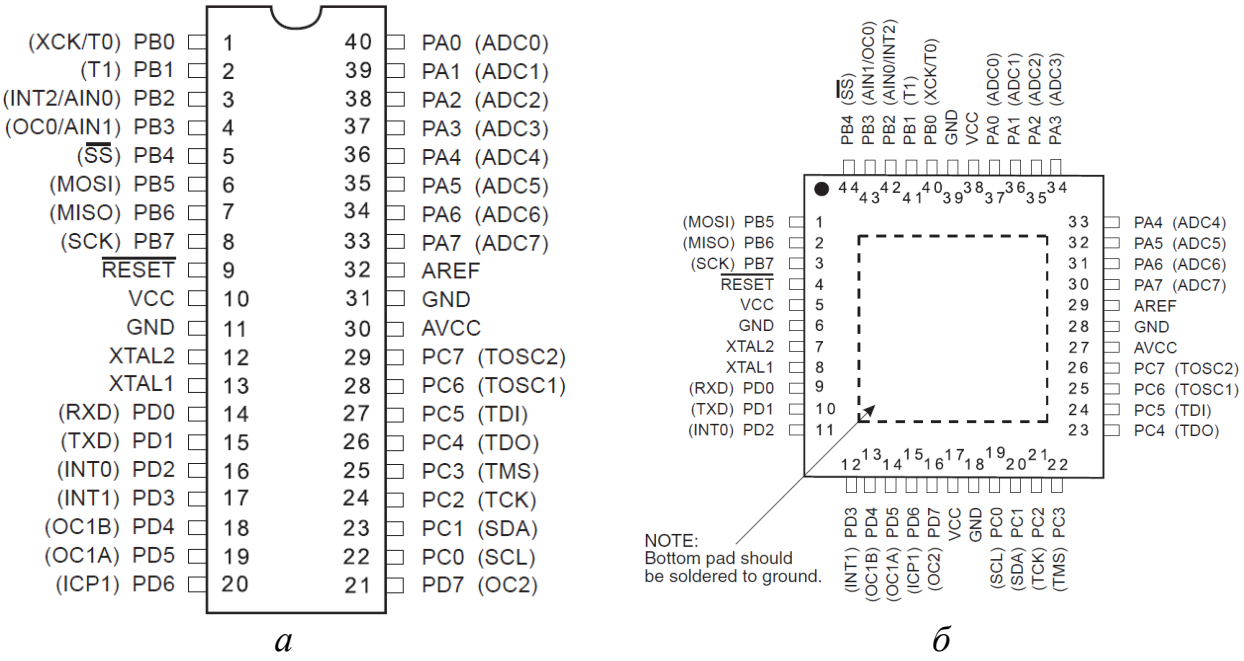
1.2 Порты ввода/вывода МК ATmega16

Микроконтроллер ATmega16 имеет четыре 8-разрядных двунаправленных параллельных порта ввода/вывода A, B, C и D с возможностью подключения внутренних подтягивающих (к напряжению питания) резисторов. Расположение выводов (цоколевка) МК ATmega16 в корпусах PDIP и TQFP/MLF показано на рисунке 2.2.

Каждый порт микроконтроллера включает восемь выводов (линий, «ножек»), через которые МК может осуществлять прием и передачу цифровых сигналов. Задание направления передачи данных через любую линию ввода/вывода может быть выполнено программным образом в любой момент времени без влияния на соседние линии порта.

Выходные буферы всех портов, имея симметричные нагрузочные характеристики, обеспечивают высокую нагрузочную способность при любом уровне сигнала. Нагрузочной способности достаточно для непосредственного управления светодиодными индикаторами. Входные буферы всех выводов построены по схеме триггера Шмитта, который представляет собой электронный двухпозиционный переключающий элемент, передаточная характеристика которого имеет петлю гистерезиса (зону неоднозначности или запаздывания принимаемого решения). Это означает, что у данного элемента имеется два порога переключения – нижний и верхний. Если значение входного напряжения меньше или равно нижнему порогу, то триггером принимается решение, что это состояние логического нуля (лог. 0). Если значение входного напряжения больше или равно верхнему порогу, то принимается решение, что это состояние логической единицы (лог. 1). Если напряжение находится между порогами переключения, то сохраняется значение предыдущего состояния.

Помимо функций ввода/вывода данных все порты имеют альтернативные функции (условные обозначения указаны в скобках на рисунке 2.2), которые используются различными устройствами встроенной периферии МК. При этом возможны две ситуации. В одних случаях требуется самостоятельно задавать конфигурацию вывода, а в других вывод конфигурируется автоматически при включении соответствующего периферийного устройства.



а *б*

а – корпус PDIP; *б* – корпус TQFP/MLF

Рисунок 2.2 – Расположение выводов микроконтроллера ATmega16

Так как все четыре порта ввода/вывода микроконтроллера ATmega16 являются 8-разрядными, то они позволяют принимать и передавать один байт данных за единицу времени. В свою очередь один байт в двоичной системе счисления представляет собой 8-разрядное двоичное число, в каждом разряде которого может находиться лог. 1 или лог. 0. Именно одно из этих двух состояний при обмене данными и формируется на выводах (линиях) портов микроконтроллера. Таким образом, номер каждой линии порта ввода/вывода однозначно связан с определенным разрядом передаваемого байта данных, т. е. номер разряда байта в двоичной системе счисления совпадет с номером в обозначении соответствующей линии порта (рисунок 2.3).

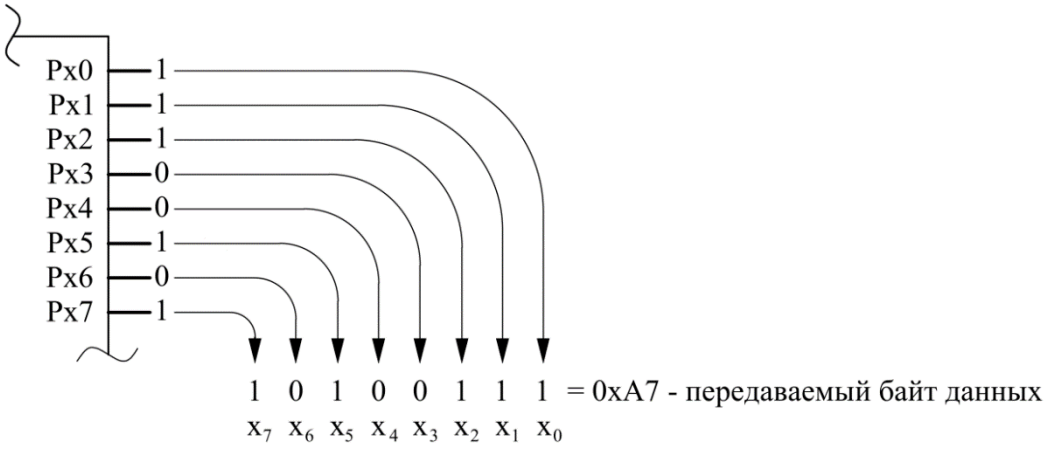


Рисунок 2.3 – Взаимосвязь выводов порта с разрядами передаваемых данных

Для управления каждым портом ввода/вывода используются три специальных регистра ввода/вывода, где x означает имя порта (A; B; C; D):

1 DDRx (Port x Data Direction Register) – регистр направления передаваемых данных через порт x.

Запись лог. 0 в разряды регистра настроит соответствующие линии порта на вход (ввод данных), а запись лог. 1 – на выход (вывод данных). Например:

```
1:    DDRA=0x00;    //Настройка всех линий порта A на вход
2:    DDRB=0x81;    //Настройка 7-й и 0-й линий порта B на выход
```

2 PORTx (Port x Data Register) – регистр вывода данных через порт x.

Если данные необходимо отправить с микроконтроллера на другие устройства, то они записываются в регистр PORTx, при этом используемые линии порта должны быть настроены на выход (вывод данных). Например:

```
1:    DDRC=0xFF;    //Настройка всех линий порта C на выход
2:    PORTC=0xA7;   //Отправка байта данных 0xA7
```

Регистр PORTx также отвечает за подключение внутренних подтягивающих (к напряжению питания V_{CC}) резисторов (Pull-up Resistors), применение которых будет оправдано, когда линии порта ввода/вывода настроены на вход и к ним не подключены никакие внешние устройства. В результате они находятся в свободном (неопределенном) состоянии и могут выступать источниками электромагнитных наводок и помех, которые могут привести к повреждению микроконтроллера. Например:

```
1:    DDRD=0x00;    //Настройка всех линий порта D на вход
2:    PORTD=0xFF;   //Подключение Pull-up резисторов ко всем ли-
      ниям порта D
```

3 PINx (Port x Input Pins Address) – регистр ввода данных через порт x.

Все принимаемые данные записываются в этот регистр. Например:

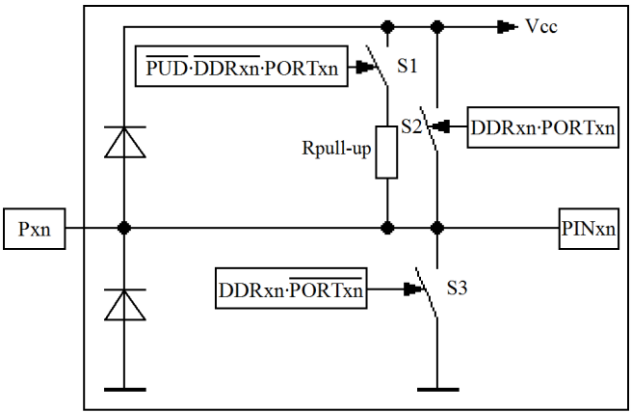
```
1:    DDRA=0x00;    //Настройка всех линий порта A на вход
2:    unsigned char temp;//Объявление переменной
3:    temp=PINA;    //Чтение регистра в переменную temp
```

Также регистр PINx показывает текущее фактическое состояние линий порта независимо от его направления работы, поэтому при подключенных подтягивающих резисторах в регистре будет находиться такое же значение, как и в регистре PORTx.

В МК ATmega16 регистр PINx доступен только для чтения, а регистры DDRx и PORTx – и для чтения, и для записи.

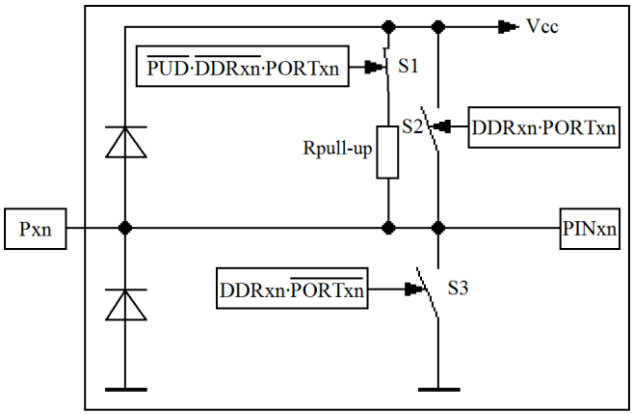
Также необходимо отметить, что в МК ATmega16 общее управление подтягивающими резисторами для всех портов ввода/вывода осуществляется разрядом PUD (Pull-up Disable) в регистре специальных функций ввода/вывода SFIOR (Special Function Input/Output Register). Если разряд PUD сброшен в лог. 0 (состояние по умолчанию), то состояние подтягивающих резисторов будет определяться состоянием разрядов регистра PORTx для каждого порта ввода/вывода. Если разряд PUD установлен в лог. 1, то подтягивающие резисторы отключаются от всех выводов микроконтроллера и становятся недоступными для программного управления.

На рисунке 2.4 представлены упрощенные схемы линий портов ввода/вывода, поясняющие основные режимы работы портов, где x – имя порта, а n – номер вывода. В микроконтроллере используются полевые транзисторы, которые в схемах для простоты объяснения заменены на ключи и диоды. Ключ замыкает цепь, когда результат выражения в рамках равен лог. 1.



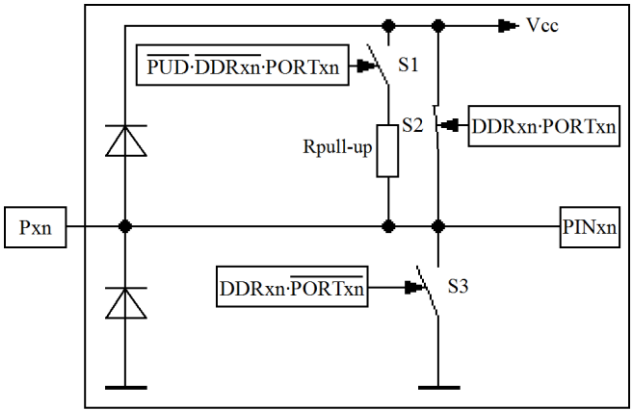
Режим: высокоимпедансный вход (Hi-Z)
PUD=0, DDRxn=0, PORTxn=0

а



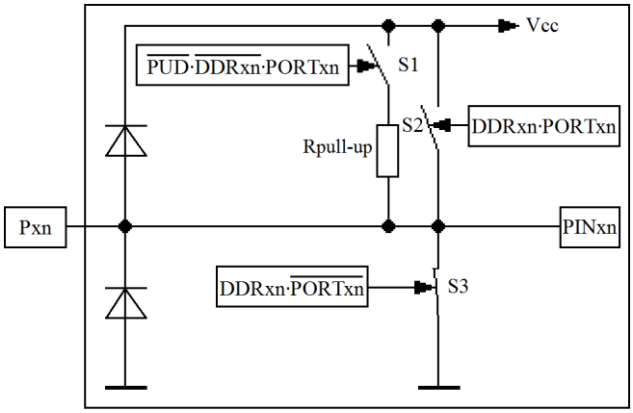
Режим: вход с подтяжкой до лог. 1 (Pull-up)
PUD=0, DDRxn=0, PORTxn=1

б



Режим: выход, состояние лог. 1 (почти V_{CC})
PUD=0, DDRxn=1, PORTxn=1

в



Режим: выход, состояние лог. 0 (почти GND)
PUD=0, DDRxn=1, PORTxn=0

г

Рисунок 2.4 – Основные режимы работы портов ввода/вывода

Пояснения к рисунку 2.4:

1 Входные диоды используются для защиты линий ввода микроконтроллера от превышения напряжения. Если напряжение будет выше напряжения питания, то верхний диод откроется и это напряжение будет «стравлено» на шину питания, где с ним будет уже бороться источник питания и его фильтры. Если на вход попадет отрицательное (ниже нулевого уровня) напряжение, то оно будет нейтрализовано через нижний диод и «погасится» на общий провод (землю). Но в целом диоды выполняют небольшую защиту и смогут защитить лишь от кратковременных импульсных помех.

2 Состояние разряда PUD по умолчанию – лог. 0.

3 Режимы входа:

3.1 Режим высокоимпедансного входа (Hi-Z-состояние или третье состояние).

Это режим по умолчанию, который устанавливается после включения источника питания или появления сигнала сброса на входе $\overline{\text{RESET}}$ (DDRxn=0 и PORTxn=0). Все ключи разомкнуты (см. рисунок 2.4, *а*), а сопротивление порта очень велико. Порт постоянно считывает напряжение на выводах в регистр PINx. Этот режим можно использовать для прослушивания какой-либо шины данных, т. к. он не оказывает на шину никакого влияния. Но если к линиям порта не подключены никакие внешние устройства, то напряжение на них может принимать различные значения от электромагнитных наводок и помех, что может привести к повреждению микроконтроллера.

3.2 Режим входа с подтяжкой до лог. 1.

Если в разряд регистра направления DDRxn записать лог. 0 и в аналогичный разряд регистра вывода PORTxn записать лог. 1 (DDRxy=0 и PORTxy=1), то ключ подтяжки S1 замыкается (см. рисунок 2.4, *б*) и к данной линии подключается внутренний высокоомный резистор. В результате на выводе появляется напряжение, близкое к напряжению источника питания, что соответствует состоянию лог. 1. Цель подтяжки – исключить случайное изменение напряжения на линии под действием электромагнитных наводок. Но если на входе появится лог. 0 (например, в результате замыкания линии на землю тактовой кнопкой или другой микросхемой), то внутренний подтягивающий резистор не сможет удерживать напряжение на линии на уровне лог. 1 и на входе будет лог. 0 (напряжение около нуля вольт).

4 Режимы выхода:

4.1 Если необходимо сформировать на линии порта лог. 1, то в соответствующий разряд регистра направления DDRxn и аналогичный разряд регистра вывода PORTxn требуемого порта записываются лог. 1 (DDRxn=1 и PORTxn=1), в результате замыкается ключ S2 (см. рисунок 2.4, *в*) и на выводе появляется напряжение, близкое к напряжению источника питания.

4.2 Если требуется сформировать на линии порта лог. 0, то в регистр вывода PORTxn записывается лог. 0 (DDRxn=1 и PORTxn=0), и уже замыкается ключ S3 (см. рисунок 2.4, *г*), что дает на выводе напряжение около нуля вольт.

В таблице 2.2 приведены все возможные варианты настройки линий портов ввода/вывода, где X – это любое логическое состояние.

Таблица 2.2 – Настройки линий портов ввода/вывода

DDRxn	PORTxn	PUD	Функция линии	Pull-Up резистор	Примечание
0	0	X	Вход	Отключен	Третье состояние (Hi-Z-состояние)
0	1	0	Вход	Подключен	При подключении нагрузки между линией Pxn и общим проводом линия является источником низкого тока
0	1	1	Вход	Отключен	Третье состояние (Hi-Z-состояние)
1	1	X	Выход	Отключен	Состояние линии установлено в лог. 1
1	0	X	Выход	Отключен	Состояние линии сброшено в лог. 0

Следует отметить, что при переключении линии между третьим состоянием (DDRxn=0 и PORTxn=0) и выходным состоянием лог. 1 (DDRxn=1 и PORTxn=1) происходит переход через одно из промежуточных состояний: либо подключается подтягивающий резистор (DDRxn=0 и PORTxn=1), либо линия переключается в выходное состояние лог. 0 (DDRxn=1 и PORTxn=0). Аналогичная ситуация возникает и при переключении между состоянием с включенным подтягивающим резистором (DDRxn=0 и PORTxn=1) и выходным состоянием лог. 0 (DDRxn=1 и PORTxn=0). В этом случае промежуточным состоянием является либо высокоимпедансное состояние (DDRxn=0 и PORTxn=0), либо выходное состояние лог. 1 (DDRxn=1 и PORTxn=1).

Также следует отметить, что после изменения разрядов регистров DDRxn и PORTxn действительное изменение состояния линий порта ввода/вывода осуществляется с задержкой, величина которой может составлять 0,5–1,5 периодов тактового сигнала, поэтому между командами изменения и последующего считывания состояний линий порта ввода/вывода необходимо вставлять пустую команду (не выполняет никаких значимых действий, но время на ее выполнение центральным процессором затрачивается), например, можно использовать макрос _NOP() при подключенном заголовочном файле <*avr/cpufunc.h*> следующим образом:

```
1:  #include <avr/cpufunc.h> //Подключение библиотеки
2:  unsigned char temp; //Объявление переменной беззнакового символьно-
    го типа
3:  DDRA=0xFF;     //Настройка всех линий порта A на выход
4:  PORTA=0xFE;    //Отправка байта данных 0xFE
5:  _NOP();        //Синхронизация
6:  temp=PINA;     //Чтение регистра в переменную temp. В temp будет 0xFE
```

1.3 Применение операторов языка C/C++ для работы с разрядами регистров

1.3.1 Оператор *поразрядный сдвиг влево* (<<)

Формат записи оператора <<:

Переменная = (*Переменная* << *Число_разрядов_для_сдвига*);

Оператор << выполняет поразрядный сдвиг значения переменной влево на указанное количество разрядов с заполнением освободившихся разрядов справа логическими нулями. При выходе за допустимые пределы типа данных сдвигаемые разряды теряются, кроме того, можно сдвигать числовые константы (таблица 2.3).

Таблица 2.3 – Примеры выполнения оператора *поразрядный сдвиг влево*

Операция	Значение переменной в двоичной системе счисления								Десятичное значение
unsigned char temp = 0x01;	0	0	0	0	0	0	0	1	1
temp = (temp << 3);	0	0	0	0	1	0	0	0	8
unsigned char temp = 0x07;	0	0	0	0	0	1	1	1	7
temp <<= 6;	1	1	0	0	0	0	0	0	192
temp = (1 << 7);	1	0	0	0	0	0	0	0	128

1.3.2 Оператор *поразрядный сдвиг вправо* (>>)

Формат записи оператора >>:

Переменная = (*Переменная* >> *Число_разрядов_для_сдвига*);

Оператор >> выполняет поразрядный сдвиг значения переменной вправо на указанное количество разрядов с заполнением освободившихся разрядов слева логическими нулями. При выходе за нулевой разряд переменной сдвигаемые разряды теряются (таблица 2.4).

Таблица 2.4 – Примеры выполнения оператора *поразрядный сдвиг вправо*

Операция	Значение переменной в двоичной системе счисления								Десятичное значение
unsigned char temp = 0xA0;	1	0	1	0	0	0	0	0	160
temp = (temp >> 3);	0	0	0	1	0	1	0	0	20
unsigned char temp = 0xBA;	1	0	1	1	1	0	1	0	186
temp >>= 2;	0	0	1	0	1	1	1	0	46

1.3.3 Оператор *поразрядное отрицание* (инверсия, ~)

Оператор ~ инвертирует каждый разряд, т. е. лог. 1 заменяется на лог. 0 и наоборот. Этот оператор – унарный, т. е. применяется к одному операнду (таблица 2.5).

Таблица 2.5 – Примеры выполнения оператора *поразрядное отрицание*

Операция	Значение переменной в двоичной системе счисления								Десятичное значение
unsigned char temp = 0xEA;	1	1	1	0	1	0	1	0	234
temp = ~ temp;	0	0	0	1	0	1	0	1	21
unsigned char temp = 0xF0;	1	1	1	1	0	0	0	0	240
temp = ~ temp;	0	0	0	0	1	1	1	1	15

1.3.4 Оператор *поразрядное И* (&)

Оператор & применяет к операндам операцию поразрядного логического умножения. Это означает, что разряд результата будет равен лог. 1 только в том случае, когда соответствующий разряд обоих операндов содержит лог. 1. В противном случае разряд будет равен лог. 0 (таблица 2.6).

Таблица 2.6 – Пример выполнения оператора *поразрядное И*

Операция	Значение переменной в двоичной системе счисления								Десятичное значение
unsigned char a, b, c;	–								–
a = 0xFF;	1	1	1	1	1	1	1	1	255
b = 0x80;	1	0	0	0	0	0	0	0	128
c = a & b;	1	0	0	0	0	0	0	0	128

1.3.5 Оператор *поразрядное ИЛИ* (|)

Оператор | применяет к операндам операцию поразрядного логического сложения. Это означает, что разряд результата будет равен лог. 1, если лог. 1 содержится в соответствующем разряде хотя бы одного операнда (таблица 2.7).

Таблица 2.7 – Пример выполнения оператора *поразрядное ИЛИ*

Операция	Значение переменной в двоичной системе счисления								Десятичное значение
unsigned char a, b, c;	–								–
a = 0xE0;	1	1	1	0	0	0	0	0	224
b = 0x07;	0	0	0	0	0	1	1	1	7
c = a | b;	1	1	1	0	0	1	1	1	231

1.3.6 Оператор *поразрядное исключающее ИЛИ* (^)

Оператор ^ применяет к операндам операцию поразрядного логического исключающего ИЛИ, когда в каждый разряд результата записывается лог. 1, если соответствующие разряды операндов различаются, или лог. 0, если они совпадают (таблица 2.8).

Таблица 2.8 – Пример выполнения оператора *поразрядное исключающее ИЛИ*

Операция	Значение переменной в двоичной системе счисления								Десятичное значение
unsigned char a, b, c;	–								–
a = 0x9A;	1	0	0	1	1	0	1	0	154
b = 0xF1;	1	1	1	1	0	0	0	1	241
c = a ^ b;	0	1	1	0	1	0	1	1	107

1.3.7 Запись логической единицы в некоторый разряд регистра с обнулением остальных разрядов

Для записи лог. 1 в некоторый разряд регистра с обнулением остальных разрядов можно воспользоваться оператором поразрядного сдвига влево <<, используя формат записи

Регистр = (1 << *Номер_разряда*);

Запись нескольких лог. 1 с обнулением остальных разрядов:

Регистр = (1 << *Номер_разряда*) | (1 << *Номер_разряда*)… | (1 << *Номер_разряда*);

Рассмотрим пример команды в Atmel Studio со схемой вычисления (таблица 2.9)

```
PORTA = (1 << PA6) | (1 << PA5) | (1 << PA1);
```

Таблица 2.9 – Пример выполнения команды пункта 1.3.7

Операция	Значение в двоичной системе счисления								Десятичное значение
(1 << PA6)	0	1	0	0	0	0	0	0	64
ИЛИ	|								–
(1 << PA5)	0	0	1	0	0	0	0	0	32
ИЛИ	|								–
(1 << PA1)	0	0	0	0	0	0	1	0	2
Результат	0	1	1	0	0	0	1	0	98

Альтернативный вариант – просто присвоить регистру значение с лог. 1 в требуемых разрядах: `PORTA = 0b01100010;`

1.3.8 Запись логической единицы в некоторый разряд регистра без обнуления остальных разрядов

Для записи лог. 1 в некоторый разряд регистра без обнуления остальных разрядов используется формат записи

Регистр |= (1 << *Номер_разряда*);

Запись нескольких лог. 1 без обнуления остальных разрядов:

Регистр |= (1 << *Номер_разряда*) | (1 << *Номер_разряда*)… | (1 << *Номер_разряда*);

Рассмотрим пример. Предположим, что `PORTB = 0xA2`. Тогда следующая команда в Atmel Studio будет выполнена по схеме, приведенной в таблице 2.10:

```
PORTB |= (1 << PB4) | (1 << PB2) | (1 << PB0);
```

Таблица 2.10 – Пример выполнения команды пункта 1.3.8

Операция	Значение в двоичной системе счисления								Десятичное значение
PORTB	1	0	1	0	0	0	1	0	162
ИЛИ					|				–
(1 << PB4)	0	0	0	1	0	0	0	0	16
ИЛИ					|				–
(1 << PB2)	0	0	0	0	0	1	0	0	4
ИЛИ					|				–
(1 << PB0)	0	0	0	0	0	0	0	1	1
Результат	1	0	1	1	0	1	1	1	183

1.3.9 Запись логического нуля в некоторый разряд регистра без обнуления остальных разрядов

Для записи лог. 0 в некоторый разряд регистра без обнуления остальных разрядов используется формат записи

Регистр &= ~ (1 << *Номер_разряда*);

Запись нескольких лог. 0 без обнуления остальных разрядов:

Регистр &= ~ (1 << *Номер_разряда*) & ~ (1 << *Номер_разряда*)… & ~ (1 << *Номер_разряда*);

Или

Регистр &= ~ ((1 << *Номер_разряда*) | (1 << *Номер_разряда)*… | (1 << *Номер_разряда*));

Рассмотрим пример. Предположим, что `DDRC = 0xFF`. Тогда следующая команда в Atmel Studio будет выполнена по схеме, приведенной в таблице 2.11:

```
DDRC &= ~ (1 << PC7) & ~ (1 << PC3) & ~ (1 << PC1);
```

Таблица 2.11 – Пример выполнения команды пункта 1.3.9

Операция	Значение в двоичной системе счисления								Десятичное значение
DDRC	1	1	1	1	1	1	1	1	255
И	&								–
~(1 << PC7)	0	1	1	1	1	1	1	1	127
И	&								–
~(1 << PC3)	1	1	1	1	0	1	1	1	247
И	&								–
~(1 << PC1)	1	1	1	1	1	1	0	1	253
Результат	0	1	1	1	0	1	0	1	117

1.3.10 Запись логического нуля в несколько разрядов с записью в остальные разряды логической единицы

Для записи лог. 0 в несколько разрядов регистра с записью лог. 1 в остальные разряды используется формат записи

Регистр = ~ (1 << *Номер_разряда*) & ~ (1 << *Номер_разряда*)… & ~ (1 << *Номер_разряда*);

Или

Регистр = ~ ((1 << *Номер_разряда*) | (1 << *Номер_разряда)*… | (1 << *Номер_разряда*));

1.3.11 Проверка некоторого разряда регистра на наличие логического нуля

Формат записи проверки с помощью оператора выбора *if*:

```
if ((Регистр & (1 << Номер_разряда)) = = 0)
{ Оператор1; }
else
{ Оператор2; }
```

Оператор1 выполнится в том случае, если в указанном разряде регистра будет лог. 0. В противном случае выполнится *Оператор2*.

Рассмотрим следующий пример команды в Atmel Studio:

```
if ((PINA & (1 << PA0)) == 0)
{ Оператор1; }
else
{ Оператор2; }
```

Первый вариант выполнения команды, если в регистре `PINA` находится значение `0b11111111`, показан в таблице 2.12.

Таблица 2.12 – Пример выполнения команды пункта 1.3.11 для первого варианта

Операция	Значение в двоичной системе счисления								Десятичное значение
PINA	1	1	1	1	1	1	1	1	255
И	&								–
(1 << PA0)	0	0	0	0	0	0	0	1	1
Результат	0	0	0	0	0	0	0	1	1

Результат не равен нулю ($1 \neq 0$), следовательно, логическое условие (выражение) оператора выбора *if* возвратит значение логического типа *Ложь* (*False*). Таким образом, выполнится *Оператор2*.

Второй вариант выполнения команды, если в регистре `PINA` находится значение `0b11111110`, показан в таблице 2.13.

Таблица 2.13 – Пример выполнения команды пункта 1.3.11 для второго варианта

Операция	Значение в двоичной системе счисления								Десятичное значение
PINA	1	1	1	1	1	1	1	0	254
И	&								–
(1 << PA0)	0	0	0	0	0	0	0	1	1
Результат	0	0	0	0	0	0	0	0	0

Результат равен нулю, следовательно, логическое условие (выражение) оператора выбора *if* возвратит значение логического типа *Истина* (*True*). Таким образом, выполнится *Оператор1*.

При подключенном заголовочном файле <*avr/sfr_defs.h*> проверка может быть выполнена с помощью макроса *bit_is_clear* («Разряд очищен (сброшен в лог. 0)?»), который возвращает ненулевое значение (*True*), если в указанном разряде регистра будет лог. 0. Тогда выполнится *Оператор1*. Если разряд до сих пор установлен в лог. 1, то макрос возвратит значение *False*, и выполнится *Оператор2*:

```
if ( bit_is_clear (Регистр, Номер_разряда))
{ Оператор1; }
else
{ Оператор2; }
```

1.3.12 Проверка некоторого разряда регистра на наличие логической единицы

Формат записи проверки:

```
if ((Регистр & (1 << Номер_разряда)) != 0)
{ Оператор1; }
else
{ Оператор2; }
```

Оператор1 выполнится, если в указанном разряде регистра будет лог. 1. В противном случае выполнится *Оператор2*.

Рассмотрим следующий пример команды в Atmel Studio:

```
if ((PINB & (1 << PB7)) != 0)
{ Оператор1; }
else
{ Оператор2; }
```

Первый вариант выполнения команды, если в регистре `PINB` находится значение `0b01111111`, показан в таблице 2.14.

Таблица 2.14 – Пример выполнения команды пункта 1.3.12 для первого варианта

Операция	Значение в двоичной системе счисления								Десятичное значение
PINB	0	1	1	1	1	1	1	1	127
И	&								–
(1 << PB7)	1	0	0	0	0	0	0	0	128
Результат	0	0	0	0	0	0	0	0	0

Результат равен нулю, но в условии проверяется, чтобы результат был не равен нулю, поэтому логическое условие не выполняется и возвращает значение *False*. Поэтому выполнится *Оператор2*.

Второй вариант выполнения команды, если в регистре `PINB` находится значение `0b11111111`, показан в таблице 2.15.

Таблица 2.15 – Пример выполнения команды пункта 1.3.12 для второго варианта

Операция	Значение в двоичной системе счисления								Десятичное значение
PINB	1	1	1	1	1	1	1	1	255
И	&								–
(1 << PB4)	1	0	0	0	0	0	0	0	128
Результат	1	0	0	0	0	0	0	0	128

Результат не равен нулю (128 ≠ 0), логическое условие выполняется и возвращает значение *True*. Поэтому выполнится *Оператор1*.

При подключенном заголовочном файле <*avr/sfr_defs.h*> проверка может быть выполнена с помощью макроса *bit_is_set* («Разряд установлен в лог. 1?»), который возвращает ненулевое значение (*True*), если в указанном разряде регистра будет лог. 1. Тогда оператор *if* перейдет к выполнению блока команд *Оператор1*. Если разряд до сих пор не установлен в лог. 1, то макрос возвратит нулевое значение (*False*), и выполнится *Оператор2*:

```
if ( bit_is_set (Регистр, Номер_разряда))
{ Оператор1; }
else
{ Оператор2; }
```

1.3.13 Ожидание появления логического нуля в некотором разряде регистра

Формат записи ожидания с помощью оператора цикла *while*:

```
while ((Регистр & (1 << Номер_разряда)) != 0);
```

Цикл ожидания будет выполняться до тех пор, пока разряд в регистре содержит лог. 1. Как только в разряде появится лог. 0, произойдет выход из цикла. Рассмотрим следующий пример команды в Atmel Studio:

```
while ((PINC & (1 << PC1)) != 0);
```

Первый вариант выполнения команды, если в регистре `PINC` находится значение `0b11111111`, показан в таблице 2.16.

Таблица 2.16 – Пример выполнения команды пункта 1.3.13 для первого варианта

Операция	Значение в двоичной системе счисления								Десятичное значение
PINC	1	1	1	1	1	1	1	1	255
И	&								–
(1 << PC1)	0	0	0	0	0	0	1	0	2
Результат	0	0	0	0	0	0	1	0	2

Результат не равен нулю (2 ≠ 0), следовательно, первый разряд регистра `PINC` содержит лог. 1, логическое условие выполняется, и выход из цикла не произойдет.

Второй вариант выполнения команды, если в первом разряде регистра `PINC` появится лог. 0, показан в таблице 2.17.

Таблица 2.17 – Пример выполнения команды пункта 1.3.13 для второго варианта

Операция	Значение в двоичной системе счисления								Десятичное значение
PINC	1	1	1	1	1	1	0	1	255
И	&								–
(1 << PC1)	0	0	0	0	0	0	1	0	1
Результат	0	0	0	0	0	0	0	0	0

Результат равен нулю, логическое условие не выполняется, и произойдет выход из цикла.

При подключенном заголовочном файле *<avr/sfr_defs.h>* ожидание может быть выполнено с помощью макроса *loop_until_bit_is_clear* («Зациклить до тех пор, пока разряд не будет сброшен в лог. 0»), работа которого аналогична рассмотренному выше примеру. Формат записи макроса:

`loop_until_bit_is_clear` (*Регистр*, *Номер_разряда*);

1.3.14 Ожидание появления логической единицы в некотором разряде регистра

Формат записи ожидания:

`while` ((*Регистр* & (1 << *Номер_разряда*)) = = 0);

Цикл ожидания будет выполняться до тех пор, пока разряд в регистре содержит лог. 0. Как только в разряде появится лог. 1, произойдет выход из цикла. Рассмотрим следующий пример команды в Atmel Studio:

```
while ((PIND & (1 << PD6)) == 0);
```

Первый вариант выполнения команды, если в регистре `PIND` находится `0b10111111`, показан в таблице 2.18.

Таблица 2.18 – Пример выполнения команды пункта 1.3.14 для первого варианта

Операция	Значение в двоичной системе счисления								Десятичное значение
PIND	1	0	1	1	1	1	1	1	191
И	&								-
(1 << PD6)	0	1	0	0	0	0	0	0	64
Результат	0	0	0	0	0	0	0	0	0

Результат равен нулю, следовательно, шестой разряд регистра `PIND` содержит лог. 0, логическое условие выполняется, и выход из цикла не произойдет.

Второй вариант выполнения команды, если в шестом разряде регистра `PIND` появится лог. 1, показан в таблице 2.19.

Таблица 2.19 – Пример выполнения команды пункта 1.3.14 для второго варианта

Операция	Значение в двоичной системе счисления								Десятичное значение
PIND	1	1	1	1	1	0	0	0	255
И	&								-
(1 << PD6)	0	1	0	0	0	0	0	0	64
Результат	0	1	0	0	0	0	0	0	64

Результат не равен нулю (64 ≠ 0), логическое условие не выполняется, и произойдет выход из цикла.

При подключенном заголовочном файле *<avr/sfr_defs.h>* ожидание может быть выполнено с помощью макроса *loop_until_bit_is_set* («Зациклить до тех пор, пока разряд не будет установлен в лог. 1»), работа которого аналогична рассмотренному выше примеру. Формат записи макроса:

`loop_until_bit_is_set` (*Регистр, Номер_разряда*);

1.4 Пример написания программы на C/C++

Требуется написать программу на языке программирования C/C++, выполнение которой микроконтроллером ATmega16 позволит реализовать циклически повторяющееся последовательное включение и выключение восьми светодиодов, расположенных в один ряд («бегущий огонь» из одного светодиода). При этом необходимо предусмотреть выполнение следующих условий и требований:

1) схема устройства в программе Proteus ISIS представлена на рисунке 2.5;

2) микроконтроллер должен работать на частоте 1 МГц от внутреннего источника тактирования;

3) время переключения (свечения) светодиодов – 200 мс;

4) необходимо реализовать запуск «бегущего огня» по тактовой кнопке SB1 с фиксацией своего состояния следующим образом:

- нажатие кнопки (первое и повторные), которое замыкает цепь, должно запускать «бегущий огонь» с первого светодиода;

- если кнопка отжимается, т. е. размыкает цепь, текущий включенный светодиод должен гаснуть, и все светодиоды должны оставаться выключенными до повторного нажатия кнопки;

5) направление «бегущего огня» задано с VD1 по VD8;

6) при переходе «бегущего огня» со светодиода VD8 на светодиод VD1 должна отсутствовать задержка по времени. Это необходимо проверить с помощью встроенного программного цифрового осциллографа OCS1, входы A и B которого подключены к отмеченным светодиодам.

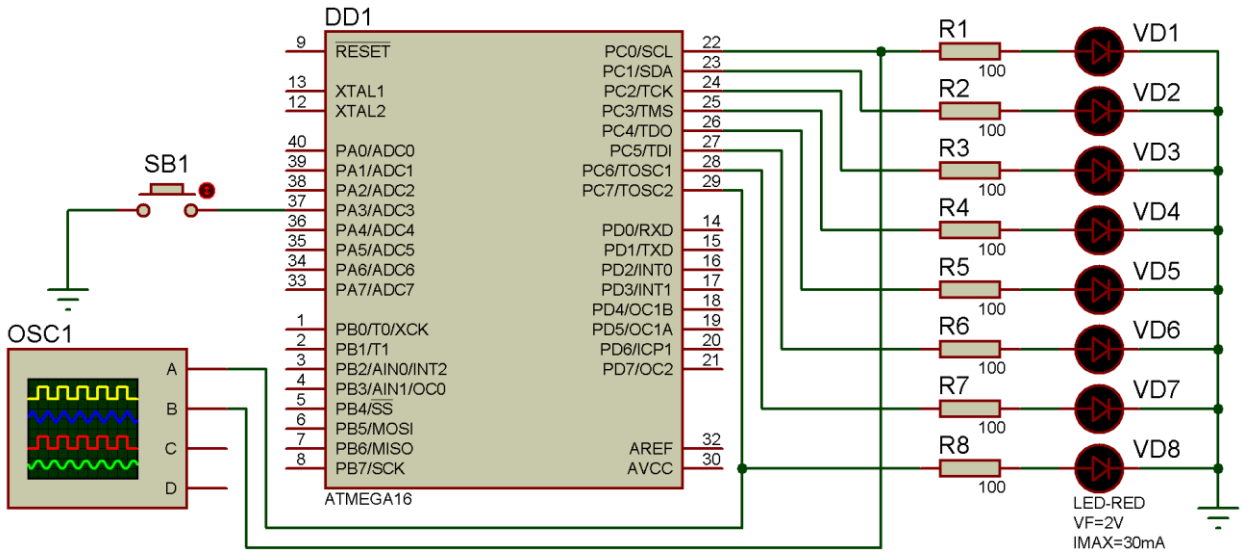


Рисунок 2.5 – Схема устройства, реализующего «бегущий огонь» по восьми светодиодам, в программе Proteus ISIS

Блок-схема алгоритма работы программы представлена на рисунке 2.6.

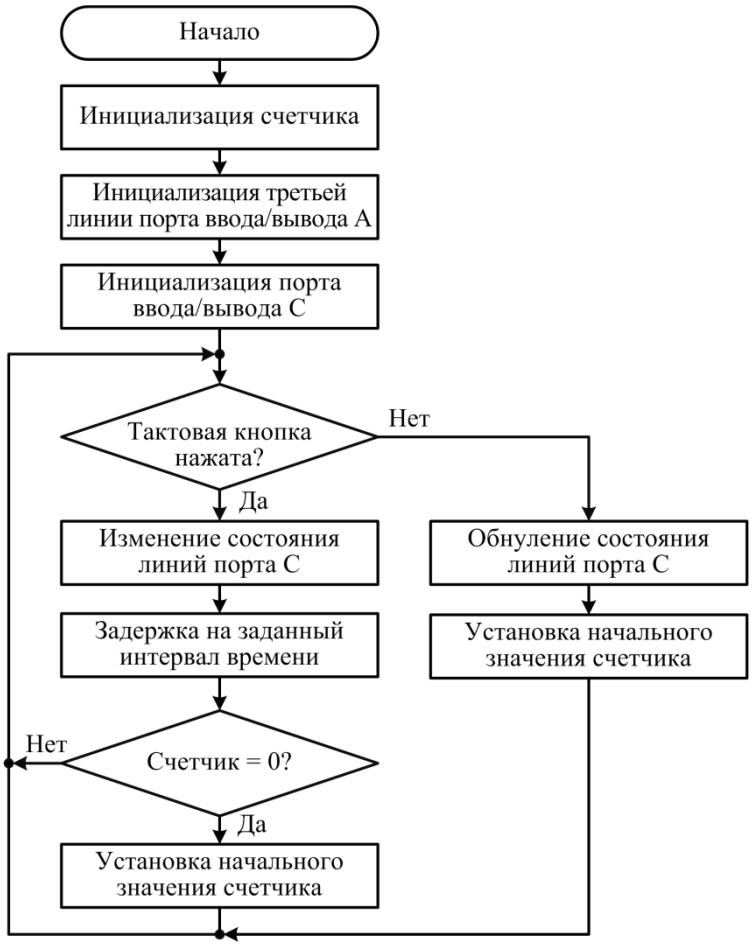


Рисунок 2.6 – Блок-схема алгоритма работы программы для примера

В соответствии с алгоритмом работы программа начинается с инициализации (начальной настройки) счетчика, который будет фиксировать включение и выключение каждого из восьми светодиодов и определять момент завершения каждого цикла «бегущего огня». Дальше выполняется инициализация третьей линии порта ввода/вывода A, которая заключается в настройке этой линии на вход (ввод данных) с подключением внутреннего подтягивающего резистора МК, потому что к этой линии подключено устройство ввода – тактовая кнопка без внешнего подтягивающего резистора, который необходим для проверки работы кнопки. Затем выполняется инициализация порта ввода/вывода C, которая заключается в настройке всех линий порта C на выход (вывод данных) с формированием на них начального состояния лог. 0, потому что к линиям порта C подключены устройства вывода – светодиоды, которые до нажатия кнопки должны быть выключены. Светодиоды подключены через токоограничивающие резисторы.

Далее находится блок ветвления, в котором проверяется, нажата ли тактовая кнопка, а именно проверяется состояние третьей линии порта ввода/вывода A. В начальном состоянии тактовая кнопка не нажата и цепь разомкнута. Благодаря подключенному внутреннему подтягивающему резистору на этой линии порта A будет находиться напряжение, близкое к напряжению источника питания, что соответствует состоянию лог. 1. При нажатии кнопки третья линия МК будет замыкаться на общий провод (землю), и состояние линии будет меняться на лог. 0. С помощью программы МК фиксирует эти изменения и выполняет переход по соответствующей ветви «Да» или «Нет».

Если кнопка нажата (ветвь «Да»), то состояние каждой линии порта C изменяется таким образом, чтобы светодиоды загорались последовательно, и каждый из них был включен в течение заданного интервала времени, который в свою очередь формируется с помощью функции задержки по времени в следующем блоке алгоритма. Затем проверяется обнуление счетчика. Если счетчик не равен нулю, то выполняется переход в начало цикла для проверки кнопки и выполнения переключения на следующий светодиод. Если счетчик равен нулю, то это означает, что «бегущий огонь» прошел по всем светодиодам и должен начаться повторно с первого светодиода. Для этого выполняется запись начального значения в счетчик. После чего весь цикл «бегущего огня» будет повторен.

Если кнопка не была нажата или отжимается (ветвь «Нет»), то происходит обнуление состояния всех линий порта C, в результате чего все светодиоды будут выключены. В следующем блоке алгоритма осуществляется запись начального значения в счетчик для запуска «бегущего огня» с первого светодиода.

Исходный текст программы на языке программирования C/C++ для рассмотренной блок-схемы представлен в листинге 2.1 и заключается в следующем.

В строке 1 с помощью директивы препроцессора *#define* определена символьная константа F_CPU, которая обозначает тактовую частоту микроконтроллера и необходима для правильной работы функции задержки по времени в строке 16. В соответствии с заданием указана частота 1 000 000 Гц.

В строке 2 программы с помощью директивы *#include* подключается стандартный заголовочный файл, который в свою очередь подключает библио-

теку идентификаторов для используемой модели микроконтроллера, выбранной при создании проекта в среде разработки Atmel Studio.

В строке 3 подключается библиотека *delay.h*, содержащая различные функции задержки по времени.

В строке 4 начинается главная функция программы, которая состоит из однократно выполняемых команд различных инициализаций (строки 6–10) и многократно (циклически) выполняемых команд (строки 11–28), находящихся в теле оператора цикла *while*, у которого логическое выражение всегда принимает значение *Истина*. Рассмотрим эти команды.

Листинг 2.1

```
1:    #define F_CPU 1000000UL
2:    #include <avr/io.h>
3:    #include <util/delay.h>
4:    int main(void)
5:    {
6:        unsigned char counter=1;
7:        DDRA&=~(1<<PA3);
8:        PORTA|=(1<<PA3);
9:        DDRC=0xFF;
10:       PORTC=0x00;
11:       while (1)
12:       {
13:           if ((PINA & (1<<PA3))==0)
14:           {
15:               PORTC=counter;
16:               _delay_ms(200);
17:               counter*=2;
18:               if (counter==0)
19:               {
20:                   counter=1;
21:               }
22:           }
23:           else
24:           {
25:               PORTC=0x00;
26:               counter=1;
27:           }
28:       }
29:   }
```

В строке 6 объявляется локальная переменная беззнакового символьного типа *counter*, которая является счетчиком в соответствии с алгоритмом работы программы. Переменная инициализируется значением 1.

В строке 7 выполняется настройка третьей линии порта A на вход (ввод данных) с помощью записи лог. 0 в третий разряд регистра DDRA.

В строке 8 осуществляется подключение внутреннего подтягивающего резистора МК к третьей линии порта A с помощью записи лог. 1 в третий разряд регистра PORTA. Для исключения влияния этих двух команд на остальные линии порта A, которые могут быть использованы в дальнейшем для подключения других устройств ввода/вывода, используются составные операторы: «&=» – поразрядное И, совмещенное с присваиванием, и «|=» – поразрядное ИЛИ, совмещенное с присваиванием, работа которых рассмотрена в пунктах 1.3.8 и 1.3.9.

В строке 9 осуществляется настройка всех линий порта C на вывод данных с помощью записи лог. 1 в каждый разряд регистра DDRC, т. к. 0xFF = 0b11111111, где префикс 0b означает двоичную систему счисления.

В строке 10 выполняется формирование сигналов низкого логического уровня на всех линиях порта с помощью записи лог. 0 в каждый разряд регистра PORTC. В результате выполнения этих двух команд все светодиоды будут находиться в начальном выключенном состоянии.

В строке 11 начинается бесконечный цикл, в котором в строке 13 осуществляется проверка тактовой кнопки SB1 с помощью оператора выбора *if* следующим образом. Регистр PINA показывает текущее фактическое состояние всех линий порта A независимо от его направления работы. Поэтому при подключенном подтягивающем резисторе к третьей линии порта A в регистре PINA будет находиться значение 0b00001000. Операция 1<<PA3 также соответствует значению 0b00001000. Поэтому:

1 Если кнопка SB1 не нажата, то результат выполнения выражения PINA & (1<<PA3) будет

$$0b00001000 \;\&\; 0b00001000 = 0b00001000 = 16 \neq 0. \qquad (2.1)$$

Это означает, что условие в строке 13 не выполняется, и осуществляется переход на блок команд после ключевого слова *else* в строках 24–27.

2 Если кнопка SB1 нажата, то состояние третьей линии порта A изменяется на лог. 0, а результат выполнения выражения PINA & (1<<PA3) будет

$$0b00000000 \;\&\; 0b00001000 = 0b00000000 = 0. \qquad (2.2)$$

Это означает, что условие в строке 13 выполняется и осуществляется переход на блок команд после оператора *if* в строках 14–22.

В строке 15 в регистр PORTC выполняется запись значения переменной *counter*, начальное значение которой равно 1 или 0b00000001. Поэтому после выполнения этой команды сигнал высокого логического уровня будет сформирован на нулевой линии порта C, что приведет к включению первого светодиода VD1. Затем в строке 16 вызывается функция задержки по времени. В течение 200 мс функция будет выполняться, и светодиод будет включен. Далее в строке 17 осуществляется увеличение переменной *counter* в два раза, а в строке 18

проверяется условие обнуления этой переменной. На данной итерации условие обнуления не выполняется, поэтому дальше осуществляется возврат в начало бесконечного цикла (начало оператора *while*).

Если кнопка SB1 еще нажата, то снова будут выполнены команды в строках 15–18. Вначале в регистр PORTC будет записано значение переменной *counter*, равное 2 или 0b00000010, поэтому сигнал высокого логического уровня будет сформирован на первой линии порта C, что приведет к одновременному выключению первого светодиода VD1 и включению второго светодиода VD2. Затем будет реализована задержка в 200 мс, увеличение переменной *counter* в два раза, проверка обнуления счетчика и возврат в начало бесконечного цикла.

Если кнопка SB1 еще нажата, то в регистр PORTC будет записано значение переменной *counter*, равное 4 или 0b00000100, поэтому сигнал высокого логического уровня будет сформирован на второй линии порта C, что приведет к одновременному выключению второго светодиода VD2 и включению третьего светодиода VD3. Таким образом, увеличение переменной в два раза на каждой итерации будет включать следующий светодиод и гасить предыдущий. В таблице 2.20 представлены используемые значения переменной и соответствующее им состояние светодиодов.

Таблица 2.20 – Взаимосвязь переменной *counter* и состояния светодиодов

Номер итерации	Значение переменной *counter*		Состояние светодиодов
	десятичное значение	двоичное значение	
1	1	0b00000001	Включен светодиод VD1
2	2	0b00000010	Включен светодиод VD2
3	4	0b00000100	Включен светодиод VD3
4	8	0b00001000	Включен светодиод VD4
5	16	0b00010000	Включен светодиод VD5
6	32	0b00100000	Включен светодиод VD6
7	64	0b01000000	Включен светодиод VD7
8	128	0b10000000	Включен светодиод VD8
9	0	0b00000000	Все светодиоды выключены

Как видно из таблицы 2.20, на каждой из первых восьми итераций будет включен определенный светодиод. На восьмой итерации умножение переменной на два приведет к появлению числа 256 или 0b100000000, которое при записи в переменную *counter* будет усечено до восьми младших разрядов и составит значение 0. Это объясняется тем, что в строке 6 для переменной *counter* был выбран беззнаковый символьный тип. Переменные этого типа имеют объем 1 байт или диапазон чисел от 0 по 255, т. е. это 8-разрядные двоичные числа.

Таким образом, на восьмой итерации переменная *counter* примет нулевое значение, и условие в строке 18 будет выполнено, в результате чего в переменную *counter* будет записано значение, равное единице. Поэтому девятая итерация не произойдет. После возврата в начало цикла вся последовательность команд будет снова повторяться, что соответствует новому циклу «бегущего огня».

Если в какой-либо момент времени тактовая кнопка SB1 будет отжата, то, как выше было отмечено, будет выполнен переход на блок команд после ключевого слова *else*. В строке 25 в регистр PORTC будет записано нулевое значение. В результате сигналы низкого логического уровня будут сформированы на всех линиях порта C, что приведет к выключению всех светодиодов. В строке 26 в переменную *counter* будет записано начальное значение, равное единице, чтобы после нажатия тактовой кнопки SB1 включился первый светодиод VD1.

Необходимо обратить внимание, что используемая тактовая кнопка имеет дребезг контактов. Эффект дребезга контактов заключается в многократных неконтролируемых замыканиях и размыканиях контактов в электромеханических коммутационных устройствах за счет упругости материалов и деталей контактной системы в момент нажатия и отпускания кнопки (переключателя). Это приводит к последовательности ложных импульсов (рисунок 2.7), регистрируя которые микроконтроллер будет принимать их за отдельные нажатия и обрабатывать в соответствии с программой, что является недопустимым. Для устранения дребезга контактов могут быть использованы аппаратные способы, например, на основе RC-фильтра или RS-триггера, либо программные способы, например, в виде задержки по времени.

В исходном тексте программы (см. листинг 2.1) защита от дребезга контактов тактовой кнопки SB1 реализуется с помощью функции задержки в строке 16.

На рисунке 2.8 представлены осциллограммы сигналов на нулевой и седьмой линиях порта C, полученные для листинга 2.1 в программе Proteus ISIS с помощью встроенного программного цифрового осциллографа OCS1. Как видно из рисунка, длительность импульсов, в течение которых будут включены светодиоды, составляет требуемые 200 мс, и отсутствует задержка перехода со светодиода VD8 на светодиод VD1, потому что фронты прямоугольных импульсов совпадают.

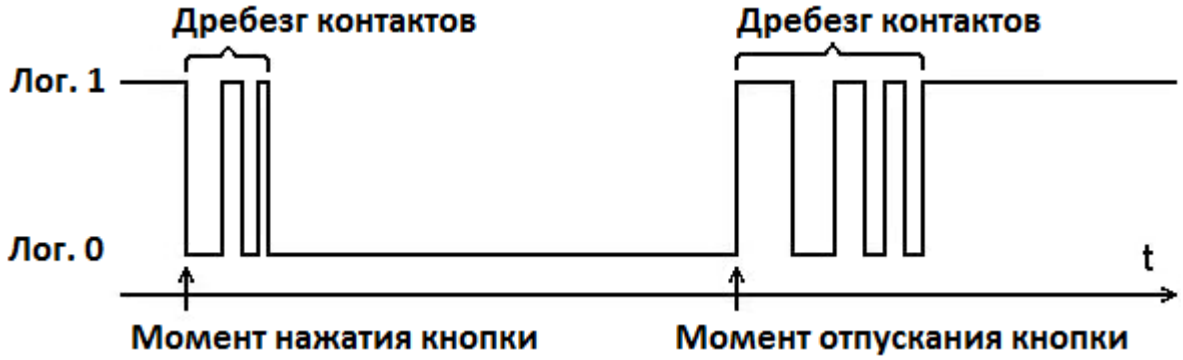


Рисунок 2.7 – Ложные импульсы дребезга контактов кнопки

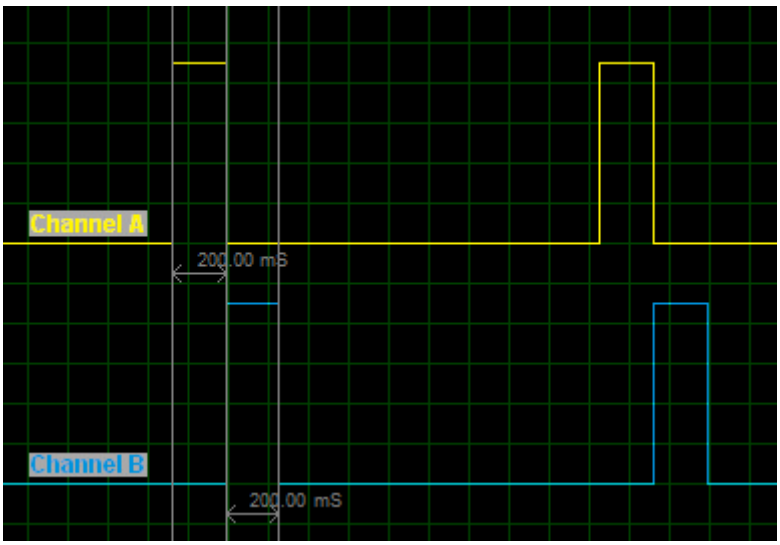


Рисунок 2.8 – Осциллограммы сигналов на нулевой и седьмой линиях порта C

2 Порядок выполнения работы

Задание 1

Требуется написать программу на языке программирования C/C++, выполнение которой микроконтроллером ATmega16 позволит реализовать циклически повторяющееся последовательное включение и выключение 16 светодиодов, расположенных в один ряд («бегущий огонь» из одного светодиода). При этом необходимо предусмотреть выполнение следующих условий и требований:

1) исследуемая схема устройства в программе Proteus ISIS представлена в приложении B;

2) микроконтроллер должен работать от внутреннего источника тактирования;

3) время переключения (свечения) светодиодов и тактовая частота микроконтроллера заданы в таблице 2.21;

4) необходимо реализовать запуск «бегущего огня» по тактовой кнопке с фиксацией своего состояния следующим образом:

- нажатие кнопки (первое и повторные), которое замыкает цепь, должно запускать «бегущий огонь» с первого или последнего светодиода, что зависит от заданного варианта;

- если кнопка отжимается, т. е. размыкает цепь, текущий включенный светодиод должен гаснуть, и все светодиоды должны оставаться выключенными до повторного нажатия кнопки;

5) в качестве управляющей кнопки необходимо выбрать одну кнопку из 10-кнопочного DIP-переключателя SB1, подключенного к портам C и D. Номер кнопки равен заданному варианту;

6) для нечетных номеров вариантов направление «бегущего огня» задано слева направо (с VD1 по VD16), а для четных – справа налево (с VD16 по VD1);

7) при написании программы необходимо использовать оператор множественного выбора *switch* для реализации правильного перехода между группами из восьми светодиодов, подключенных к разным портам ввода/вывода таким образом, чтобы первый *case* позволил реализовать «бегущий огонь» по первой группе светодиодов, а второй *case* – по второй группе светодиодов.

Таблица 2.21 – Исходные данные для выполнения задания 1 лабораторной работы №2

Номер варианта	Тактовая частота микроконтроллера, МГц	Время переключения светодиодов, мс
1	2	200
2	4	300
3	8	400
4	2	500
5	4	200
6	8	300
7	2	400
8	4	500
9	8	200
10	2	300

Задание 2

Проверить работоспособность составленной программы для заданной схемы, выполнив моделирование в программе Proteus ISIS.

С помощью встроенного программного цифрового осциллографа OCS1 проверить длительность свечения светодиодов и убедиться в правильности переходов между парами светодиодов: VD8 и VD9; VD1 и VD16. Для этого входы осциллографа OCS1 подключены к отмеченным светодиодам. По полученным осциллограммам с помощью визирных линий необходимо убедиться в совпадении фронтов прямоугольных импульсов и измерить длительность импульсов, которая должна соответствовать заданному времени переключения светодиодов.

3 Содержание отчета

3.1 Титульный лист.

3.2 Цель работы.

3.3 Исследуемая схема устройства в Proteus ISIS (см. приложение В).

3.4 Исходные данные для выполнения задания 1.

3.5 Блок-схема алгоритма работы разработанной программы.

3.6 Листинг программы на языке программирования C/C++ с подробными комментариями, поясняющими назначение каждой используемой команды.

3.7 Осциллограммы, полученные в программе моделирования Proteus ISIS и подтверждающие правильность работы составленной программы. На осциллограммах с помощью визирных линий должны быть показаны длительности импульсов.

4 Контрольные вопросы

4.1 По какой архитектуре строится память в микроконтроллере ATmega16? Какие типы памяти различают в этом микроконтроллере? Укажите пределы их адресных пространств и их объемы.

4.2 Какие дополнительные названия имеют память программ и данных? В чем отличие этих типов памяти?

4.3 Каково назначение памяти программ и данных? Из каких частей они состоят?

4.4 Для чего необходимы РОН и РВВ? На какие группы делятся РОН и РВВ?

4.5 Из каких типов состоит память данных в микроконтроллере ATmega16? Опишите их.

4.6 Что такое стек и для чего используется ЭСППЗУ в микроконтроллере?

4.7 Какая адресация и какой разрядности шины адреса и данных используются при обращении к памяти программ и данных? Почему?

4.8 Сколько портов ввода/вывода в микроконтроллере ATmega16? Охарактеризуйте их (название, разрядность, вид).

4.9 Какие регистры ввода/вывода используются для управления портами микроконтроллера? Укажите их назначение и приведите примеры.

4.10 Настройте любой порт ввода/вывода МК на выход и запишите в него число 100.

4.11 Опишите основные режимы работы портов ввода/вывода МК.

4.12 Какие операторы языка программирования C/C++ применяются для работы с разрядами регистров?

4.13 С помощью какой команды на языке программирования C/C++ можно установить в лог. 1 состояние определенной линии порта ввода/вывода микроконтроллера ATmega16, не влияя на состояние остальных линий этого порта?

4.14 С помощью какой команды на языке программирования C/C++ можно сбросить в лог. 0 состояние определенной линии порта ввода/вывода микроконтроллера ATmega16, не влияя на состояние остальных линий этого порта?

4.15 Как выполняется проверка состояния разрядов регистра на наличие лог. 0 (лог. 1)?

4.16 Как выполняется ожидание появления лог. 0 (лог. 1) в разрядах требуемого регистра?

4.17 В чем заключается эффект дребезга контактов?

ЛАБОРАТОРНАЯ РАБОТА №3

ИЗУЧЕНИЕ СИСТЕМЫ ПРЕРЫВАНИЙ НА ОСНОВЕ 8- И 16-РАЗРЯДНЫХ ТАЙМЕРОВ/СЧЕТЧИКОВ МИКРОКОНТРОЛЛЕРА ATMEGA16

Цель работы: изучить принципы работы системы прерываний для управления устройствами встроенной периферии микроконтроллера ATmega16 на примере использования 8- и 16-разрядных таймеров/счетчиков в двух режимах работы.

1 Теоретические сведения

1.1 Система прерываний микроконтроллера ATmega16

Система прерываний (Interrupts) семейства микроконтроллеров AVR представляет собой инструмент для управления взаимодействием ЦП МК с устройствами, как встроенной периферии, так и внешней подключаемой периферии. К встроенной периферии МК относятся таймеры/счетчики, аналоговый компаратор, аналого-цифровой преобразователь (АЦП), электрически стираемое перепрограммируемое постоянное запоминающее устройство (ЭСППЗУ) и последовательные интерфейсы различных типов. К внешней подключаемой периферии относятся светодиодные, семисегментные и жидкокристаллические индикаторы, тактовые кнопки, матричные клавиатуры, различные датчики, другие микроконтроллеры и т. д.

Прерывание – это аппаратно-программный процесс, возникающий в МК в результате наступления какого-либо события, которое связано с работой встроенной либо внешней периферии. Во время прерывания выполнение текущей последовательности команд основной программы приостанавливается, и управление передается процедуре обработки прерывания (Interrupt Service Routine (ISR)), соответствующей наступившему событию. После выполнения данной процедуры, выполнение основной программы продолжается с той же самой команды, где оно было прервано.

Преимущество использования прерываний заключается в том, что устройства периферии работают в автономном режиме и не загружают ЦП микроконтроллера своими задачами, например, ожиданием приема данных по какому-либо интерфейсу. Автономная работа осуществляется до тех пор, пока не станет нужным выполнить определенное действие периферийным устройством, для которого уже обязательно требуется использовать ЦП. Например, в составе встроенной периферии МК имеется последовательный интерфейс USART, который в асинхронном режиме использует две линии связи: на передачу и на прием данных. Пока данные по линии приема не поступают, интерфейс находится в состоянии ожидания и никак не задействует ЦП МК, т. е. не отвлекает его от выполнения команд основной программы. Но как только будут

приняты данные для обработки, наступает событие «Завершение приема по интерфейсу USART». В этом случае модуль интерфейса формирует сигнал запроса на обработку прерывания (Interrupt ReQuest (IRQ)). Если прерывания глобально и локально разрешены, то запрос не будет отклонен и принятые данные по интерфейсу USART будут обработаны с помощью команд соответствующей процедуры обработки прерывания. После окончания этой процедуры управление передается основной программе, а интерфейс USART возвращается в исходное состояние ожидания новых данных, никак не загружая ЦП.

Таким образом, процедура обработки прерывания (обработчик прерывания) – это последовательность команд, которую следует выполнить при возникновении определенного события (таблица 3.1). Отличие этой процедуры от обычных функций состоит в том, что вместо стандартной машинной команды возврата из функции используется специальная машинная команда возврата из прерывания. Для правильной работы команд возврата должен быть обязательно инициализирован указатель стека.

Чтобы любое прерывание было реализовано, должна быть записана логическая единица (лог. 1) в два флага разрешения прерываний: глобальный (общий) и локальный. Флаг глобального разрешения прерываний I (Global Interrupt Enable) находится в самом старшем разряде регистра состояния (флагов) SREG (Status Register). В программе на языке программирования C/C++ для установки флага I в лог. 1 используется макрос *sei()*, а для сброса в логический нуль (лог. 0) – макрос *cli()*, которые будут доступны в тексте программы после подключения заголовочного файла <*avr/interrupt.h*>.

Флаги локального разрешения прерываний находятся в соответствующих конфигурационных регистрах микроконтроллера для каждого конкретного устройства используемой периферии. Даже если локальный флаг установлен, то прерывание не будет обработано, пока не будет установлен глобальный флаг разрешения.

Для МК ATmega16 характерно 21 прерывание. За каждым прерыванием закреплен свой вектор прерывания. Все векторы прерываний, располагаются в самом начале flash-памяти и вместе формируют таблицу векторов прерываний, которая представлена в таблице 3.1.

Каждый вектор прерывания характеризуется следующими параметрами:

- адрес flash-памяти, по которому располагается 4-байтовая машинная команда вызова процедуры обработки прерывания;

- имя (идентификатор), которое используется для определения процедуры обработки прерывания в программе на языке C/C++. Список идентификаторов находится в заголовочном файле используемой модели микроконтроллера, например, для ATmega16 – *iom16.h*, для ATmega8 – *iom8.h*;

- приоритет, который означает, что если два или более прерывания возникают одновременно, то первым будет обработано то, у которого приоритет выше, т. е. номер вектора прерывания меньше. Флаг глобального разрешения прерываний I аппаратно сбрасывается в лог. 0 после вызова процедуры обработки прерывания и устанавливается обратно в лог. 1 после выхода из этой процеду-

ры. Для реализации вложенных прерываний необходимо вначале обработчика прерывания использовать макрос *sei()*.

Таблица 3.1 – Векторы прерываний микроконтроллера ATmega16

Номер вектора	Адрес	Имя вектора прерывания	Описание события, вызывающего прерывание
1	0x000	–	Сброс по выводу $\overline{\text{RESET}}$ и сторожевому таймеру
2	0x002	INT0_vect	Внешнее прерывание 0
3	0x004	INT1_vect	Внешнее прерывание 1
4	0x006	TIMER2_COMP_vect	Совпадение таймера/счетчика TC2
5	0x008	TIMER2_OVF_vect	Переполнение таймера/счетчика TC2
6	0x00A	TIMER1_CAPT_vect	Захват таймера/счетчика TC1
7	0x00C	TIMER1_COMPA_vect	Совпадение A таймера/счетчика TC1
8	0x00E	TIMER1_COMPB_vect	Совпадение B таймера/счетчика TC1
9	0x010	TIMER1_OVF_vect	Переполнение таймера/счетчика TC1
10	0x012	TIMER0_OVF_vect	Переполнение таймера/счетчика TC0
11	0x014	SPI_STC_vect	Завершение передачи по интерфейсу SPI
12	0x016	USART_RXC_vect	Завершение приема по интерфейсу USART
13	0x018	USART_UDRE_vect	Регистр данных интерфейса USART пуст
14	0x01A	USART_TXC_vect	Завершение передачи по USART
15	0x01C	ADC_vect	Завершение АЦП преобразования
16	0x01E	EE_RDY_vect	Готовность ЭСППЗУ (EEPROM)
17	0x020	ANA_COMP_vect	Срабатывание аналогового компаратора
18	0x022	TWI_vect	Прерывание от интерфейса TWI
19	0x024	INT2_vect	Внешнее прерывание 2
20	0x026	TIMER0_COMP_vect	Совпадение таймера/счетчика TC0
21	0x028	SPM_RDY_vect	Готовность загрузки в память программ

Процедура обработки прерывания на языке C/C++ реализуется через макрос, который определен в заголовочном файле *<avr/interrupt.h>*. Формат записи процедуры:

```
ISR (Имя_вектора_прерывания)
{ Оператор1; }
```

Чтобы исключить непреднамеренное выполнение векторов прерываний, по нулевому адресу (0x0000) памяти программ находится 4-байтовая машинная команда перехода (вектор сброса) к инициализационной части исполняемой прикладной программы. Положение таблицы векторов прерываний может быть изменено. Они могут размещаться не только в начале памяти программ, но и в области загрузчика. В этом случае прикладная программа может начинаться с нулевой ячейки памяти. Для изменения размещения таблицы векторов прерываний используются два разряда IVCE и IVSEL в общем регистре управления прерываниями GICR (General Interrupt Control Register).

1.2 Таймеры/счетчики микроконтроллера ATmega16

Таймеры/счетчики (Timer/Counters) можно использовать для формирования временных интервалов, подсчета импульсов различных внешних сигналов, поступающих на выводы микроконтроллера, а также для формирования сигналов с периодической последовательностью прямоугольных импульсов, тактирования приемопередатчика последовательных интерфейсов. В режиме широтно-импульсной модуляции (ШИМ, Pulse-Width Modulation (PWM)) таймер/счетчик может использоваться для формирования сигнала с программно-изменяемой частотой и скважностью. Таймеры/счетчики способны генерировать сигналы запросов на обработку прерываний по событиям, переключая ЦП МК на их обслуживание и освобождая ЦП от необходимости периодического опроса состояния таймеров/счетчиков. Так как микроконтроллеры находят широкое применение в системах реального времени, то таймеры/счетчики являются одним из наиболее важных устройств встроенной в МК периферии.

В МК ATmega16 есть три таймера/счетчика – два 8-разрядных Timer/Counter 0 (TC0) и Timer/Counter 2 (TC2) и один 16-разрядный Timer/Counter 1 (TC1). Каждый из таймеров/счетчиков обладает своим определенным набором функций, сравнение которых представлено в таблице 3.2.

Таблица 3.2 – Функции таймеров/счетчиков микроконтроллера ATmega16

Функция	Таймер/счетчик TC0	Таймер/счетчик TC1	Таймер/счетчик TC2
Счетчик тактовых импульсов	Да (8-разрядный)	Да (16-разрядный)	Да (8-разрядный)
Счетчик внешних событий	Да	Да	Нет
Число каналов ШИМ	1	2	1
Часы реального времени (асинхронный режим работы)	Нет	Нет	Да
Режим захвата	Нет	Да	Нет

Каждый таймер/счетчик использует один или более выводов микроконтроллера. Как правило, эти выводы – это линии портов ввода/вывода, а функции, реализуемые этими выводами при работе с таймерами/счетчиками, являются их альтернативными функциями. Выводы микроконтроллера ATmega16, используемые таймерами/счетчиками, представлены в таблице 3.3.

Для управления 8-разрядными таймерами/счетчиками МК ATmega16 используются отдельные регистры ввода/вывода (PBB), где n – это номер таймера/счетчика (0 или 2):

- TCNTn (Timer/Counter Register) – счетный регистр TCn;
- OCRn (Output Compare Register) – регистр сравнения TCn;
- TCCRn (Timer/Counter Control Register) – регистр управления TCn;
- ASSR (Asynchronous Status Register) – регистр состояния асинхронного режима таймера/счетчика TC2.

Таблица 3.3 – Выводы МК, используемые таймерами/счетчиками

Обозначение вывода		Описание вывода
как линия порта ввода/вывода	как функция таймера/счетчика	
PB0	T0	Вход внешнего тактового сигнала таймера/счетчика TC0
PB3	OC0	Выход генератора ШИМ-сигнала таймера/счетчика TC0
PB1	T1	Вход внешнего тактового сигнала таймера/счетчика TC1
PD6	ICP1	Вход блока захвата таймера/счетчика TC1
PD5	OC1A	Выход A генератора ШИМ-сигнала таймера/счетчика TC1
PD4	OC1B	Выход B генератора ШИМ-сигнала таймера/счетчика TC1
PD7	OC2	Выход генератора ШИМ-сигнала таймера/счетчика TC2
PC6	TOSC1	Вывод для подключения внешнего резонатора к таймеру/счетчику TC2
PC7	TOSC2	Вывод для подключения внешнего резонатора к таймеру/счетчику TC2

Для управления 16-разрядным таймером/счетчиком TC1 МК ATmega16 используются 16-разрядные регистры ввода/вывода:

- TCNT1 (Timer/Counter 1 Register) – счетный регистр таймера/счетчика TC1, который физически размещается в двух PBB: TCNT1H и TCNT1L, отвечающих соответственно за старший (High) и младший (Low) байт значения счетного регистра;

- OCR1A (Output Compare Register 1 A) – регистр сравнения A таймера/счетчика TC1, который физически размещается в двух PBB: OCR1AH и OCR1AL;

- OCR1B (Output Compare Register 1 B) – регистр сравнения B таймера/счетчика TC1, который физически размещается в двух PBB: OCR1BH и OCR1BL;

- ICR1 (Input Capture Register 1) – регистр захвата таймера/счетчика TC1, который физически размещается в двух PBB: ICR1H и ICR1L.

Также используются следующие 8-разрядные регистры ввода/вывода:

- TCCR1A (Timer/Counter 1 Control Register A) – регистр управления A таймера/счетчика TC1;

- TCCR1B (Timer/Counter 1 Control Register B) – регистр управления B таймера/счетчика TC1.

Для управления всеми тремя таймерами/счетчиками используются три общих регистра ввода/вывода:

- TIMSK (Timer/Counter Interrupt Mask Register) – регистр масок прерываний таймеров/счетчиков;

- TIFR (Timer/Counter Interrupt Flag Register) – регистр флагов прерываний таймеров/счетчиков;

- SFIOR (Special Function Input/Output Register) – регистр специальных функций ввода/вывода.

Все таймеры/счетчики МК ATmega16 могут работать с использованием своего тактового сигнала f_{TC0}, f_{TC1}, f_{TC2}, который может совпадать с тактовой частотой микроконтроллера и может отличаться благодаря использованию блоков предделителей.

8-разрядный таймер/счетчик TC0 и 16-разрядный таймер/счетчик TC1 используют один и тот же 10-разрядный предделитель, а 8-разрядный таймер/счетчик TC2 использует свой отдельный предделитель. Для сброса настроек предделителей необходимо записать лог. 1 в разряды PSR2 и PSR10 регистра SFIOR.

Таймеры/счетчики TC0 и TC1 могут тактироваться от внешнего сигнала, поступающего на входы микроконтроллера T0 и T1. При этом частота внешнего сигнала должна быть в 2,5 раза ниже частоты тактового сигнала микроконтроллера.

Для работы таймера/счетчика TC2 в асинхронном режиме необходимо использовать внешний кварцевый резонатор часовой частоты 32,768 кГц, который подключается к выводам TOSC1 и TOSC2 микроконтроллера. При этом тактовая частота микроконтроллера должна быть как минимум в четыре раза больше.

1.2.1 8-разрядный таймер/счетчик TC0

Счетный регистр TCNT0 является программируемым двунаправленным (реверсивным) 8-разрядным счетчиком (рисунок 3.1). Регистр доступен для чтения и записи в любой момент времени. В зависимости от выбранного режима работы содержимое счетного регистра инкрементируется, декрементируется или сбрасывается по каждому импульсу тактового сигнала f_{TC0}. Когда на вход таймера/счетчика TC0 начинают поступать импульсы тактового сигнала, то он включается в режим счета. После прихода каждого тактового импульса содержимое счетного регистра увеличивается на единицу. При переполнении регистра его содержимое обнуляется (счет начинается сначала), и устанавливается флаг прерывания TOV0 в регистре TIFR и генерируется прерывание (если оно разрешено) по событию «Переполнение таймера/счетчика TC0» (см. вектор 10 в таблице 3.1). Если источник тактирования будет отключен, то TC0 останавливается. Однако состояние регистра TCNT0 доступно ЦП независимо от того, работает тактирование таймера/счетчика или нет. Запись в счетный регистр перекрывает любые действия самого счетчика: сброс или счет и блокирует работу блока сравнения на время одного периода тактового сигнала f_{TC0}, т. е. имеет более высокий приоритет.

Номер разряда	7	6	5	4	3	2	1	0
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.1 – Структура счетного регистра TCNT0

Регистр сравнения OCR0 является программируемым 8-разрядным регистром (рисунок 3.2). Регистр доступен для чтения и для записи в любой момент времени.

Во время работы таймера/счетчика производится непрерывное (в каждом периоде тактового сигнала (такте) f_{TC0}) сравнение этого регистра с регистром TCNT0. В случае равенства содержимого этих регистров в следующем такте устанавливается флаг прерывания OCF0 в регистре TIFR и генерируется прерывание (если оно разрешено) по событию «Совпадение таймера/счетчика TC0» (см. вектор 20 в таблице 3.1). Кроме того, при наступлении этого события может изменяться состояние вывода OC0 (Timer/Counter 0 Output Compare Match) микроконтроллера. Чтобы таймер/счетчик TC0 мог управлять состоянием этого вывода, третья линия порта B должна быть сконфигурирована на выход (запись лог. 1 в третий разряд регистра DDRB). Также следует отметить, что любая запись в счетный регистр блокирует формирование сигнала о совпадении, если оно произойдет в следующем такте.

Номер разряда	7	6	5	4	3	2	1	0
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.2 – Структура регистра сравнения OCR0

Регистр масок прерываний таймеров/счетчиков TIMSK содержит локальные флаги разрешения прерываний всех таймеров/счетчиков ATmega16. Как уже выше было отмечено, таймер/счетчик TC0 может генерировать прерывания при переполнении счетного регистра и при совпадении счетного регистра и регистра сравнения. Прерывания будут обработаны, только если дополнительно установлен в лог. 1 флаг глобального разрешения прерываний I в регистре SREG. Для таймера/счетчика TC0 в регистре TIMSK отведены только два младших разряда OCIE0 и TOIE0 (рисунок 3.3 и таблица 3.4). Разряды, которые относятся к другим таймерам/счетчикам, выделены серым фоном и будут рассмотрены далее.

Номер разряда	7	6	5	4	3	2	1	0
Имя разряда	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.3 – Флаги разрешения прерываний TC0 в регистре TIMSK

Таблица 3.4 – Флаги разрешения прерываний таймера/счетчика TC0

Номер разряда	Имя разряда	Описание
1	OCIE0	Timer/Counter 0 Output Compare Match Interrupt Enable. **Флаг разрешения прерывания по событию «Совпадение таймера/счетчика TC0».** Запись в разряд лог. 0 запрещает прерывание, а лог. 1 – локально разрешает
0	TOIE0	Timer/Counter 0 Overflow Interrupt Enable. **Флаг разрешения прерывания по событию «Переполнение таймера/счетчика TC0».** Запись в разряд лог. 0 запрещает прерывание, а лог. 1 – локально разрешает

Регистр флагов прерываний таймеров/счетчиков TIFR содержит флаги прерываний для всех таймеров/счетчиков МК ATmega16, которые аппаратно устанавливаются в лог. 1 при возникновении событий. Для таймера/счетчика TC0 это следующие события: «Переполнение таймера/счетчика TC0», «Совпадение таймера/счетчика TC0». Если в эти моменты в регистрах TIMSK и SREG прерывания локально и глобально разрешены, то микроконтроллер вызовет соответствующий обработчик прерывания. Флаги прерываний автоматически сбрасываются в лог. 0 при запуске процедуры обработки прерывания. Для таймера/счетчика TC0 в регистре TIFR отведены два младших разряда OCF0 и TOV0 (рисунок 3.4 и таблица 3.5). Разряды, относящиеся к другим таймерам/счетчикам, выделены серым фоном и будут рассмотрены далее.

Номер разряда	7	6	5	4	3	2	1	0
Имя разряда	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.4 – Флаги прерываний таймера/счетчика TC0 в регистре TIFR

Таблица 3.5 – Флаги прерываний таймера/счетчика TC0

Номер разряда	Имя разряда	Описание
1	OCF0	Timer/Counter 0 Output Compare Flag. **Флаг прерывания по событию «Совпадение таймера/счетчика TC0».** Аппаратно устанавливается в лог. 1 при совпадении регистра TCNT0 с регистром OCR0
0	TOV0	Timer/Counter 0 Overflow Flag. **Флаг прерывания по событию «Переполнение таймера/счетчика TC0».** Аппаратно устанавливается в лог. 1 при переполнении счетного регистра TCNT0

Регистр управления TCCR0 позволяет настроить режимы работы таймера/счетчика TC0, выбрать источник тактирования и режимы управления со-

стоянием вывода OC0. Структура и описание всех разрядов регистра представлены на рисунке 3.5 и в таблице 3.6.

Номер разряда	7	6	5	4	3	2	1	0
Имя разряда	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Чтение/запись	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.5 – Структура регистра управления TCCR0

Таблица 3.6 – Разряды регистра управления TCCR0

Номер разряда	Имя разряда	Описание
7	FOC0	Force Output Compare. **Принудительное изменение состояния вывода OC0 (режимы Normal и CTC).** При записи лог. 1 в этот разряд состояние вывода OC0 изменяется в соответствии с установками разрядов COM01:COM00. Прерывание при этом не создается и сброс счетного регистра (в режиме CTC) не производится. В режимах Fast PWM и Phase Correct PWM этот разряд должен быть сброшен в лог. 0. При чтении разряда всегда возвращается лог. 0
3, 6	WGM01: WGM00	Waveform Generation Mode. **Режим работы таймера/счетчика TC0**. Все режимы представлены в таблице 3.7
5:4	COM01: COM00	Compare Match Output Mode. **Режим управления состоянием вывода OC0.** Эти разряды определяют состояние вывода OC0 при наступлении события «Совпадение таймера/счетчика TC0». Влияние содержимого этих разрядов на состояние вывода зависит от режима работы таймера/счетчика TC0
2:0	CS02: CS01: CS00	Clock Select. **Выбор источника тактирования (запуск и остановка таймера/счетчика).** Эти разряды определяют источник тактового сигнала для таймера/счетчика TC0. Варианты частот тактового сигнала представлены в таблице 3.8

Таблица 3.7 – Режимы работы таймера/счетчика TC0

Номер режима	WGM01	WGM00	Режим работы	Верхний предел счета	Условие обновления регистра сравнения OCR0
0	0	0	Normal (Нормальный)	0xFF	Сразу после записи в регистр
1	0	1	Phase Correct PWM (ШИМ с точной фазой)	0xFF	Достижение регистром TCNT0 верхнего предела счета
2	1	0	CTC (Clear Timer on Compare, Сброс при совпадении)	OCR0	Сразу после записи в регистр
3	1	1	Fast PWM (Быстрая ШИМ)	0xFF	Достижение регистром TCNT0 верхнего предела счета

Таблица 3.8 – Выбор источника тактирования таймера/счетчика TC0

Номер режима	CS02	CS01	CS00	Описание
0	0	0	0	Источника тактирования нет, таймер/счетчик TC0 остановлен
1	0	0	1	Тактовая частота микроконтроллера (без предделителя)
2	0	1	0	Тактовая частота микроконтроллера, деленная на 8
3	0	1	1	Тактовая частота микроконтроллера, деленная на 64
4	1	0	0	Тактовая частота микроконтроллера, деленная на 256
5	1	0	1	Тактовая частота микроконтроллера, деленная на 1024
6	1	1	0	Внешний источник на выводе T0. Срабатывание по спадающему фронту
7	1	1	1	Внешний источник на выводе T0. Срабатывание по нарастающему фронту

Рассмотрим первые два режима работы 8-разрядного таймера/счетчика TC0.

Режим работы Normal (Нормальный).

Это наиболее простой режим работы таймера/счетчика TC0. В этом режиме счетный регистр TCNT0 функционирует как обычный суммирующий счетчик – по каждому импульсу тактового сигнала f_{TC0} осуществляется его инкрементирование. При переходе через значение 0xFF возникает переполнение, и счет продолжается со значения 0x00. В том же такте сигнала f_{TC0}, в котором обнуляется регистр TCNT0, флаг прерывания по событию «Переполнение таймера/счетчика TC0» TOV0 устанавливается в лог. 1. При равенстве счетного регистра TCNT0 и регистра сравнения OCR0 флаг прерывания по событию «Совпадение таймера/счетчика TC0» OCF0 устанавливается в лог. 1.

Режим работы CTC (Clear Timer on Compare, Сброс при совпадении).

В этом режиме счетный регистр тоже функционирует как обычный суммирующий счетчик, инкрементирование которого осуществляется по каждому импульсу тактового сигнала f_{TC0}. Однако максимально возможное значение счетного регистра и, следовательно, разрешающая способность таймера/счетчика и генератора ШИМ-сигнала определяется регистром сравнения OCR0. После достижения значения, записанного в регистре сравнения, значение счетного регистра TCNT0 аппаратно сбрасывается, и счет продолжается со значения 0x00. В том же такте сигнала f_{TC0}, в котором обнуляется счетный регистр TCNT0, флаг прерывания по событию «Переполнение таймера/счетчика TC0» TOV0 устанавливается в лог. 1. При равенстве счетного регистра TCNT0 и регистра сравнения OCR0 флаг прерывания по событию «Совпадение таймера/счетчика TC0» OCF0 устанавливается в лог. 1.

Если значение, записанное в OCR0, меньше текущего значения TCNT0, то сброс счетчика по событию «Совпадение таймера/счетчика TC0» наступит только после перехода счетного регистра через максимальное значение 0xFF в исходное нулевое состояние 0x00 и достижения нового значения OCR0.

Все режимы работы 8-разрядного таймера/счетчика TC0 могут быть использованы для формирования ШИМ-сигналов и будут рассмотрены более подробно в следующей лабораторной работе.

1.2.2 16-разрядный таймер/счетчик TC1

Счетный регистр TCNT1 является программируемым двунаправленным (реверсивным) 16-разрядным счетчиком. Регистр доступен для чтения и для записи в любой момент времени и физически размещается в двух PBB: TCNT1H и TCNT1L (рисунок 3.6). В зависимости от выбранного режима работы содержимое счетного регистра инкрементируется, декрементируется или сбрасывается по каждому импульсу тактового сигнала f_{TC1}. Когда на вход таймера/счетчика TC1 начинают поступать импульсы тактового сигнала, то он включается в режим счета. После прихода каждого тактового импульса содержимое счетного регистра увеличивается на единицу. При переполнении регистра его содержимое обнуляется (счет начинается сначала), и устанавливается флаг прерывания TOV1 регистра TIFR и генерируется прерывание (если оно разрешено) по событию «Переполнение таймера/счетчика TC1» (см. вектор 9 в таблице 3.1). Если источник тактирования будет не задан, то TC1 останавливается. Однако состояние регистра TCNT1 доступно ЦП независимо от того работает тактирование таймера/счетчика или нет. Запись в счетный регистр перекрывает любые действия самого счетчика: сброс или счет и блокирует работу двух блоков сравнения на время одного периода тактового сигнала f_{TC1}.

Номер разряда	7	6	5	4	3	2	1	0
Регистр TCNT1H	TCNT1[15:8]							
Регистр TCNT1L	TCNT1[7:0]							
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.6 – Структура счетного регистра TCNT1

Регистры сравнения A и B (OCR1A и OCR1B) являются программируемыми 16-разрядными регистрами. Каждый из регистров доступен для чтения и для записи в любой момент времени и физически размещается в соответствующих двух PBB: для OCR1A – это OCR1AH и OCR1AL, для OCR1B – это OCR1BH и OCR1BL (рисунки 3.7, 3.8). Каждый из регистров OCR1A и OCR1B входит в состав отдельного блока сравнения. Во время работы таймера/счетчика производится непрерывное (в каждом периоде тактового сигнала (такте) f_{TC1}) сравнение этих регистров с регистром TCNT1.

В случае равенства содержимого счетного регистра TCNT1 и регистра сравнения OCR1A в следующем такте устанавливается флаг прерывания OCF1A в регистре TIFR и генерируется прерывание (если оно разрешено) по событию «Совпадение A таймера/счетчика TC1» (см. вектор 7 в таблице 3.1).

Кроме того, при наступлении этого события может изменяться состояние вывода OC1A (Timer/Counter 1 Output Compare A Match) микроконтроллера. Чтобы таймер/счетчик TC1 мог управлять состоянием этого вывода, пятая линия порта D должна быть сконфигурирована на выход.

В случае равенства содержимого счетного регистра TCNT1 и регистра сравнения OCR1B в следующем такте устанавливается флаг прерывания OCF1B в регистре TIFR и генерируется прерывание (если оно разрешено) по событию «Совпадение B таймера/счетчика TC1» (см. вектор 8 в таблице 3.1). Кроме того, при наступлении этого события может изменяться состояние вывода OC1B (Timer/Counter 1 Output Compare B Match) микроконтроллера. Чтобы таймер/счетчик TC1 мог управлять состоянием этого вывода, четвертая линия порта D должна быть сконфигурирована на выход.

Также следует отметить, что любая запись в счетный регистр TCNT1 блокирует формирование сигнала о совпадении, если оно произойдет в следующем такте.

Номер разряда	7	6	5	4	3	2	1	0
Регистр OCR1AH	OCR1A[15:8]							
Регистр OCR1AL	OCR1A[7:0]							
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.7 – Структура регистра сравнения OCR1A

Номер разряда	7	6	5	4	3	2	1	0
Регистр OCR1BH	OCR1B[15:8]							
Регистр OCR1BL	OCR1B[7:0]							
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.8 – Структура регистра сравнения OCR1B

В режимах работы таймера/счетчика TC1, предназначенных для формирования ШИМ-сигналов, оба блока сравнения обладают двойной буферизацией записи в регистры сравнения OCR1A и OCR1B. Она заключается в том, что записываемое число на самом деле сохраняется в специальном буферном регистре, а изменение содержимого регистра сравнения происходит только в момент достижения счетным регистром TCNT1 верхнего предела счета. Благодаря такому решению исключается появление помех – несимметричных импульсов сигнала (импульсов, длина которых равна нечетному количеству тактов) на выходах A и B ШИМ-генератора, которые были неизбежны при непосредственной записи в регистры сравнения.

Регистр захвата ICR1 является 16-разрядным и физически размещается в двух РВВ: ICR1H и ICR1L (рисунок 3.9). Регистр захвата входит в состав бло-

ка захвата, который запоминает значение счетного регистра TCNT1 при возникновении внешнего события, тем самым определяя время его возникновения. Источником такого события может выступать либо активный фронт сигнала на выводе ICP1 микроконтроллера, либо сигнал от аналогового компаратора. Одновременно с записью в регистр захвата устанавливается флаг прерывания ICF1 в регистре TIFR и генерируется прерывание (если оно разрешено) по событию «Захват таймера/счетчика TC1» (см. вектор 6 в таблице 3.1). Программно запись в регистр ICR1 возможна только в режимах, в которых регистр захвата определяет верхний предел счета таймера/счетчика TC1. Соответственно вывод ICP1 в этих режимах отключен, а функция захвата недоступна.

Номер разряда	7	6	5	4	3	2	1	0
Регистр ICR1H	ICR1[15:8]							
Регистр ICR1L	ICR1[7:0]							
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.9 – Структура регистра захвата ICR1

Чтобы таймер/счетчик TC1 мог выполнить захват по сигналу с вывода ICP1, 6-я линия порта D должна быть сконфигурирована на вход (запись лог. 0 в 6-й разряд регистра DDRD).

Доступ к 16-разрядным регистрам таймера/счетчика TC1 осуществляется через 8-разрядную шину данных, поэтому для обращения к ним требуется выполнить по две операции чтения или записи. Для того чтобы запись или чтение обоих байтов содержимого 16-разрядного регистра происходили одновременно, в составе таймера/счетчика имеется 8-разрядный регистр для временного хранения старшего байта значения (этот регистр используется только ЦП и программно недоступен).

Для выполнения записи 16-разрядного регистра первым должен быть загружен старший байт значения, который помещается во временный регистр. При последующей записи младшего байта он объединяется с содержимым временного регистра, и оба байта одновременно (в одном и том же такте) записываются в 16-разрядный регистр. Если требуется изменить несколько 16-разрядных регистров таймера/счетчика TC1, а старшие байты всех записываемых значений одинаковы, то загрузку старшего байта достаточно выполнить только один раз.

Для выполнения чтения 16-разрядного регистра первым должен быть прочитан младший байт. При его чтении содержимое старшего байта помещается во временный регистр. При последующем чтении старшего байта возвращается значение, сохраненное во временный регистр. Исключение составляют только регистры сравнения OCR1A и OCR1B, при чтении которых временный регистр не используется.

Таким образом, для обращения ко всем 16-разрядным регистрам TC1 используется один и тот же временный регистр. Чтобы записать значение в 16-разрядный регистр, необходимо сначала записать старший байт, а затем младший. А при чтении 16-разрядного регистра, наоборот, сначала считывается младший байт, а затем старший. При написании программы на языке программирования C/C++ в среде разработки Atmel Studio запись или чтение 16-разрядных регистров таймера/счетчика TC1 **не нужно** разделять на две отдельные команды, т. к. компилятор выполнит это в автоматическом режиме. На примере регистра TCNT1 запись и чтение выполняется следующим образом:

```
1:    unsigned int temp; //Объявление переменной temp
2:    ...
3:    TCNT1=0x01F9; //Запись значения в счетный регистр TCNT1
4:    temp=TCNT1; //Чтение регистра TCNT1 в переменную temp
5:    ...
```

При выполнении обращения к 16-разрядным регистрам таймера/счетчика TC1 прерывания должны быть запрещены. В противном случае, если прерывание произойдет между двумя командами обращения к 16-разрядному регистру, а в процедуре обработки прерывания тоже будет произведено обращение к какому-либо из 16-разрядных регистров таймера/счетчика, содержимое временного регистра будет изменено. Как следствие, результат обращения к 16-разрядному регистру в основной программе будет неверным.

Ниже представлен пример реализации функции в среде разработки Atmel Studio для записи значения в 16-разрядный регистр TCNT1 без опасности изменения содержимого временного регистра в прерываниях:

```
1:    #include <avr/interrupt.h> //Для использования макроса cli()
2:    #include <avr/common.h> //Для использования макроса SREG
3:    void TC1_writeTCNT1(unsigned int temp)
4:    {
5:        unsigned char sreg; //Объявление переменной sreg для сохране-
      ния значения регистра флагов SREG
6:        sreg=SREG; //Состояние флага прерываний I сохранено в sreg
7:        cli(); //Прерывания глобально запрещены
8:        TCNT1=temp; //Запись аргумента функции temp в регистр TCNT1
9:        SREG=sreg; //Состояние флага прерываний I восстановлено
10:   }
```

Ниже представлен пример реализации функции в среде разработки Atmel Studio для чтения значения из 16-разрядного регистра TCNT1 без опасности изменения содержимого временного регистра в прерываниях:

```
1:    #include <avr/interrupt.h> //Для использования макроса cli()
2:    #include <avr/common.h> //Для использования макроса SREG
3:    unsigned int TC1_readTCNT1(void)
4:    {
5:         unsigned char sreg; //Объявление переменной sreg для сохране-
      ния значения регистра флагов SREG
6:         unsigned int temp; //Объявление переменной temp
7:         sreg=SREG;  //Состояние флага прерываний I сохранено в sreg
8:         cli(); //Прерывания глобально запрещены
9:         temp=TCNT1; //Чтение регистра TCNT1 в переменную temp
10:        SREG=sreg; //Состояние флага прерываний I восстановлено
11:        return temp; //Возврат функцией считанного значения
12:   }
```

Представленные выше примеры следует использовать аналогичным образом и при выполнении обращения к регистрам OCR1A, OCR1B и ICR1.

Регистр масок прерываний таймеров/счетчиков TIMSK содержит также и локальные флаги разрешения прерываний таймера/счетчика TC1 МК ATmega16. Прерывания будут обработаны, только если дополнительно установлен флаг глобального разрешения прерываний I в регистре SREG. Для таймера/счетчика TC1 в регистре TIMSK отведены четыре разряда TICIE1, OCIE1A, OCIE1B и TOIE1 (рисунок 3.10 и таблица 3.9). Разряды, которые относятся к другим таймерам/счетчикам, выделены серым фоном.

Номер разряда	7	6	5	4	3	2	1	0
Имя разряда	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.10 – Флаги разрешения прерываний TC1 в регистре TIMSK

Регистр флагов прерываний таймеров/счетчиков TIFR содержит также и флаги прерываний таймера/счетчика TC1 МК ATmega16, которые аппаратно устанавливаются в лог. 1 при возникновении событий. Если в моменты этих событий в регистрах TIMSK и SREG прерывания локально и глобально разрешены, то микроконтроллер вызовет соответствующий обработчик прерывания. Флаги прерываний автоматически сбрасываются в лог. 0 при запуске процедуры обработки прерывания. Для таймера/счетчика TC1 в регистре TIFR отведены четыре разряда ICF1, OCF1A, OCF1B и TOV1 (рисунок 3.11 и таблица 3.10). Разряды, относящиеся к другим таймерам/счетчикам, выделены серым фоном.

Таблица 3.9 – Флаги разрешения прерываний таймера/счетчика TC1

Номер разряда	Имя разряда	Описание
5	TICIE1	Timer/Counter 1 Input Capture Interrupt Enable. **Флаг разрешения прерывания по событию «Захват таймера/счетчика TC1».** Запись в разряд лог. 0 запрещает прерывание, а лог. 1 – локально разрешает
4	OCIE1A	Timer/Counter 1 Output Compare A Match Interrupt Enable. **Флаг разрешения прерывания по событию «Совпадение А таймера/счетчика TC1».** Запись в разряд лог. 0 запрещает прерывание, а лог. 1 – локально разрешает
3	OCIE1B	Timer/Counter 1 Output Compare B Match Interrupt Enable. **Флаг разрешения прерывания по событию «Совпадение В таймера/счетчика TC1».** Запись в разряд лог. 0 запрещает прерывание, а лог. 1 – локально разрешает
2	TOIE1	Timer/Counter 1 Overflow Interrupt Enable. **Флаг разрешения прерывания по событию «Переполнение таймера/счетчика TC1».** Запись в разряд лог. 0 запрещает прерывание, а лог. 1 – локально разрешает

Номер разряда	7	6	5	4	3	2	1	0
Имя разряда	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.11 – Флаги прерываний таймера/счетчика TC1 в регистре TIFR

Таблица 3.10 – Флаги прерываний таймера/счетчика TC1

Номер разряда	Имя разряда	Описание
5	ICF1	Timer/Counter 1 Input Capture Flag. **Флаг прерывания по событию «Захват таймера/счетчика TC1».** Аппаратно устанавливается в лог. 1 при записи значения в регистр захвата ICR1
4	OCF1A	Timer/Counter 1 Output Compare A Match Flag. **Флаг прерывания по событию «Совпадение А таймера/счетчика TC1».** Аппаратно устанавливается в лог. 1 при совпадении регистра TCNT1 с регистром OCR1A
3	OCF1B	Timer/Counter 1 Output Compare B Match Flag. **Флаг прерывания по событию «Совпадение В таймера/счетчика TC1».** Аппаратно устанавливается в лог. 1 при совпадении регистра TCNT1 с регистром OCR1B
2	TOV1	Timer/Counter 1 Overflow Flag. **Флаг прерывания по событию «Переполнение таймера/счетчика TC1».** Аппаратно устанавливается в лог. 1 при переполнении регистра TCNT1

Для управления 16-разрядным таймером/счетчиком TC1 используются следующие регистры:

- **регистр управления A (TCCR1A)**, структура и описание разрядов которого представлены на рисунке 3.12 и в таблице 3.11;

- **регистр управления B (TCCR1B)**, структура и описание разрядов которого представлены на рисунке 3.13 и в таблице 3.12.

Номер разряда	7	6	5	4	3	2	1	0
Имя разряда	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
Чтение/запись	R/W	R/W	R/W	R/W	W	W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.12 – Структура регистра управления TCCR1A

Таблица 3.11 – Разряды регистра управления TCCR1A

Номер разряда	Имя разряда	Описание
7:6	COM1A1: COM1A0	Compare Output Mode for Channel A. **Режим управления состоянием вывода OC1A.** Эти разряды определяют состояние вывода OC1A при наступлении события «Совпадение A таймера/счетчика TC1». Влияние содержимого этих разрядов на состояние вывода зависит от режима работы TC1
5:4	COM1B1: COM1B0	Compare Output Mode for Channel B. **Режим управления состоянием вывода OC1B.** Эти разряды определяют состояние вывода OC1B при наступлении события «Совпадение B таймера/счетчика TC1». Влияние содержимого этих разрядов на состояние вывода зависит от режима работы TC1
3	FOC1A	Force Output Compare for Channel A. **Принудительное изменение состояния вывода OC1A (режимы Normal и CTC).** При записи лог. 1 в этот разряд состояние вывода OC1A изменяется в соответствии с установками разрядов COM1A1:COM1A0. Прерывание при этом не создается и сброс счетного регистра (в режиме CTC) не производится. В режимах формирования ШИМ-сигнала эта функция недоступна. При чтении разряда всегда возвращается лог. 0
2	FOC1B	Force Output Compare for Channel B. **Принудительное изменение состояния вывода OC1B (режимы Normal и CTC).** При записи лог. 1 в этот разряд состояние вывода OC1B изменяется в соответствии с установками разрядов COM1B1:COM1B0. Прерывание при этом не создается и сброс счетного регистра (в режиме CTC) не производится. В режимах формирования ШИМ-сигнала эта функция недоступна. При чтении разряда всегда возвращается лог. 0
1:0	WGM11: WGM10	Waveform Generation Mode. **Режим работы таймера/счетчика TC1.** Совместно с разрядами WGM13:WGM12 из регистра TCCR1B определяют режим работы таймера/счетчика TC1. Все режимы представлены в таблице 3.13

Номер разряда	7	6	5	4	3	2	1	0
Имя разряда	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
Чтение/запись	R/W	R/W	R	R/W	W	W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.13 – Структура регистра управления TCCR1B

Таблица 3.12 – Разряды регистра управления TCCR1B

Номер разряда	Имя разряда	Описание
7	ICNC1	Input Capture Noise Canceler. **Подавитель шума на входе блока захвата.** Запись лог. 0 в этот разряд выключает подавитель шума, и захват производится по первому активному фронту сигнала на выводе ICP1 либо от аналогового компаратора. Запись лог. 1 включает подавитель шума, тогда при появлении активного фронта производится четыре выборки с частотой, равной тактовой частоте микроконтроллера. Захват будет выполнен только в том случае, если все выборки имеют уровень, соответствующий активному фронту сигнала (лог. 1 для нарастающего фронта и лог. 0 для спадающего фронта)
6	ICES1	Input Capture Edge Select. **Выбор активного фронта сигнала.** Если разряд сброшен в лог. 0, сохранение счетного регистра TCNT1 в регистре захвата ICR1 осуществляется по спадающему фронту сигнала. Если разряд установлен в лог. 1, то сохранение осуществляется по нарастающему фронту сигнала. Если регистр ICR1 используется для хранения значения верхнего предела счета (таблица 3.13), то вход ICP1 отключается от соответствующего вывода микроконтроллера и функция захвата блокируется
5	–	Не используется, считывается как лог. 0
4:3	WGM13: WGM12	Waveform Generation Mode. **Режим работы таймера/счетчика TC1.** Совместно с разрядами WGM11:WGM10 из регистра TCCR1A определяют режим работы таймера/счетчика TC1. Все режимы представлены в таблице 3.13
2:0	CS12: CS11: CS10	Clock Select. **Выбор источника тактирования (запуск и остановка таймера/счетчика).** Эти разряды определяют источник тактового сигнала для таймера/счетчика TC0. Варианты частот тактового сигнала представлены в таблице 3.14

Таблица 3.13 – Режимы работы таймера/счетчика TC1

Номер режима	WGM13	WGM12	WGM11	WGM10	Название режима работы	Верхний предел счета (TOP)	Условие обновления регистра сравнения OCR1A (OCR1B)
0	0	0	0	0	Normal	0xFFFF	Сразу после записи
1	0	0	0	1	Phase correct PWM, 8-разрядный	0x00FF	При достижении TOP
2	0	0	1	0	Phase correct PWM, 9-разрядный	0x01FF	При достижении TOP
3	0	0	1	1	Phase correct PWM, 10-разрядный	0x03FF	При достижении TOP
4	0	1	0	0	CTC (Clear Timer on Compare)	OCR1A	Сразу после записи
5	0	1	0	1	Fast PWM, 8-разрядный	0x00FF	При достижении TOP
6	0	1	1	0	Fast PWM, 9-разрядный	0x01FF	При достижении TOP
7	0	1	1	1	Fast PWM, 10-разрядный	0x03FF	При достижении TOP
8	1	0	0	0	Phase and Frequency Correct PWM	ICR1	На нижнем пределе счета
9	1	0	0	1	Phase and Frequency Correct PWM	OCR1A	На нижнем пределе счета
10	1	0	1	0	Phase correct PWM	ICR1	При достижении TOP
11	1	0	1	1	Phase correct PWM	OCR1A	При достижении TOP
12	1	1	0	0	CTC (Clear Timer on Compare)	ICR1	Сразу после записи
13	1	1	0	1	Зарезервировано (не используется)	–	–
14	1	1	1	0	Fast PWM	ICR1	При достижении TOP
15	1	1	1	1	Fast PWM	OCR1A	При достижении TOP

Таблица 3.14 – Выбор источника тактирования таймера/счетчика TC1

Номер режима	CS12	CS11	CS10	Описание
0	0	0	0	Источника тактирования нет, таймер/счетчик TC1 остановлен
1	0	0	1	Тактовая частота микроконтроллера (без предделителя)
2	0	1	0	Тактовая частота микроконтроллера, деленная на 8
3	0	1	1	Тактовая частота микроконтроллера, деленная на 64
4	1	0	0	Тактовая частота микроконтроллера, деленная на 256
5	1	0	1	Тактовая частота микроконтроллера, деленная на 1024
6	1	1	0	Внешний источник на выводе T1. Срабатывание по спадающему фронту
7	1	1	1	Внешний источник на выводе T1. Срабатывание по нарастающему фронту

Регистр специальных функций ввода/вывода SFIOR позволяет осуществлять сброс предделителя таймеров/счетчиков TC0 и TC1 (рисунок 3.14 и таблица 3.15). Разряды, которые относятся к другим устройствам встроенной периферии микроконтроллера, выделены серым фоном.

Номер разряда	7	6	5	4	3	2	1	0
Имя разряда	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10
Чтение/запись	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Рисунок 3.14 – Структура регистра SFIOR

Таблица 3.15 – Разряд регистра SFIOR, используемый таймерами/счетчиками

Номер разряда	Имя разряда	Описание
0	PSR10	Prescaler Reset Timer/Counter 1 and Timer/Counter 0. **Сброс предделителя таймеров/счетчиков TC0 и TC1.** При записи лог. 1 в этот разряд предделитель таймеров/счетчиков TC0 и TC1 переводится в исходное состояние, в котором таймеры/счетчики будут работать на тактовой частоте микроконтроллера, т. е. без предделителя (коэффициент деления равен единице). Этот разряд аппаратно сбрасывается в лог. 0 после выполнения операции сброса. Следует обратить внимание, что таймеры/счетчики TC0 и TC1 используют один и тот же предделитель, и сброс этого предделителя оказывает влияние на оба таймера/счетчика

Рассмотрим первые два режима работы 16-разрядного таймера/счетчика TC1.

Режим работы Normal (Нормальный).

Это наиболее простой режим работы таймера/счетчика TC1. В этом режиме счетный регистр TCNT1 функционирует как обычный суммирующий счетчик – по каждому импульсу тактового сигнала f_{TC1} осуществляется его инкрементирование. При переходе через значение 0xFFFF возникает переполнение, и счет продолжается со значения 0x0000. В том же такте сигнала f_{TC1}, в котором обнуляется регистр TCNT1, флаг прерывания по событию «Переполнение таймера/счетчика TC1» TOV1 устанавливается в лог. 1. При равенстве счетного регистра TCNT1 и регистра сравнения OCR1A (OCR1B) флаг прерывания по событию «Совпадение A (B) таймера/счетчика TC1» OCF1A (OCF1B) устанавливается в лог. 1.

Режим работы CTC (Clear Timer on Compare, Сброс при совпадении).

В этом режиме счетный регистр TCNT1 тоже функционирует как обычный суммирующий счетчик, инкрементирование которого осуществляется по каждому импульсу тактового сигнала f_{TC1}. Аппаратный сброс счетного регистра происходит только в тех случаях, когда его значение совпадает со значением регистра сравнения OCR1A (режим WGM13:0=4) или регистра захвата ICR1 (режим WGM13:0=12). Таким образом, значение регистра OCR1A (ICR1) определяет верхний предел счета и, следовательно, разрешающую способность таймера/счетчика и генератора ШИМ-сигнала.

В режимах 4 и 12:

- флаг прерывания по событию «Переполнение» TOV1 устанавливается в лог. 1 только при изменении значения счетного регистра с 0xFFFF на 0x0000;

- при равенстве счетного регистра и регистра сравнения OCR1A (OCR1B) флаг прерывания по событию «Совпадение A (B)» OCF1A (OCF1B) устанавливается в лог. 1.

Запись значения верхнего предела счета, близкого к значению нижнего предела счета (0x0000), когда счетчик работает без предделителя или с малым значением коэффициента деления, необходимо выполнять с особой осторожно-

стью, т. к. в режиме CTC нет двойной буферизации. Если значение, записанное в OCR1A (ICR1), меньше текущего значения TCNT1, то сброс счетчика наступит только после перехода счетного регистра через максимальное значение 0xFFFF в исходное нулевое состояние 0x0000 и достижения нового значения OCR1A (ICR1).

Все режимы работы 16-разрядного таймера/счетчика TC1 могут быть использованы для формирования ШИМ-сигналов и будут рассмотрены более подробно в следующей лабораторной работе.

1.2.3 Вызов обработчика прерывания по разным событиям через определенный интервал времени, формируемый таймерами/счетчиками

Как уже выше было отмечено, таймер/счетчик TC0 выполняет подсчет импульсов тактового сигнала f_{TC0} поступающего на его вход, в результате чего инкрементируется значение счетного регистра TCNT0. Поскольку таймер/счетчик TC0 является 8-разрядным, то максимальным значением TCNT0 является 255, но для его переполнения (перехода в начальное нулевое состояние) требуется выполнить на один отсчет больше, т. е. 255 + 1 отсчетов. Каждый отсчет выполняется за один период тактового сигнала T_{TCO}, следовательно, 255 + 1 отсчетов займут по времени 255 + 1 периодов:

$$t = (255 + 1)\cdot T_{TC0} = \frac{255 + 1}{f_{TC0}}. \qquad (3.1)$$

В соответствии с установками разрядов CS02:0 таймер/счетчик TC0 может тактироваться сигналом с частотой, равной тактовой частоте МК f_T, либо с уменьшенной частотой в результате использования блока предделителя, либо от внешнего сигнала, поступающего на вход T0.

В МК ATmega16 имеется внутренний источник тактовых импульсов – калиброванный RC-генератор, который позволяет формировать тактовый сигнал с частотами 1, 2, 4 и 8 МГц. Из формулы (3.1) видно, что величина времени, которая необходима таймеру/счетчику TC0 для обнуления счетного регистра, зависит от частоты, но без использования внешних источников тактовых импульсов величина времени будет наибольшей только на частоте $f_{TC0} = f_T = 1$ МГц и составит

$$t = \frac{255 + 1}{f_{TC0}} = \frac{255 + 1}{1 \cdot 10^6} = 256 \text{ (мкс)}. \qquad (3.2)$$

Для увеличения величины времени t можно использовать блок предделителя, который использует один из фиксированных коэффициентов деления $k_{дел}$. Коэффициент деления показывает, сколько периодов тактового сигнала f_T

должно пройти, прежде чем значение счетного регистра TCNT0 изменится на единицу. При использовании коэффициента деления частота тактового сигнала таймера/счетчика TC0 будет рассчитываться по формуле

$$f_{TC0} = \frac{f_T}{k_{дел}}. \quad (3.3)$$

Для 8-разрядного таймера/счетчика TC0 коэффициент деления $k_{дел}$ может принимать значения 1, 8, 64, 256, 1024. Но только при $k_{дел} = 1024$ и $f_T = 1$ МГц будет достигнута максимальная величина временного интервала, по истечению которого обнуляется счетный регистр TCNT0 и которая составит

$$\Delta t_{TC0} = \frac{255 + 1}{f_{TC0}} = \frac{(255 + 1)\cdot k_{дел}}{f_T} = \frac{256 \cdot 1024}{1 \cdot 10^6} = 262{,}144 \text{ (мс)}. \quad (3.4)$$

Для расчета временного интервала требуемой длительности необходимо формулу (3.4) переписать в следующем виде:

$$\Delta t_{TC0} = \frac{(N_{отсч} + 1)\cdot k_{дел}}{f_T}, \quad (3.5)$$

где $N_{отсч}$ – количество отсчетов, выполняемое таймером/счетчиком TC0 посредством инкрементирования значения счетного регистра TCNT0 в пределах 0–255.

Из формулы (3.5) можно выразить $N_{отсч}$:

$$N_{отсч} = \frac{\Delta t_{TC0} \cdot f_T}{k_{дел}} - 1. \quad (3.6)$$

Для вызова обработчика прерывания через определенный интервал времени в секундах, используя режим работы Normal и прерывание по событию «Переполнение таймера/счетчика TC0», при коэффициенте деления $k_{дел}$ и тактовой частоте f_T в герцах в счетный регистр TCNTn необходимо записать начальное значение, которое можно определить по формуле:

1) для 8-разрядного таймера/счетчика TC0:

$$TCNT0 = 255 - N_{отсч} = 256 - \frac{\Delta t_{TC0} \cdot f_T}{k_{дел}}; \quad (3.7)$$

2) для 16-разрядного таймера/счетчика TC1:

$$TCNT1 = 65535 - N_{отсч} = 65536 - \frac{\Delta t_{TC1} \cdot f_T}{k_{дел}}. \quad (3.8)$$

Для вызова обработчика прерывания через определенный интервал времени в секундах, используя режим работы Normal или CTC и прерывание по событию «Совпадение таймера/счетчика TC0», при коэффициенте деления $k_{дел}$ и тактовой частоте f_T в герцах в счетный регистр TCNT0 (TCNT1) необходимо записать нулевое значение 0x00 (0x0000), а в регистр сравнения OCR0 (OCR1A/OCR1B) – значение, определяемое по формуле

$$N_{отсч.OCR} = \frac{\Delta t_{TCn} \cdot f_T}{k_{дел}} - 1. \quad (3.9)$$

В формуле (3.9) вычитается единица, потому что вызов обработчика прерывания по событию «Совпадение» будет выполнен только в следующем периоде тактового сигнала таймера/счетчика после такта равенства значений счетного регистра и регистра сравнения.

Следует отметить, что формулы (3.7)–(3.9) не могут быть использованы для реализации нулевого временного интервала ($t_{TCn} = 0$), т. к. запись значений 256 и 65536 в регистры TCNT0 и TCNT1 приведет к переполнению и обнулению регистров, а запись «–1» – к переполнению и записи 255 (65535). В результате будут сформированы максимальные временные интервалы.

Регистры таймеров/счетчиков TC0 и TC1 могут содержать только **целочисленные значения**. Это означает, что дробная часть чисел, получаемых в формулах (3.7)–(3.9), будет отбрасываться, следовательно, временной интервал будет отсчитываться таймерами/счетчиками с погрешностью.

1.2.4 Запуск и остановка работы таймеров/счетчиков по тактовой кнопке

Все три таймера/счетчика микроконтроллера ATmega16 работают на тактовой частоте, величина которой задается разрядами CSn2:0 в соответствующем регистре управления (n – номер таймера/счетчика). В режиме CSn2:0=0 источник тактирования не подключен, и таймер/счетчик будет остановлен. Поэтому нажатие тактовой кнопки, которое замыкает цепь, должно приводить к подключению источника тактового сигнала с помощью записи требуемой ненулевой комбинации в разряды CSn2:0, а отжатие кнопки, которое размыкает цепь, должно приводить к отключению источника с помощью записи нулевой комбинации.

Ниже представлен пример реализации управления 8-разрядным таймером/счетчиком TC0, который работает на частоте, равной тактовой частоте микроконтроллера, с помощью тактовой кнопки, подключенной к нулевой линии порта A, на языке программирования C/C++ в среде разработки Atmel Studio:

```
1:    if ((PINA & (1<<PA0))==0)
2:    {
3:        TCCR0|=(0<<CS02)|(0<<CS01)|(1<<CS00);
4:    }
5:    else
6:    {
7:        TCCR0&=~((0<<CS02)|(0<<CS01)|(1<<CS00));
8:    }
```

Ниже представлен пример реализации управления 16-разрядным таймером/счетчиком TC1, который работает на частоте, равной тактовой частоте микроконтроллера, с помощью тактовой кнопки, подключенной к нулевой линии порта A, на языке программирования C/C++ в среде разработки Atmel Studio:

```
1:    if ((PINA & (1<<PA0))==0)
2:    {
3:        TCCR1B|=(0<<CS12)|(0<<CS11)|(1<<CS10);
4:    }
5:    else
6:    {
7:        TCCR1B&=~((0<<CS12)|(0<<CS11)|(1<<CS10));
8:    }
```

В обоих примерах предполагается, что используется тактовая кнопка с фиксацией своего состояния, которая с одной стороны подключена к нулевой линии порта A, а с другой – к выводу GND микроконтроллера, поэтому должна быть предварительно выполнена программная настройка этой линии порта A на вход (ввод данных) с подключением внутреннего подтягивающего резистора.

Если при работе тактовой кнопки возникает эффект дребезга контактов, то для его устранения необходимо использовать программный способ в виде функции задержки по времени *_delay_ms()*, которая будет доступна после подключения заголовочного файла *<util/delay.h>*.

1.3 Пример написания программ на C/C++

Требуется написать программу на языке программирования C/C++, выполнение которой микроконтроллером ATmega16 позволит реализовать мигание светодиода с частотой 5 Гц с помощью 8-разрядного таймера/счетчика TC0

и процедуры обработки прерывания. При этом необходимо предусмотреть выполнение следующих условий и требований:

1) схема устройства в программе Proteus ISIS представлена на рисунке 3.15;

2) микроконтроллер должен работать от внутреннего источника тактирования на тактовой частоте, которая позволит реализовать заданную частоту мигания светодиода;

3) интервалы времени, в течение которых светодиод будет включен и выключен, должны быть сформированы с использованием одного из режимов работы 8-разрядного таймера/счетчика TC0 и процедуры обработки прерывания по событию «Переполнение таймера/счетчика TC0» или «Совпадение таймера/счетчика TC0»;

4) проверить частоту мигания светодиода с помощью встроенного программного цифрового осциллографа OCS1 в программе Proteus ISIS.

Светодиод будет мигать с периодом повторения

$$T_{миг} = \frac{1}{f_{миг}} = \frac{1}{5} = 200 \text{ (мс)}. \qquad (3.10)$$

Одну половину периода светодиод будет включен, а вторую – выключен, следовательно, через каждые 100 мс таймер/счетчик TC0 должен создавать прерывание, в обработчике которого состояние третьей линии порта A будет меняться на противоположное, тем самым реализуя требуемую частоту мигания светодиода.

Осуществить поставленную задачу с помощью таймера/счетчика TC0 можно несколькими способами:

1) в режиме Normal с использованием прерывания по событию «Переполнение таймера/счетчика TC0»;

2) в режиме Normal с использованием прерывания по событию «Совпадение таймера/счетчика TC0»;

3) в режиме CTC с использованием прерывания по событию «Переполнение таймера/счетчика TC0»;

4) в режиме CTC с использованием прерывания по событию «Совпадение таймера/счетчика TC0».

Рассмотрим данные способы.

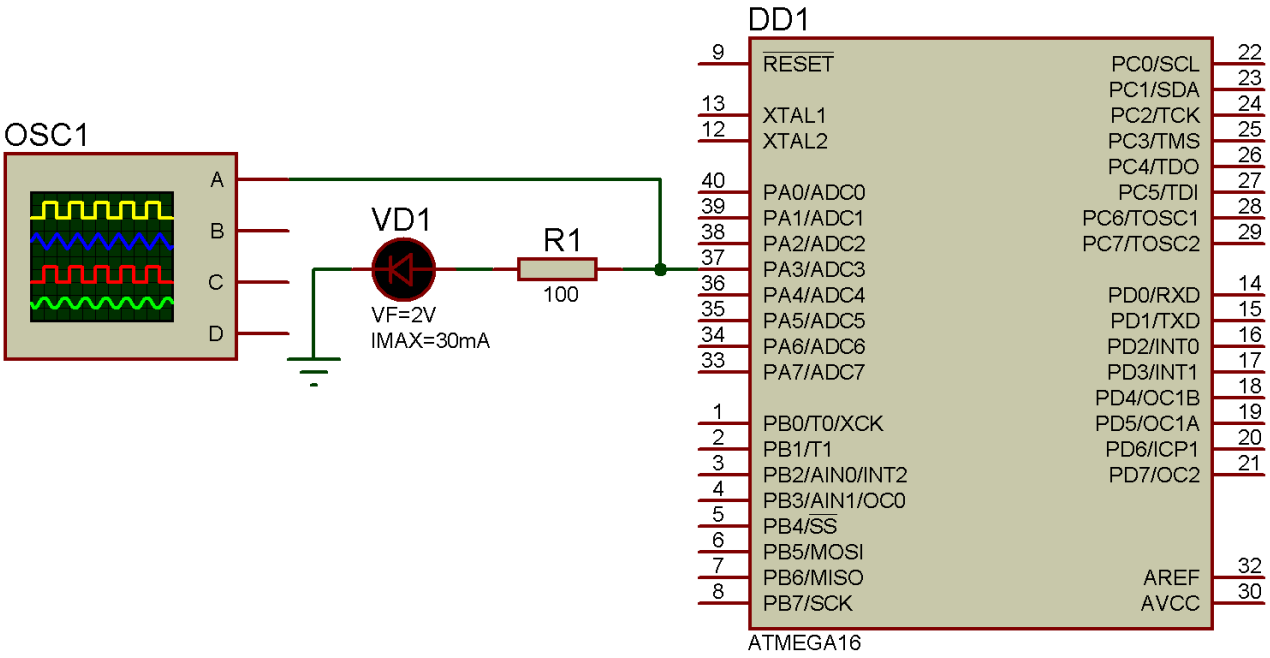


Рисунок 3.15 – Схема устройства, выполняющего мигание светодиодом с заданной частотой, в программе Proteus ISIS

1.3.1 Режим работы Normal таймера/счетчика TC0 с использованием прерывания по событию «Переполнение таймера/счетчика TC0»

Алгоритм работы программы можно условно разделить на две части: блоки команд основной программы, выполняемые в главной и обычных функциях, и блоки команд, относящиеся к процедуре обработки прерывания. Блок-схема алгоритма работы программы представлена на рисунке 3.16.

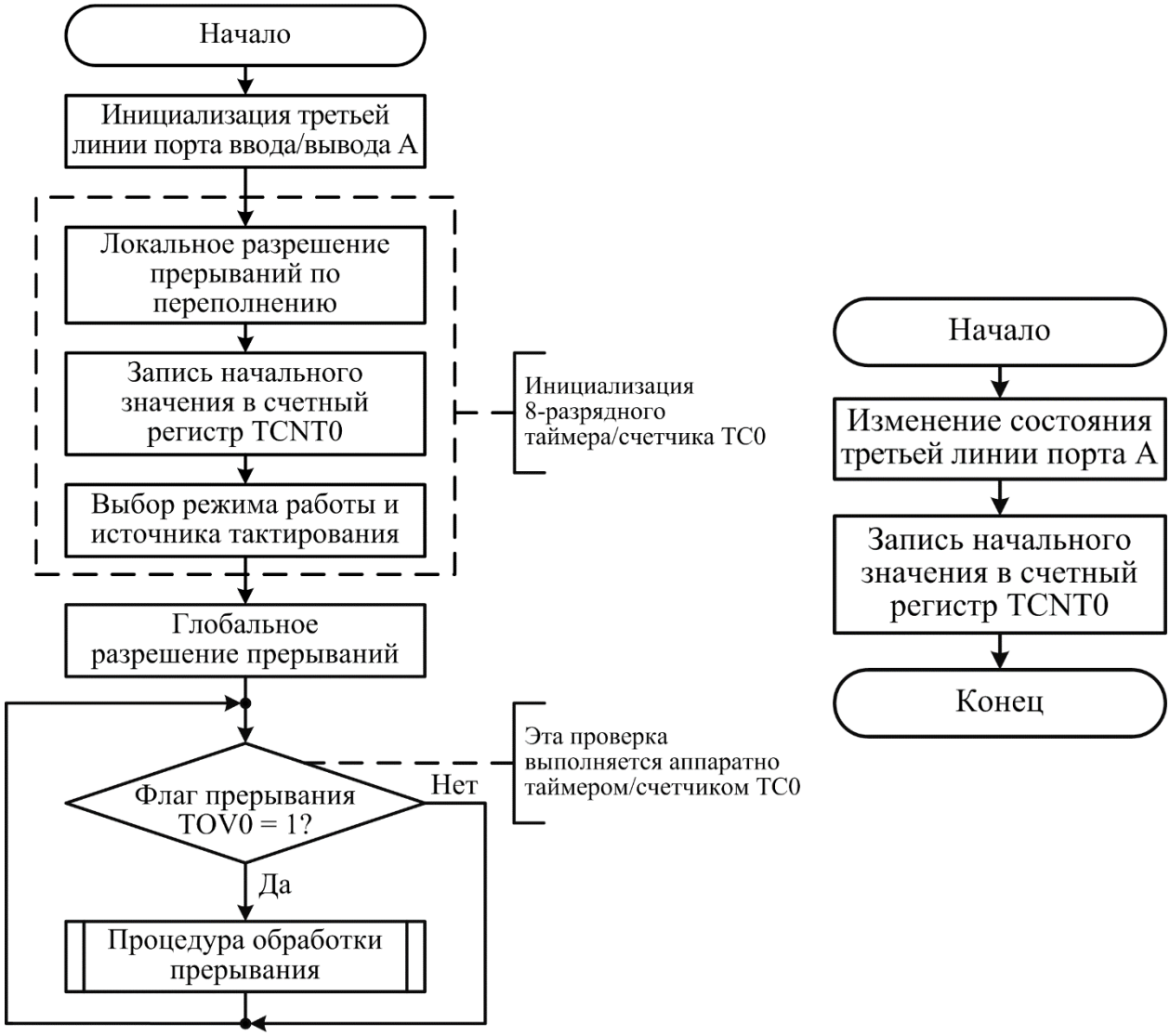
В соответствии с первой частью алгоритма работы программа начинается с инициализации 3-й линии порта ввода/вывода A, которая заключается в настройке этой линии на выход (вывод данных) с формированием на ней сигнала низкого логического уровня, потому что к линии подключен светодиод, который первые 100 мс должен быть выключенным.

Дальше находится группа блоков, в которой выполняется инициализация 8-разрядного таймера/счетчика TC0. Вначале локально разрешаются прерывания по событию «Переполнение таймера/счетчика TC0». Затем выполняется **запись начального значения в счетный регистр TCNT0**, которая представлена отдельным блоком, потому что является обязательным этапом работы программы и соответственно всего устройства для реализации правильного временного интервала до первого прерывания. В конце задается режим работы Normal и выбирается источник тактового сигнала, после подачи которого осуществляется запуск таймера/счетчика TC0.

Затем осуществляется глобальное разрешение прерываний.

Следующий блок ветвления выполняется **не программно, а аппаратно** таймером/счетчиком TC0 микроконтроллера. Флаг прерывания TOV0 по собы-

тию «Переполнение таймера/счетчика TC0» будет установлен в лог. 1 в том случае, когда произойдет переполнение счетного регистра TCNT0. Только после этого будет выполнен переход ко второй части алгоритма работы программы – процедуре обработки прерывания, блок схема которой представлена на рисунке 3.16, *б*.



а – алгоритм основной программы; *б* – алгоритм процедуры обработки прерывания

Рисунок 3.16 – Блок-схема алгоритма работы программы в соответствии с пунктом 1.3.1

Процедура обработки прерывания по событию «Переполнение таймера/счетчика TC0» состоит из двух блоков. В первом выполняется изменение состояния используемой линии порта А на противоположное, что приводит к ми-

ганию светодиода. Во втором блоке записывается начальное значение в счетный регистр для реализации мигания светодиода с требуемой частотой.

После окончания выполнения обработчика прерывания осуществляется возврат в основную программу.

Исходный текст программы на языке программирования C/C++ для рассмотренной блок-схемы представлен в листинге 3.1.

Листинг 3.1

```
1:   #define F_CPU           1000000UL
2:   #define TIME            100*pow(10,-3)
3:   #define K_DEL           1024
4:   #define TCNT0_VALUE     (256-TIME*F_CPU/K_DEL)
5:   #include <avr/io.h>
6:   #include <avr/interrupt.h>
7:   #include <math.h>
8:   int main(void)
9:   {
10:      DDRA|=(1<<PA3);
11:      PORTA&=~(1<<PA3);
12:      TIMSK=(1<<TOIE0);
13:      TCNT0=TCNT0_VALUE;
14:      TCCR0=(0<<WGM01)|(0<<WGM00)|(1<<CS02)|(0<<CS01)|(1<<CS00);
15:      sei();
16:      while (1)
17:      {
18:      }
19:  }
20:  ISR(TIMER0_OVF_vect)
21:  {
22:      PORTA^=(1<<PA3);
23:      TCNT0=TCNT0_VALUE;
24:  }
```

В строках 1–4 с помощью директивы *#define* определены символьные константы:

- F_CPU обозначает тактовую частоту микроконтроллера. Для возможности реализации временных интервалов с максимальной длительностью (262,144 мс) на основе используемого 8-разрядного таймера/счетчика TC0 и внутреннего источника тактирования выбрана и указана частота 1 000 000 Гц, т. е. 1 МГц;

- TIME обозначает время переключения светодиода в секундах. Чтобы светодиод мигал с частотой 5 Гц, нужно сформировать временной интервал с длительностью 100 мс;

- K_DEL обозначает коэффициент деления блока предделителя используемого таймера/счетчика. Для возможности реализации временных интервалов с максимальной длительностью необходимо выбрать коэффициент деления, равный 1024;

- TCNT0_VALUE обозначает начальное значение счетного регистра TCNT0, от которого будет вестись счет таймером/счетчиком TC0, т. е. отсчет значений от этого начального до нулевого (обнуления счетного регистра) будет происходить за время, равное требуемому временному интервалу между переключениями светодиода. Для расчета значения использована формула (3.7). Величина значения составит

$$\text{TCNT0\_VALUE} = 256 - \frac{\text{TIME} \cdot \text{F\_CPU}}{\text{K\_DEL}} = 256 - \frac{100 \cdot 10^{-3} \cdot 10^{6}}{1024} = 158{,}34375. \quad (3.11)$$

В строке 5 программы с помощью директивы *#include* подключается стандартный заголовочный файл, который в свою очередь подключает библиотеку идентификаторов для используемой модели микроконтроллера, выбранной при создании проекта в среде разработки Atmel Studio. В строке 6 подключается библиотека команд для работы с прерываниями *<avr/interrupt.h>*, а в строке 7 – библиотека математических функций *math.h*, из которой будет использована функция возведения числа x в степень y *pow(x,y)* для более наглядной записи значения времени в строке 2.

В строке 8 начинается главная функция программы.

В строках 10–14 находятся однократно выполняемые команды инициализации третьей линии порта ввода/вывода A и 8-разрядного таймера/счетчика. Так в строке 10 выполняется настройка третьей линии порта A на выход (вывод данных) с помощью записи лог. 1 в третий разряд регистра DDRA.

В строке 11 осуществляется формирование сигнала низкого логического уровня на этой линии с помощью записи лог. 0 в третий разряд регистра PORTA. В результате выполнения этих двух команд светодиод будет находиться в начальном выключенном состоянии.

В строке 12 локально разрешаются прерывания по событию «Переполнение таймера/счетчика TC0». Для этого в разряд TOIE0 регистра масок прерываний таймеров/счетчиков TIMSK записывается лог. 1.

В строке 13 записывается начальное значение в счетный регистр TCNT0 в результате присваивания регистру значения символьной константы TCNT0_VALUE. Так как регистры ввода/вывода могут содержать только целочисленные значения, то при записи дробная часть числа будет отбрасываться. **Поэтому длительность формируемого временного интервала будет с погрешностью**.

В строке 14 в результате записи в регистр управления TCCR0 выбирается режим работы Normal (режим WGM01:0=0) и источник тактового сигнала, от которого будет работать таймер/счетчик TC0. Частота тактирования равна частоте микроконтроллера, деленной на 1024 (режим CS02:0=5). Сразу же после выполнения этой команды на таймер/счетчик TC0 подается заданный тактовый сигнал, и TC0 начинает работать.

В строке 15 с помощью макроса *sei()* осуществляется глобальное разрешение прерываний. Если этого не сделать, то процедура обработки прерывания по переполнению таймера/счетчика TC0 никогда не будет запущена.

В строках 16–18 находится бесконечный цикл, в теле которого отсутствуют команды, потому что переключения светодиода осуществляются только в процедуре обработки прерывания по переполнению TC0. После компиляции бесконечный цикл заменяется одной 2-байтовой машинной командой, которая выполняет переход сама на себя. Выполнение этой команды приостанавливается, только когда требуется перейти к обработчику прерывания. Таким образом, центральный процессор микроконтроллера никак не загружен на длительность временного интервала (100 мс), отсчитываемого таймером/счетчиком TC0.

В строках 20–24 находится процедура обработки прерывания **по переполнению** в виде макроса, который определен в библиотеке прерываний, поэтому его можно размещать в исходном тексте программы независимо от положения главной функции. В качестве аргумента передается имя вектора прерывания **TIMER0_OVF_vect**, тем самым реализуется взаимосвязь между блоком команд этого макроса и 4-байтовой машинной командой перехода к этому блоку (командой вызова обработчика прерывания), которая в соответствии с таблицей векторов прерываний будет находиться во flash-памяти по адресу 0x012 (см. таблицу 3.1). После того как произойдет переполнение счетного регистра TCNT0 таймера/счетчика TC0, флаг прерывания TOV0 аппаратно установится в лог. 1 в регистре флагов прерываний таймеров/счетчиков TIFR, поэтому центральный процессор приостановит выполнение команды бесконечного цикла, обратится к адресу 0x012 flash-памяти, прочитает 4-байтовую команду вызова обработчика прерывания по переполнению TC0 и после ее выполнения перейдет к первой команде процедуры обработки прерывания, которая в листинге 3.1 находится в строке 22. Эта команда с помощью оператора *поразрядное исключающее ИЛИ* (^) изменяет состояние третьей линии порта A на противоположное, тем самым мигая светодиодом. В строке 23 обязательно **в счетный регистр записывается значение символьной константы TCNT0_VALUE**, чтобы следующий вызов обработчика и соответственно переключение светодиода произошло снова через 100 мс, реализуя частоту мигания 5 Гц. После окончания выполнения команд обработчика прерывания осуществляется возврат в бесконечный цикл главной функции. Следует отметить, что после вызова обработчика прерывания флаг прерывания TOV0 будет аппаратно сброшен в лог. 0. Также и флаг глобального разрешения прерываний I будет аппаратно сброшен, в результате чего исключается возникновение прерывания в прерывании. Глобально прерывания будут разрешены аппаратным способом только после выхода из процедуры.

На рисунке 3.17 представлена осциллограмма сигнала на третьей линии порта A, полученная для листинга 3.1 в программе Proteus ISIS с помощью осциллографа OCS1. Как видно из рисунка, пять периодов прямоугольного сигнала, соответствующих пяти миганиям светодиода, происходят за одну секунду, т. е. с частотой 5 Гц.

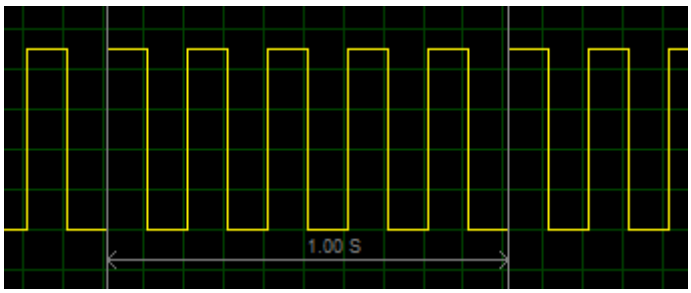


Рисунок 3.17 – Осциллограмма сигнала на третьей линии порта A

1.3.2 Режим работы Normal таймера/счетчика TC0 с использованием прерывания по событию «Совпадение таймера/счетчика TC0»

Блок-схема алгоритма работы программы представлена на рисунке 3.18, а исходный текст программы на языке C/C++ представлен в листинге 3.2.

Прерывание по событию «Совпадение таймера/счетчика TC0» генерируется в случае равенства значений счетного регистра TCNT0 и регистра сравнения OCR0 и при обязательной установке локального OCIE0 в регистре TIMSK и глобального I в регистре SREG флагов разрешения прерываний. Поэтому в блок-схеме алгоритма работы основной программы (рисунок 3.18, *а*) при инициализации 8-разрядного таймера/счетчика TC0 необходимо локально разрешить прерывание по событию «Совпадение таймера/счетчика TC0» и выполнить запись требуемых значений в счетный регистр TCNT0 и в регистр сравнения OCR0.

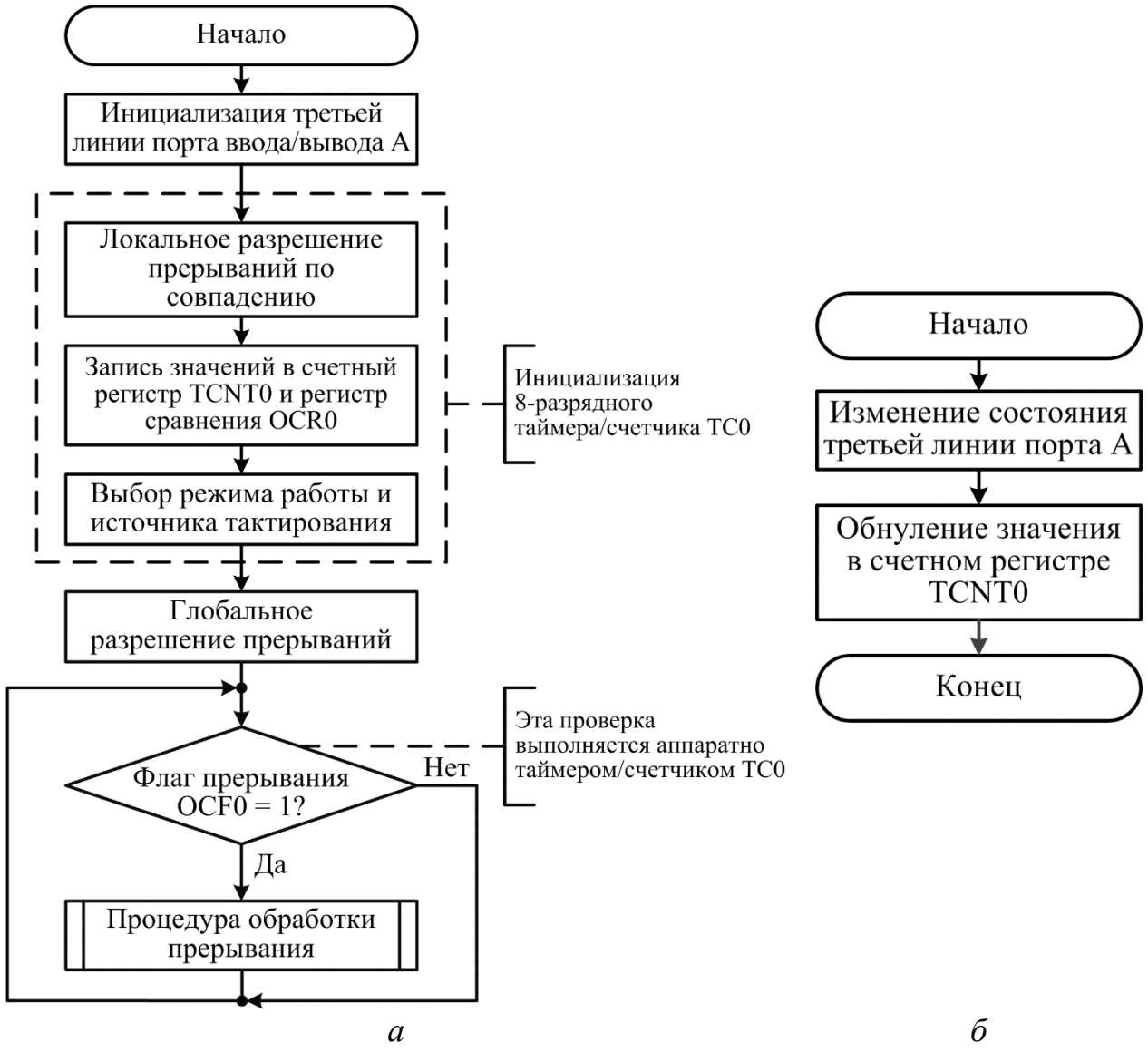
Проверка установки флага прерывания OCF0 при совпадении значений регистра TCNT0 с регистром OCR0 выполняется также **не программно**, **а аппаратно** таймером/счетчиком TC0 микроконтроллера. Появление флага OCF0 в регистре TIFR будет сообщать центральному процессору микроконтроллера о том, что необходимо перейти к процедуре обработки прерывания (рисунок 3.18, *б*). В обработчике прерывания осуществляется изменение состояния третьей линии порта A на противоположное, что приводит к миганию светодиода, и **обнуляется значение счетного регистра** для реализации мигания светодиода с требуемой частотой, т. е. отсчет значений от нулевого до значения, записанного в регистр OCR0, будет происходить за время, равное требуемому временному интервалу между переключениями светодиода.

В листинге программы 3.2 по сравнению с листингом 3.1 имеются следующие изменения:

1 В строке 4 с помощью директивы препроцессора *#define* определяется символьная константа OCR0_VALUE, значение которой будет записано в регистр сравнения OCR0. Для расчета значения использована формула (3.9). Величина значения составит

$$\text{OCR0\_VALUE} = \frac{\text{TIME} \cdot \text{F\_CPU}}{\text{K\_DEL}} = \frac{100 \cdot 10^{-3} \cdot 10^{6}}{1024} = 97{,}65625. \qquad (3.12)$$

2 В строке 12 локально разрешаются прерывания по событию «Совпадение таймера/счетчика TC0». Для этого в разряд OCIE0 регистра масок прерываний таймеров/счетчиков TIMSK записывается лог. 1.



а – алгоритм основной программы; *б* – алгоритм процедуры обработки прерывания

Рисунок 3.18 – Блок-схема алгоритма работы программы для пункта 1.3.2

3 В строках 13-14 выполняется запись значений в регистры TCNT0 и OCR0. В счетный регистр TCNT0 записывается нулевое значение, а в регистр сравнения OCR0 – значение символьной константы OCR0_VALUE, потому что временной интервал требуемой длительности (100 мс) будет сформирован за время, затраченное на выполнение счета от 0x00 по OCR0_VALUE. При совпадении значений регистров TCNT0 и OCR0 будет сгенерирован запрос на прерывание и, если прерывания локально и глобально разрешены, запущен обра-

ботчик прерывания. Так как регистры ввода/вывода могут содержать только целочисленные значения, то при записи дробная часть числа будет отбрасываться. **Поэтому длительность формируемого временного интервала будет с погрешностью**.

4 В строке 21 находится заголовок процедуры обработки прерывания **по совпадению** в виде макроса, в качестве аргумента которого передается имя вектора прерывания **TIMER0_COMP_vect**.

5 Обязательно последней командой в обработчике прерывания по совпадению выполняется **обнуление значения счетного регистра TCNT0** (строка 24), чтобы следующий вызов обработчика и соответственно переключение светодиода произошло снова через 100 мс, реализуя частоту мигания 5 Гц.

Следует отметить, что после вызова обработчика прерывания флаг прерывания OCF0 будет аппаратно сброшен в лог. 0. Также и флаг глобального разрешения прерываний I будет аппаратно сброшен, в результате чего исключается возникновение прерывания в прерывании. Глобально прерывания будут разрешены аппаратным способом только после выхода из процедуры.

Листинг 3.2

```
1:    #define F_CPU            1000000UL
2:    #define TIME             100*pow(10,-3)
3:    #define K_DEL            1024
4:    #define OCR0_VALUE       TIME*F_CPU/K_DEL
5:    #include <avr/io.h>
6:    #include <avr/interrupt.h>
7:    #include <math.h>
8:    int main(void)
9:    {
10:       DDRA|=(1<<PA3);
11:       PORTA&=~(1<<PA3);
12:       TIMSK=(1<<OCIE0);
13:       TCNT0=0x00;
14:       OCR0=OCR0_VALUE;
15:       TCCR0=(0<<WGM01)|(0<<WGM00)|(1<<CS02)|(0<<CS01)|(1<<CS00);
16:       sei();
17:       while (1)
18:       {
19:       }
20:   }
21:   ISR(TIMER0_COMP_vect)
22:   {
23:       PORTA^=(1<<PA3);
24:       TCNT0=0x00;
25:   }
```

1.3.3 Режим работы CTC таймера/счетчика TC0 с использованием прерывания по событию «Переполнение таймера/счетчика TC0»

В режиме работы CTC максимально возможное значение счетного регистра и, следовательно, разрешающая способность таймера/счетчика определяется регистром сравнения OCR0. После достижения значения, записанного в регистре сравнения, значение счетного регистра TCNT0 **сбрасывается аппаратным способом**, и счет продолжается со значения 0x00. При обнулении счетного регистра TCNT0 флаг прерывания по событию «Переполнение» TOV0 устанавливается в лог. 1. В результате будет сформирован запрос на прерывание, и, если прерывания локально и глобально разрешены, то будет запущен обработчик прерывания по событию «Переполнение таймера/счетчика TC0».

Блок-схема алгоритма работы программы, которая представлена на рисунке 3.19, основывается на блок-схеме для режима работы Normal и прерывания по совпадению. Отличия заключаются в следующем:

1) в алгоритме работы основной программы при инициализации 8-разрядного таймера/счетчика TC0 локально разрешаются прерывания по событию «Переполнение таймера/счетчика TC0»;

2) выбирается режим работы CTC (Clear Timer on Compare, Сброс при совпадении);

3) переход к обработчику прерывания осуществляется только после установки аппаратным способом флага TOV0 в регистре TIFR;

4) в алгоритме работы процедуры обработки прерывания осуществляется только изменение состояния используемой линии порта A на противоположное, а обнуление счетного регистра не выполняется, т. к. **регистр обнуляется аппаратным способом**.

В листинге 3.3 программы по сравнению с листингом 3.2 имеются следующие изменения:

1) в строке 12 локально разрешаются прерывания по событию «Переполнение таймера/счетчика TC0». Для этого в разряд TOIE0 регистра масок прерываний таймеров/счетчиков TIMSK записывается лог. 1;

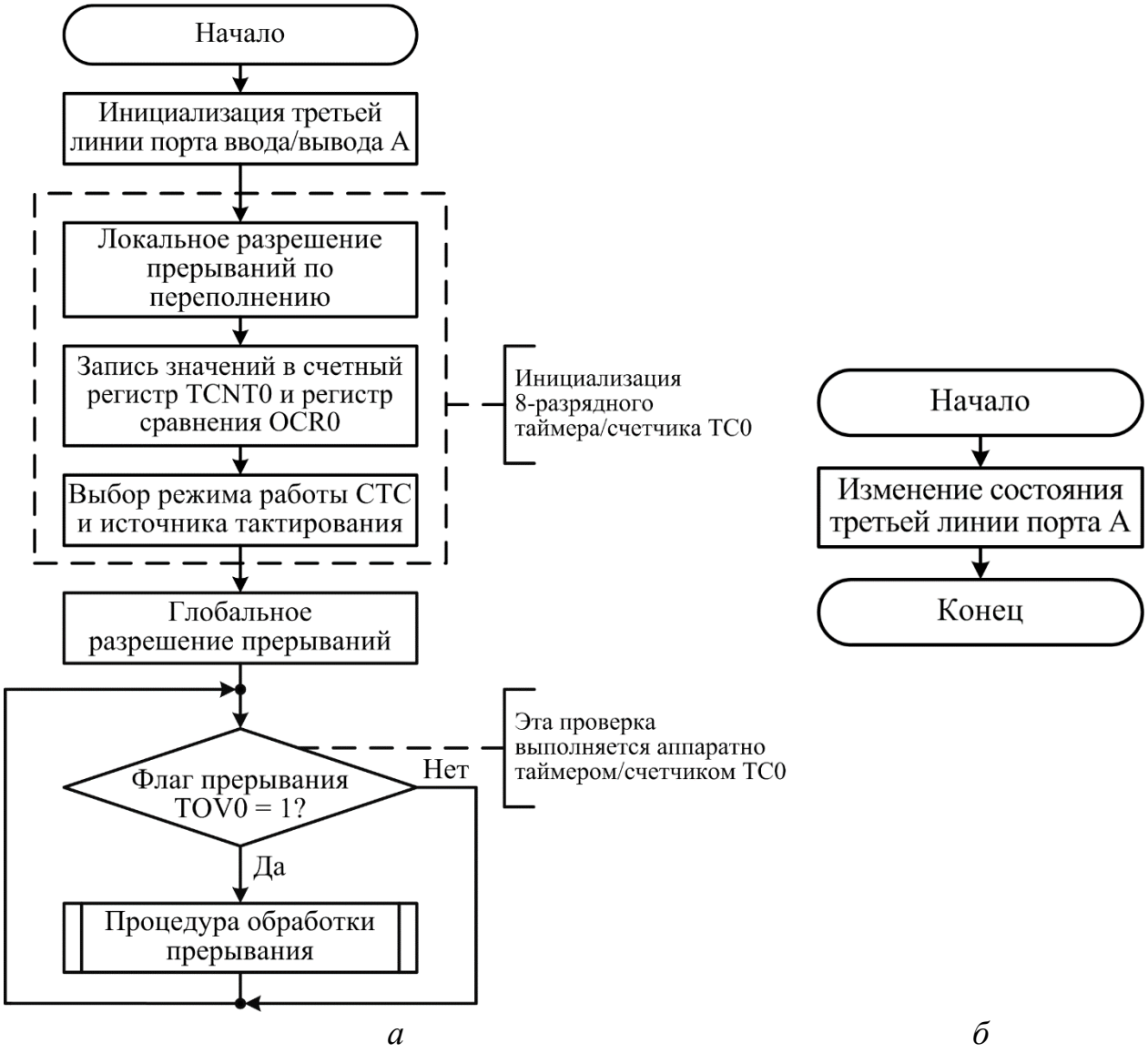
2) в строке 15 в результате записи в регистр управления TCCR0 выбирается режим работы CTC (режим WGM01:0=2);

3) в строке 21 находится заголовок процедуры обработки прерывания **по переполнению** в виде макроса, в качестве аргумента которого передается имя вектора прерывания **TIMER0_OVF_vect**;

4) в процедуре обработки прерывания осуществляется только изменение состояния используемой линии порта A на противоположное, а обнуление счетного регистра не выполняется, т. к. **регистр обнуляется аппаратным способом**.

Следует отметить, что после вызова обработчика прерывания флаг прерывания TOV0 будет аппаратно сброшен в лог. 0. Также и флаг глобального разрешения прерываний I будет аппаратно сброшен, в результате чего исклю-

чается возникновение прерывания в прерывании. Глобально прерывания будут разрешены аппаратным способом только после выхода из процедуры.



а – алгоритм основной программы; *б* – алгоритм процедуры обработки прерывания

Рисунок 3.19 – Блок-схема алгоритма работы программы для пункта 1.3.3

Исходный текст программы на языке программирования C/C++ для рассмотренной блок-схемы представлен в листинге 3.3.

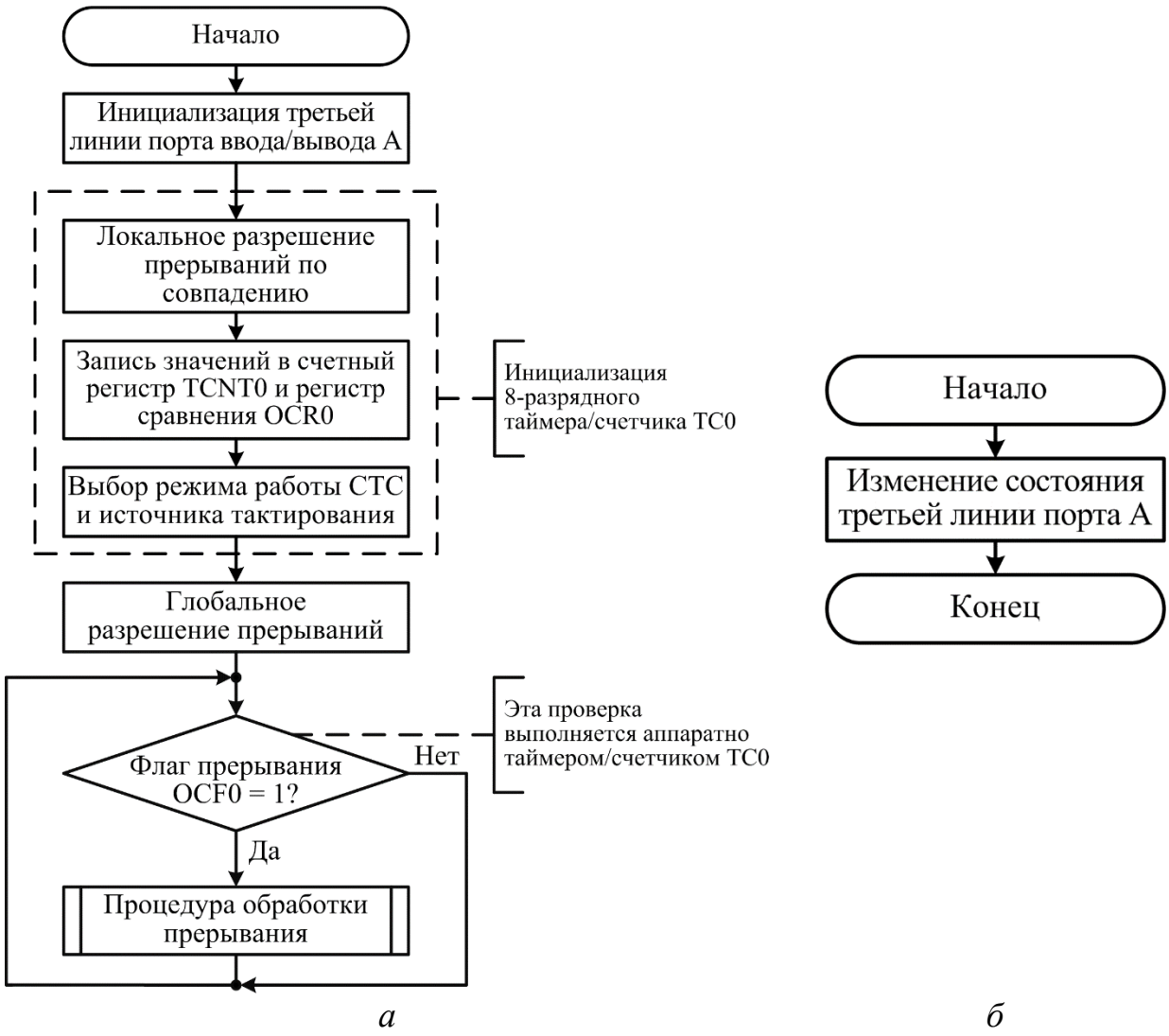
Листинг 3.3

```
1:    #define F_CPU         1000000UL
2:    #define TIME          100*pow(10,-3)
3:    #define K_DEL         1024
4:    #define OCR0_VALUE       TIME*F_CPU/K_DEL
5:    #include <avr/io.h>
6:    #include <avr/interrupt.h>
7:    #include <math.h>
8:    int main(void)
9:    {
10:       DDRA|=(1<<PA3);
11:       PORTA&=~(1<<PA3);
12:       TIMSK=(1<<TOIE0);
13:       TCNT0=0x00;
14:       OCR0=OCR0_VALUE;
15:       TCCR0=(1<<WGM01)|(0<<WGM00)|(1<<CS02)|(0<<CS01)|(1<<CS00);
16:       sei();
17:       while (1)
18:       {
19:       }
20:   }
21:   ISR(TIMER0_OVF_vect)
22:   {
23:       PORTA^=(1<<PA3);
24:   }
```

1.3.4 Режим работы CTC таймера/счетчика TC0 с использованием прерывания по событию «Совпадение таймера/счетчика TC0»

В режиме работы CTC максимально возможное значение счетного регистра и, следовательно, разрешающая способность таймера/счетчика определяется регистром сравнения OCR0. После достижения значения, записанного в регистре сравнения, значение счетного регистра TCNT0 **сбрасывается аппаратным способом**, и счет продолжается со значения 0x00. При равенстве счетного регистра TCNT0 и регистра сравнения OCR0 флаг прерывания по событию «Совпадение» OCF0 устанавливается в лог. 1. В результате будет сформирован запрос на прерывание, и если прерывания локально и глобально разрешены, то будет запущен обработчик прерывания по событию «Совпадение таймера/счетчика TC0».

Блок-схема алгоритма работы программы, которая представлена на рисунке 3.20, основывается на блок-схеме для режима работы CTC и прерывания по переполнению.



а – алгоритм основной программы; *б* – алгоритм процедуры обработки прерывания

Рисунок 3.20 – Блок-схема алгоритма работы программы для пункта 1.3.4

Отличия заключаются в следующем:

1) в алгоритме работы основной программы при инициализации 8-разрядного таймера/счетчика ТС0 локально разрешаются прерывания по событию «Совпадение таймера/счетчика ТС0»;

2) переход к обработчику прерывания осуществляется только после установки аппаратным способом флага OCF0 в регистре TIFR.

Исходный текст программы на языке программирования С/С++ для рассмотренной блок-схемы представлен в листинге 3.4.

Листинг 3.4

```
1:    #define F_CPU             1000000UL
2:    #define TIME              100*pow(10,-3)
3:    #define K_DEL            1024
4:    #define OCR0_VALUE       TIME*F_CPU/K_DEL
5:    #include <avr/io.h>
6:    #include <avr/interrupt.h>
7:    #include <math.h>
8:    int main(void)
9:    {
10:       DDRA|=(1<<PA3);
11:       PORTA&=~(1<<PA3);
12:       TIMSK=(1<<OCIE0);
13:       TCNT0=0x00;
14:       OCR0=OCR0_VALUE;
15:       TCCR0=(1<<WGM01)|(0<<WGM00)|(1<<CS02)|(0<<CS01)|(1<<CS00);
16:       sei();
17:       while (1)
18:       {
19:       }
20:   }
21:   ISR(TIMER0_COMP_vect)
22:   {
23:       PORTA^=(1<<PA3);
24:   }
```

В листинге 3.4 программы по сравнению с листингом 3.3 имеются следующие изменения:

1) в строке 12 локально разрешаются прерывания по событию «Совпадение таймера/счетчика TC0». Для этого в разряд OCIE0 регистра масок прерываний таймеров/счетчиков TIMSK записывается лог. 1;

2) в строке 21 находится заголовок процедуры обработки прерывания **по совпадению** в виде макроса, в качестве аргумента которого передается имя вектора прерывания **TIMER0_COMP_vect**.

Следует отметить, что после вызова обработчика прерывания флаг прерывания OCF0 будет аппаратно сброшен в лог. 0. Также и флаг глобального разрешения прерываний I будет аппаратно сброшен, в результате чего исключается возникновение прерывания в прерывании. Глобально прерывания будут разрешены аппаратным способом только после выхода из процедуры.

2 Порядок выполнения работы

Задание 1

Требуется написать программу на языке программирования C/C++, выполнение которой микроконтроллером ATmega16 позволит реализовать циклически повторяющийся «бегущий огонь» из одного светодиода по расположен-

ным в один ряд 16 светодиодам таким образом, чтобы временные интервалы между переключениями светодиодов были получены не программным способом с помощью функции задержки по времени *_delay_ms()* из библиотеки задержек *delay.h*, а аппаратным с использованием одного из режимов работы 16-разрядного таймера/счетчика TC1 и процедуры обработки прерывания. При этом необходимо предусмотреть выполнение следующих условий и требований:

1) исследуемая схема устройства в программе Proteus ISIS представлена в приложении В;

2) микроконтроллер должен работать на частоте 4 МГц от внутреннего источника тактирования;

3) время переключения (свечения) светодиодов, режим работы таймера/счетчика TC1 и событие, вызывающее прерывание, заданы в таблице 3.16;

4) управление «бегущим огнем» (включение/выключение) требуется реализовать с помощью одной кнопки через запуск и остановку работы таймера/счетчика TC1;

5) в качестве управляющей кнопки необходимо выбрать одну из 10-кнопочного DIP-переключателя SB1, подключенного к портам C и D. Номер кнопки равен заданному варианту;

6) для устранения дребезга контактов кнопки необходимо использовать функцию задержки по времени из библиотеки *delay.h*;

7) для нечетных номеров вариантов направление «бегущего огня» задано слева направо (с VD1 по VD16), а для четных – справа налево (с VD16 по VD1);

8) при написании программы необходимо использовать фрагменты исходного текста программы из решения задания 1 для соответствующего варианта лабораторной работы №2, а именно команды инициализации портов ввода/вывода, блок команд оператора множественного выбора *switch* и команды проверки состояния управляющей кнопки.

Таблица 3.16 – Исходные данные для выполнения задания 1 лабораторной работы №3

Номер варианта	Время переключения светодиодов, мс	Режим работы таймера/счетчика TC1	Прерывание по событию
1	300	Нормальный (Normal)	Переполнение
2	400	Нормальный (Normal)	Переполнение
3	500	Нормальный (Normal)	Совпадение A
4	300	Нормальный (Normal)	Совпадение B
5	400	Нормальный (Normal)	Совпадение B
6	500	Сброс при совпадении (CTC)	Совпадение A
7	300	Сброс при совпадении (CTC)	Совпадение A
8	400	Сброс при совпадении (CTC)	Переполнение
9	500	Сброс при совпадении (CTC)	Переполнение
10	300	Сброс при совпадении (CTC)	Совпадение B

Задание 2

Проверьте работоспособность составленной программы для заданной схемы, выполнив моделирование в программе Proteus ISIS.

С помощью встроенного программного цифрового осциллографа OCS1 проверьте длительность свечения светодиодов и убедитесь в правильности переходов между парами светодиодов: VD8 и VD9; VD1 и VD16. Для этого входы осциллографа OCS1 подключены к отмеченным светодиодам. По полученным осциллограммам с помощью визирных линий убедитесь в совпадении фронтов прямоугольных импульсов и измерьте длительность импульсов, которая должна соответствовать заданному времени переключения светодиодов.

3 Содержание отчета

3.1 Титульный лист.

3.2 Цель работы.

3.3 Исследуемая схема устройства в Proteus ISIS (см. приложение В).

3.4 Исходные данные для выполнения задания 1.

3.5 Блок-схема алгоритма работы разработанной программы.

3.6 Листинг программы на языке программирования C/C++ с подробными комментариями, поясняющими назначение каждой используемой команды.

3.7 Осциллограммы, полученные в программе моделирования Proteus ISIS и подтверждающие правильность работы составленной программы. На осциллограммах с помощью визирных линий должны быть показаны длительности импульсов.

4 Контрольные вопросы

4.1 Что такое система прерываний в семействе микроконтроллеров AVR?

4.2 В чем заключается прерывание в общем случае?

4.3 Дайте определение процедуре обработки прерывания.

4.4 Какие виды флагов разрешения прерываний используются в микроконтроллерах AVR?

4.5 Какими параметрами характеризуется каждый вектор прерывания?

4.6 Какие события вызывают прерывания в микроконтроллере ATmega16?

4.7 Приведите формат записи обработчика прерывания на языке C/C++.

4.8 Какие таймеры/счетчики содержит микроконтроллер ATmega16?

4.9 Опишите функции таймеров/счетчиков микроконтроллера ATmega16.

4.10 Какие регистры ввода/вывода используются для управления 8- и 16-разрядными таймерами/счетчиками?

4.11 В чем заключается назначение счетного регистра, регистра сравнения и управления?

4.12 С какими событиями в работе таймеров/счетчиков микроконтроллера ATmega16 связано появление запросов на обработку прерываний?

4.13 Перечислите режимы работы 8-разрядного таймера/счетчика.

4.14 Перечислите режимы работы 16-разрядного таймера/счетчика.

4.15 Каким образом можно реализовать вызов обработчика прерывания по событию «Переполнение таймера/счетчика» через одинаковые интервалы времени в нормальном режиме работы таймера/счетчика?

4.16 Каким образом можно реализовать вызов обработчика прерывания по событию «Совпадение таймера/счетчика» через одинаковые интервалы времени в нормальном режиме работы таймера/счетчика?

4.17 Каким образом можно реализовать вызов обработчика прерывания по событию «Переполнение таймера/счетчика» через одинаковые интервалы времени в режиме работы СТС таймера/счетчика?

4.18 Каким образом можно реализовать вызов обработчика прерывания по событию «Совпадение таймера/счетчика» через одинаковые интервалы времени в режиме работы СТС таймера/счетчика?

ЛАБОРАТОРНАЯ РАБОТА №4

ИЗУЧЕНИЕ РЕЖИМОВ РАБОТЫ ТАЙМЕРОВ/СЧЕТЧИКОВ МИКРОКОНТРОЛЛЕРА ATMEGA16, ИСПОЛЬЗУЕМЫХ ДЛЯ ФОРМИРОВАНИЯ СИГНАЛОВ С ШИРОТНО-ИМПУЛЬСНОЙ МОДУЛЯЦИЕЙ

Цель работы: изучить принципы работы 8- и 16-разрядных таймеров/счетчиков микроконтроллера ATmega16 в различных режимах формирования сигналов с широтно-импульсной модуляцией.

1 Теоретические сведения

1.1 Режимы работы 8-разрядного таймера/счетчика TC0

8-разрядный таймер/счетчик TC0 поддерживает четыре режима работы:
- Normal (Нормальный);
- CTC (Clear Timer on Compare, Сброс при совпадении);
- Fast PWM (Быстрая ШИМ);
- Phase Correct PWM (ШИМ с точной фазой).

Режим работы Normal (Нормальный).

В этом режиме по каждому импульсу тактового сигнала f_{TC0} осуществляется инкрементирование счетного регистра TCNT0. При переходе через значение 0xFF возникает переполнение, и счет продолжается со значения 0x00. При равенстве счетного регистра TCNT0 и регистра сравнения OCR0 наступает событие «Совпадение таймера/счетчика TC0» и может изменяться состояние вывода OC0 микроконтроллера в соответствии с установками разрядов COM01:COM00 в регистре TCCR0 (таблица 4.1).

Таблица 4.1 – Режимы управления состоянием вывода OC0 таймера/счетчика TC0 в режиме Normal (Нормальный)

Номер режима	COM01	COM00	Описание
0	0	0	Таймер/счетчик TC0 отключен от вывода OC0
1	0	1	Состояние вывода меняется на противоположное при каждом совпадении счетного регистра TCNT0 и регистра сравнения OCR0
2	1	0	Состояние вывода OC0 сбрасывается в лог. 0 при каждом совпадении
3	1	1	Состояние вывода OC0 устанавливается в лог. 1 при каждом совпадении

При необходимости состояние вывода OC0 может быть изменено принудительно записью лог. 1 в разряд FOC0 в регистре TCCR0. Прерывание при этом не генерируется.

Режим работы CTC (Clear Timer on Compare, Сброс при совпадении).

В этом режиме тоже по каждому импульсу тактового сигнала f_{TC0} осуществляется инкрементирование счетного регистра TCNT0. Однако максимально возможное значение счетного регистра и, следовательно, разрешающая способность таймера/счетчика и генератора ШИМ-сигнала определяется регистром сравнения OCR0. После достижения значения, записанного в регистре сравнения, значение счетного регистра TCNT0 аппаратно сбрасывается, и счет продолжается со значения 0x00. При равенстве счетного регистра TCNT0 и регистра сравнения OCR0 наступает событие «Совпадение таймера/счетчика TC0» и может изменяться состояние вывода OC0 в соответствии с установками разрядов COM01:COM00 в регистре TCCR0 (таблица 4.2, рисунок 4.1).

Таблица 4.2 – Режимы управления состоянием вывода OC0 таймера/счетчика TC0 в режиме CTC (Сброс при совпадении)

Номер режима	COM01	COM00	Описание
0	0	0	Таймер/счетчик TC0 отключен от вывода OC0
1	0	1	Состояние вывода OC0 меняется на противоположное при каждом совпадении счетного регистра TCNT0 и регистра сравнения OCR0
2	1	0	Состояние вывода OC0 сбрасывается в лог. 0 при каждом совпадении
3	1	1	Состояние вывода OC0 устанавливается в лог. 1 при каждом совпадении

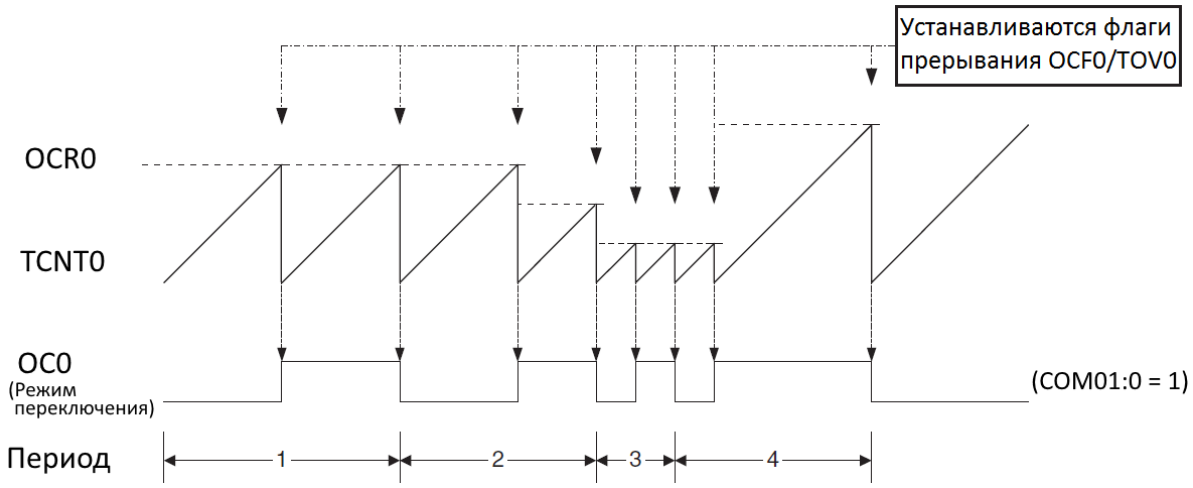


Рисунок 4.1 – Временные диаграммы формирования сигнала в режиме CTC таймера/счетчика TC0

В режиме переключения состояния вывода OC0 (режим COM01:0=1) частота генерируемого сигнала в герцах будет определяться по формуле

$$f_{OC0.CTC} = \frac{f_T}{2 \cdot k_{дел} \cdot (1 + OCR0)}, \quad (4.1)$$

где f_T – тактовая частота микроконтроллера, Гц;

$k_{дел}$ – коэффициент деления предделителя (1, 8, 64, 256, 1024).

При необходимости состояние вывода OC0 может быть изменено принудительно записью лог. 1 в разряд FOC0 регистра TCCR0. Прерывание при этом не генерируется и сброс счетного регистра не производится. В режиме CTC нет двойной буферизации.

Режим работы Fast PWM (Быстрая ШИМ).

Режим Fast PWM (FPWM) позволяет генерировать высокочастотный сигнал с широтно-импульсной модуляцией. Счетный регистр TCNT0 в этом режиме функционирует как суммирующий счетчик, инкрементирование которого осуществляется по каждому импульсу тактового сигнала таймера/счетчика f_{TC0}. Состояние счетчика изменяется от нижнего предела счета 0x00 до максимального предела счета 0xFF, после чего счетный регистр TCNT0 сбрасывается, и счет продолжается снова со значения 0x00. В том же такте сигнала f_{TC0}, в котором обнуляется счетный регистр TCNT0, флаг прерывания по событию «Переполнение таймера/счетчика TC0» TOV0 устанавливается в лог. 1. При равенстве счетного регистра TCNT0 и регистра сравнения OCR0 флаг прерывания по событию «Совпадение таймера/счетчика TC0» OCF0 устанавливается в лог. 1.

Состояние вывода OC0 может изменяться в соответствии с установками разрядов COM01:COM00 в регистре TCCR0 (таблица 4.3, рисунок 4.2).

Таблица 4.3 – Режимы управления состоянием вывода OC0 в режиме Fast PWM

Номер режима	COM01	COM00	Описание
0	0	0	Таймер/счетчик TC0 отключен от вывода OC0
1	0	1	Зарезервировано (не используется)
2	1	0	Состояние вывода OC0 сбрасывается в лог. 0 при равенстве регистров TCNT0 и OCR0. Устанавливается в лог. 1 при достижении счетчиком верхнего предела счета 0xFF (прямой режим)
3	1	1	Состояние вывода OC0 устанавливается в лог. 1 при равенстве регистров TCNT0 и OCR0. Сбрасывается в лог. 0 при достижении счетчиком верхнего предела счета 0xFF (инверсный режим)

Особенностью работы блока сравнения в режиме Fast PWM является двойная буферизация записи в регистр OCR0, которая заключается в том, что записываемое число на самом деле сохраняется в специальном буферном регистре, а изменение содержимого регистра сравнения происходит только в момент достижения счетным регистром верхнего предела счета 0xFF. Благодаря такому решению исключается появление помех – несимметричных импульсов

сигнала (импульсов, длина которых равна нечетному количеству тактов) на выходе ШИМ-генератора, которые были неизбежны при непосредственной записи в регистр сравнения.

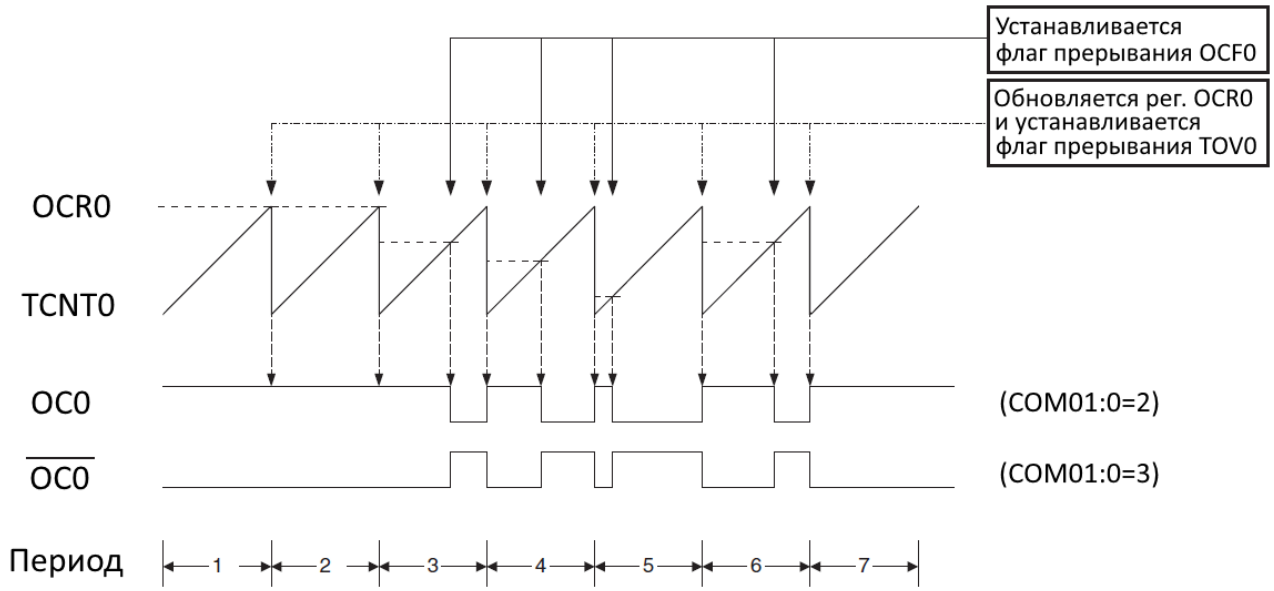


Рисунок 4.2 – Временные диаграммы формирования ШИМ-сигнала в режиме Fast PWM таймера/счетчика TC0

Частота генерируемого ШИМ-сигнала в герцах будет определяться по формуле

$$f_{OC0.FPWM}=\frac{f_T}{256\cdot k_{дел}}. \qquad (4.2)$$

Установка предельных значений регистра сравнения OCR0 связана с особыми случаями генерации ШИМ-сигнала:

1) если в прямом режиме формирования ШИМ-сигнала (режим COM01:0=2) в регистр сравнения OCR0 записать значение, равное нижнему пределу счета (0x00), то на выводе OC0 будут наблюдаться короткие импульсы с длительностью $k_{дел}/f_T$ и периодом повторения $256\cdot k_{дел}/f_T$, а в инверсном режиме (COM01:0=3) импульсы будут иметь длительность $255\cdot k_{дел}/f_T$ и период повторения $256\cdot k_{дел}/f_T$;

2) если в регистр сравнения OCR0 записать значение, равное максимальному значению (0xFF), то вывод OC0 будет иметь устойчивое (постоянное) состояние лог. 1 (режим COM01:0=2) или лог. 0 (режим COM01:0=3).

Возможность генерации высокочастотных ШИМ-сигналов делает использование режима Fast PWM полезным в задачах стабилизации питания, выпрямления и цифроаналогового преобразования. Высокая частота при этом позволя-

ет использовать внешние элементы физически малых размеров (индуктивности, конденсаторы), тем самым снижая общую стоимость системы.

Режим работы Phase Correct PWM (ШИМ с точной фазой/фазовой коррекцией).

Этот режим, как и режим Fast PWM, предназначен для формирования сигналов с широтно-импульсной модуляцией. Режим Phase Correct PWM (PCPWM) основан на двунаправленной работе счетного регистра TCNT0. Состояние счетчика изменяется по каждому импульсу тактового сигнала таймера/счетчика f_{TC0} следующим образом: вначале осуществляется инкрементирование от 0x00 до верхнего предела счета 0xFF (прямой счет или счет вверх), а затем выполняется декрементирование в обратном направлении до нижнего предела счета 0x00 (обратный счет или счет вниз), при достижении которого снова происходит смена направления счета и устанавливается в лог. 1 флаг прерывания по событию «Переполнение таймера/счетчика TC0» TOV0. Следовательно, для обнуления счетного регистра требуется в два раза больше времени, чем в режиме Fast PWM, и соответственно максимальная частота ШИМ-сигнала будет в два раза меньше максимальной частоты в режиме Fast PWM. Но тем не менее благодаря симметричности изменения состояния счетного регистра TCNT0 **отсутствует сдвиг фазы**, и поэтому режим Phase Correct PWM предпочтительнее использовать для решения задач управления двигателями.

При равенстве счетного регистра TCNT0 и регистра сравнения OCR0 флаг прерывания по событию «Совпадение таймера/счетчика TC0» OCF0 устанавливается в лог. 1 и изменяется состояние вывода OC0 в соответствии с установками разрядов COM01:COM00 в регистре TCCR0 (таблица 4.4, рисунок 4.3).

Таблица 4.4 – Режимы управления состоянием вывода OC0 в режиме PCPWM

Номер режима	COM01	COM00	Описание
0	0	0	Таймер/счетчик TC0 отключен от вывода OC0
1	0	1	Зарезервировано (не используется)
2	1	0	Состояние вывода OC0 сбрасывается в лог. 0 при равенстве регистров TCNT0 и OCR0 во время прямого счета и устанавливается в лог. 1 при равенстве во время обратного счета (прямой режим)
3	1	1	Состояние вывода OC0 устанавливается в лог. 1 при равенстве регистров TCNT0 и OCR0 во время прямого счета и сбрасывается в лог. 0 при равенстве во время обратного счета (инверсный режим)

Для исключения несимметричных импульсов ШИМ-сигнала в этом режиме тоже реализована двойная буферизация записи в регистр сравнения OCR0. Поэтому действительное изменение содержимого этого регистра происходит только в момент достижения счетным регистром TCNT0 максимального значения 0xFF.

Если в регистр сравнения записать минимальное или максимальное значение, то при следующем совпадении с содержимым счетного регистра вывод OC0 будет иметь устойчивое (постоянное) состояние в соответствии с таблицей 4.5.

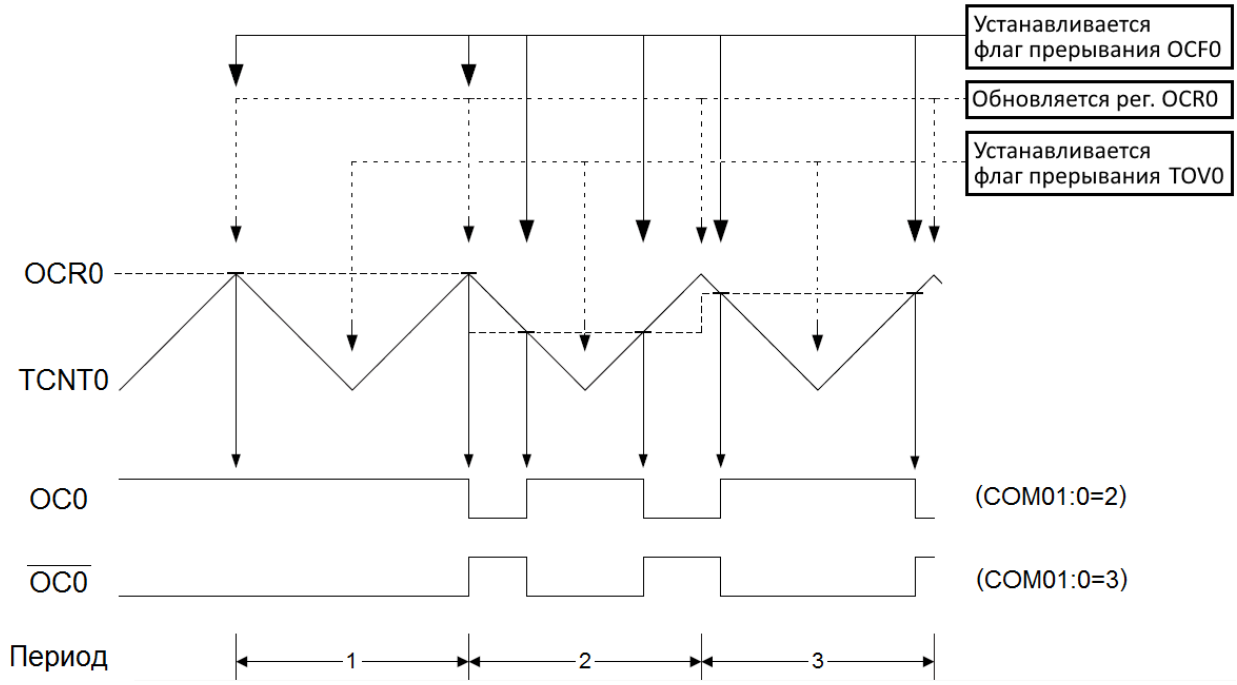


Рисунок 4.3 – Временные диаграммы формирования ШИМ-сигнала в режиме Phase Correct PWM таймера/счетчика TC0

Таблица 4.5 – Устойчивые состояния вывода OC0 в режиме PCPWM

Номер режима	COM01	COM00	Регистр OCR0	Состояние вывода OC0
2	1	0	Минимальное значение (0x00)	0
2	1	0	Максимальное значение (0xFF)	1
3	1	1	Минимальное значение (0x00)	1
3	1	1	Максимальное значение (0xFF)	0

Частота генерируемого ШИМ-сигнала в герцах будет определяться по формуле

$$f_{OC0.PCPWM}=\frac{f_T}{510\cdot k_{дел}}. \quad (4.3)$$

1.2 Режимы работы 16-разрядного таймера/счетчика TC1

16-разрядный таймер/счетчик TC1 поддерживает пять основных режимов работы, каждый из которых имеет свои разновидности в соответствии с таблицей 3.13:

- Normal (Нормальный);
- CTC (Clear Timer on Compare, Сброс при совпадении);
- Fast PWM (Быстрая ШИМ);
- Phase Correct PWM (ШИМ с точной фазой);
- Phase and Frequency Correct PWM (ШИМ с точной фазой и частотой/фазовой и частотной коррекцией).

Режим работы Normal (Нормальный).

В этом режиме по каждому импульсу тактового сигнала f_{TC1} осуществляется инкрементирование счетного регистра TCNT1. При переходе через значение 0xFFFF возникает переполнение, и счет продолжается со значения 0x0000. При равенстве счетного регистра TCNT1 и регистра сравнения OCR1A (OCR1B) наступает событие «Совпадение A (B) таймера/счетчика TC1» и может изменяться состояние вывода OC1A (OC1B) микроконтроллера в соответствии с установками разрядов COM1A1:COM1A0 (COM1B1:COM1B0) регистра TCCR1A (таблица 4.6, где x означает канал A (B) ШИМ-генератора).

Таблица 4.6 – Режимы управления состоянием вывода OC1x в режиме Normal

Номер режима	COM1x1	COM1x0	Описание
0	0	0	Таймер/счетчик TC1 отключен от вывода OC1x
1	0	1	Состояние вывода OC1x меняется на противоположное при каждом совпадении счетного регистра TCNT1 и регистра сравнения OCR1x
2	1	0	Состояние вывода OC1x сбрасывается в лог. 0 при каждом совпадении
3	1	1	Состояние вывода OC1x устанавливается в лог. 1 при каждом совпадении

При необходимости состояние вывода OC1A (OC1B) может быть изменено принудительно записью лог. 1 в разряд FOC1A (FOC1B) регистра TCCR1A. Прерывание при этом не генерируется.

Режим работы CTC (Clear Timer on Compare, Сброс при совпадении).

В этом режиме тоже по каждому импульсу тактового сигнала f_{TC1} осуществляется инкрементирование счетного регистра TCNT1. Аппаратный сброс счетного регистра происходит только в тех случаях, когда его значение совпадает со значением регистра сравнения OCR1A (режим WGM13:0=4) или регистра захвата ICR1 (режим WGM13:0=12). Таким образом, значение регистра OCR1A (ICR1) определяет верхний предел счета и, следовательно, разрешающую способность таймера/счетчика и генератора ШИМ-сигнала.

В режимах работы 4 и 12 при наступлении события «Совпадение A (B) таймера/счетчика TC1» состояние вывода OC1A (OC1B) микроконтроллера может изменяться в соответствии с установками разрядов COM1A1:COM1A0 (COM1B1:COM1B0) регистра TCCR1A (таблица 4.7, где x означает канал A (B) ШИМ-генератора).

Таблица 4.7 – Режимы управления состоянием вывода OC1x в режиме CTC

Номер режима	COM1x1	COM1x0	Описание
0	0	0	Таймер/счетчик TC1 отключен от вывода OC1x
1	0	1	Состояние вывода OC1x меняется на противоположное при каждом совпадении счетного регистра TCNT1 и регистра сравнения OCR1x
2	1	0	Состояние вывода OC1x сбрасывается в лог. 0 при каждом совпадении
3	1	1	Состояние вывода OC1x устанавливается в лог. 1 при каждом совпадении

На рисунке 4.4 представлены временные диаграммы формирования сигнала в режиме CTC (WGM13:0=4) таймера/счетчика TC1.

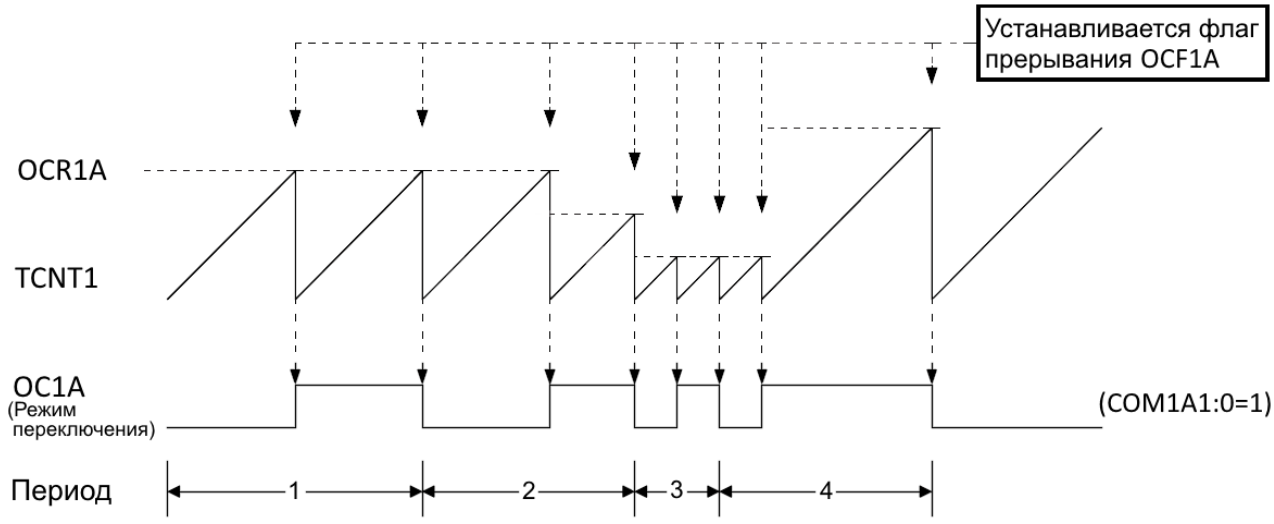


Рисунок 4.4 – Временные диаграммы формирования сигнала в режиме CTC (WGM13:0=4) таймера/счетчика TC1

В режиме переключения состояния вывода OC1x (режим COM1x1:0=1), где x означает канал A (B) ШИМ-генератора, частота генерируемого сигнала в герцах будет определяться по формуле

$$f_{OC1x.CTC} = \frac{f_T}{2 \cdot k_{дел} \cdot (1 + TOP)}, \quad (4.4)$$

где f_T – тактовая частота микроконтроллера, Гц;

$k_{дел}$ – коэффициент деления предделителя (1, 8, 64, 256, 1024);

TOP – верхний предел счета, определяемый регистром OCR1A или ICR1.

Таким образом, т. к. в режиме 4 частота генерируемого сигнала и сброс счетного регистра будут зависеть только от значения регистра OCR1A, то при COM1A1:0=COM1B1:0=1 сигнал на выводе OC1B будет:

- совпадать по частоте и фазе с сигналом на выводе OC1A, если OCR1B = OCR1A;
- отличаться только по фазе с сигналом на выводе OC1A, если OCR1B ≤ OCR1A;
- отсутствовать, если OCR1B > OCR1A.

В режиме 12 частота генерируемого сигнала и сброс счетного регистра будут зависеть только от значения регистра ICR1, поэтому при COM1A1:0=COM1B1:0=1 сигналы на выводах OC1A и OC1B будут:

- совпадать по частоте и фазе, если (OCR1A = OCR1B) ≤ ICR1;
- отличаться только по фазе, если (OCR1A ≠ OCR1B) ≤ ICR1;
- отсутствовать, если OCR1A > ICR1, OCR1B > ICR1.

Режим работы Fast PWM (Быстрая ШИМ).

Этот режим позволяет генерировать высокочастотный ШИМ-сигнал различной разрядности. Счетный регистр TCNT1 в этом режиме функционирует как суммирующий счетчик, инкрементирование которого осуществляется по каждому импульсу тактового сигнала таймера/счетчика f_{TC1}. Состояние счетчика изменяется от нижнего предела счета 0x0000 до верхнего предела счета, который может принимать фиксированные значения либо определяется регистром ICR1 или OCR1A, после чего счетный регистр TCNT1 сбрасывается, и счет продолжается снова со значения 0x0000. Таким образом, значение верхнего предела счета определяет разрешающую способность таймера/счетчика и ШИМ-генератора и соответственно разрешение ШИМ-сигнала (таблица 4.8). Разрешающую способность ШИМ-генератора можно рассчитать по формуле

$$R_{PWM} = \frac{\log_{10}(TOP + 1)}{\log_{10}(2)}, \qquad (4.5)$$

где TOP – верхний предел счета.

Таблица 4.8 – Разрешающая способность ШИМ-генератора в режиме Fast PWM

Номер режима	WGM13	WGM12	WGM11	WGM10	Разрешающая способность	Верхний предел счета (TOP)
5	0	1	0	1	8 разрядов	0x00FF
6	0	1	1	0	9 разрядов	0x01FF
7	0	1	1	1	10 разрядов	0x03FF
14	1	1	1	0	Переменная (2–16)	ICR1 (0x0003–0xFFFF)
15	1	1	1	1	Переменная (2–16)	OCR1A (0x0003–0xFFFF)

Режимы 14 и 15 обеспечивают более широкий диапазон регулировки частоты генерируемых ШИМ-сигналов благодаря изменению в широких пределах разрешающей способности ШИМ-генератора, т. е. изменению значений в регистре ICR1 или OCR1A.

Режимы 14 и 15 отличаются между собой тем, что если при генерации ШИМ-сигнала его частота меняется очень часто, то для задания верхнего предела счета необходимо использовать регистр сравнения OCR1A. В этом случае за счет двойной буферизации записи в регистр сравнения (для ICR1 буферизация не выполняется) исключается появление несимметричных импульсов сигнала на выходе ШИМ-генератора.

В режиме 15 формирование ШИМ-сигнала с переменой скважностью будет возможно только на выводе OC1B, т. к. сигнал на выводе OC1A будет:

- иметь постоянную скважность, равную 2, при COM1A1:0=1;
- устойчивое состояние лог. 1 при COM1A1:0=2;
- устойчивое состояние лог. 0 при COM1A1:0=3.

Режимы управления состоянием вывода OC1A (OC1B) в режиме Fast PWM представлены в таблице 4.9, а временные диаграммы формирования ШИМ-сигнала показаны на рисунке 4.5, где x означает канал A (B) ШИМ-генератора.

Таблица 4.9 – Режимы управления состоянием вывода OC1x в Fast PWM

Номер режима	COM1x1	COM1x0	Описание
0	0	0	Таймер/счетчик TC1 отключен от вывода OC1x
1	0	1	В режимах 14 и 15 состояние вывода OC1x меняется на противоположное при каждом совпадении счетного регистра TCNT1 и регистра сравнения OCR1x. В режимах 5–7 таймер/счетчик TC1 отключен от вывода OC1x
2	1	0	Состояние вывода OC1x сбрасывается в лог. 0 при равенстве регистров TCNT1 и OCR1x и устанавливается в лог. 1 при достижении счетчиком верхнего предела счета TOP (прямой режим)
3	1	1	Состояние вывода OC1x устанавливается в лог. 1 при равенстве регистров TCNT1 и OCR1x и сбрасывается в лог. 0 при достижении счетчиком верхнего предела счета TOP (инверсный режим)

Частота генерируемого ШИМ-сигнала в герцах будет определяться по формуле

$$f_{OC1x.FPWM} = \frac{f_T}{k_{дел} \cdot (1 + TOP)}, \quad (4.6)$$

где TOP – верхний предел счета, определяемый в соответствии с таблицей 4.8.

Установка предельных значений регистра сравнения OCR1x связана с особыми случаями генерации ШИМ-сигнала в режиме Fast PWM таймера/счетчика TC1:

1) если в прямом режиме формирования ШИМ-сигнала (режим COM1x1:0=2) в регистр сравнения OCR1x записать значение, равное нижнему пределу счета (0x0000), то на выводе OC1x будут наблюдаться короткие импульсы с длительностью $k_{дел}/f_T$ и периодом повторения $k_{дел}\cdot(1 + TOP)/f_T$, а в инверсном режиме (COM1x1:0=3) импульсы будут иметь длительность $k_{дел}\cdot TOP/f_T$ и период повторения $k_{дел}\cdot(1 + TOP)/f_T$;

2) если в регистр сравнения OCR1x записать значение, равное верхнему пределу счета (TOP), то вывод OC1x будет иметь устойчивое (постоянное) состояние лог. 1 (режим COM1x1:0=2) или лог. 0 (режим COM1x1:0=3).

В том же такте сигнала f_{TC1}, в котором обнуляется счетный регистр TCNT1, флаг прерывания по событию «Переполнение таймера/счетчика TC1» TOV1 устанавливается в лог. 1. При равенстве счетного регистра TCNT1 и регистра сравнения OCR1x флаг прерывания по событию «Совпадение x таймера/счетчика TC1» OCF1x устанавливается в лог. 1. Флаг прерывания по событию «Захват» ICF1 устанавливается в лог. 1 при изменении значения счетного регистра с верхнего предела счета, определяемого регистром ICR1, на значение 0x0000. Таким образом, в режимах 5–7 и 15 могут быть реализованы только прерывания по переполнению, совпадению A и совпадению B, а в режиме 14 – дополнительно и прерывание по захвату.

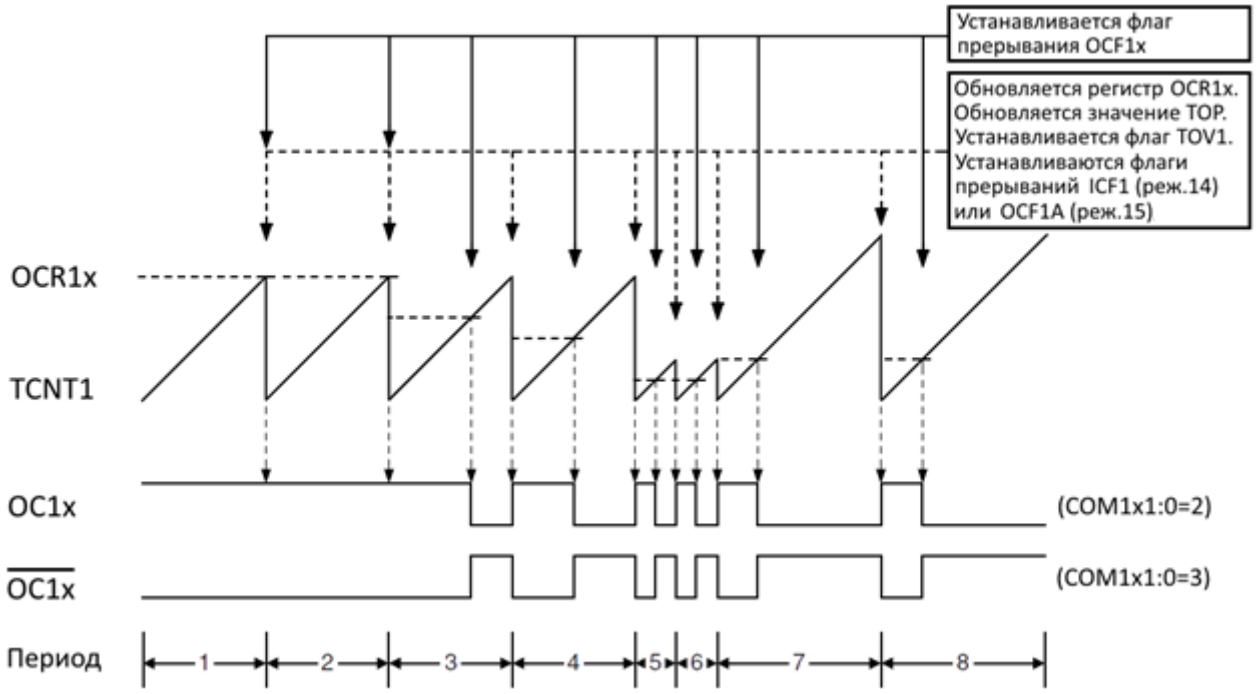


Рисунок 4.5 – Временные диаграммы формирования ШИМ-сигнала в режиме Fast PWM таймера/счетчика TC1

Режим работы Phase Correct PWM (ШИМ с точной фазой/фазовой коррекцией).

В этом режиме счетный регистр TCNT1 функционирует как реверсивный счетчик, состояние которого вначале инкрементируется от 0x0000 до верхнего предела счета TOP (прямой счет или счет вверх), а затем выполняется декрементирование в обратном направлении до нижнего предела счета 0x0000 (обратный счет или счет вниз), при достижении которого снова происходит смена направления счета. Следовательно, для обнуления счетного регистра требуется в два раза больше времени, чем в режиме Fast PWM, и соответственно максимальная частота ШИМ-сигнала будет в два раза меньше максимальной частоты в режиме Fast PWM. Благодаря симметричности изменения состояния счетного регистра TCNT1 отсутствует сдвиг фазы ШИМ-сигнала по сравнению с режимом Fast PWM. В зависимости от установок разрядов WGM13:0 верхний предел счета TOP может принимать либо фиксированные значения, либо определяется регистром ICR1 или OCR1A. Разрешающую способность генератора ШИМ-сигнала в режиме Phase Correct PWM (PCPWM) можно также рассчитать по формуле (4.5) (таблица 4.10).

Таблица 4.10 – Разрешающая способность ШИМ-генератора в режиме PCPWM

Номер режима	WGM13	WGM12	WGM11	WGM10	Разрешающая способность	Верхний предел счета (TOP)
1	0	0	0	1	8 разрядов	0x00FF
2	0	0	1	0	9 разрядов	0x01FF
3	0	0	1	1	10 разрядов	0x03FF
10	1	0	1	0	Переменная (2–16)	ICR1 (0x0003–0xFFFF)
11	1	0	1	1	Переменная (2–16)	OCR1A (0x0003–0xFFFF)

Режимы 10 и 11 обеспечивают более широкий диапазон регулировки частоты генерируемых ШИМ-сигналов благодаря изменению в широких пределах разрешающей способности ШИМ-генератора, т. е. изменению значений в регистре ICR1 или OCR1A.

Режимы 10 и 11 отличаются между собой тем, что если при генерации ШИМ-сигнала его частота меняется очень часто, то для задания верхнего предела счета необходимо использовать регистр сравнения OCR1A. В этом случае за счет двойной буферизации записи в регистр сравнения (для ICR1 буферизация не выполняется) исключается появление несимметричных импульсов сигнала на выходе ШИМ-генератора.

В режиме 11 формирование ШИМ-сигнала с переменой скважностью будет возможно только на выводе OC1B.

Режимы управления состоянием вывода OC1A (OC1B) в режиме Phase Correct PWM представлены в таблице 4.11, а временные диаграммы формиро-

вания ШИМ-сигнала показаны на рисунке 4.6, где x означает канал A (B) ШИМ-генератора.

Таблица 4.11 – Режимы управления состоянием вывода OC1x в PCPWM

Номер режима	COM1x1	COM1x0	Описание
0	0	0	Таймер/счетчик TC1 отключен от вывода OC1x
1	0	1	В режиме 11 состояние вывода OC1A меняется на противоположное при каждом совпадении счетного регистра TCNT1 и регистра сравнения OCR1A. В режимах 1–3 таймер/счетчик TC1 отключен от вывода OC1x
2	1	0	Состояние вывода OC1x сбрасывается в лог. 0 при равенстве регистров TCNT1 и OCR1x во время прямого счета и устанавливается в лог. 1 при равенстве во время обратного счета (прямой режим)
3	1	1	Состояние вывода OC1x устанавливается в лог. 1 при равенстве регистров TCNT1 и OCR1x во время прямого счета и сбрасывается в лог. 0 при равенстве во время обратного счета (инверсный режим)

Частота генерируемого ШИМ-сигнала в герцах будет определяться по формуле

$$f_{OC1x.PCPWM} = \frac{f_T}{2 \cdot k_{дел} \cdot TOP}, \quad (4.7)$$

где TOP – верхний предел счета, определяемый в соответствии с таблицей 4.10.

Вывод OC1x будет иметь устойчивое (постоянное) состояние в соответствии с таблицей 4.12, где x означает канал A (B) ШИМ-генератора.

Таблица 4.12 – Устойчивые состояния вывода OC1x в режиме PCPWM

Номер режима	COM1x1	COM1x0	Регистр OCR1x	Состояние вывода OC1x
2	1	0	Нижний предел счета (0x0000)	0
2	1	0	Верхний предел счета (TOP)	1
3	1	1	Нижний предел счета (0x0000)	1
3	1	1	Верхний предел счета (TOP)	0

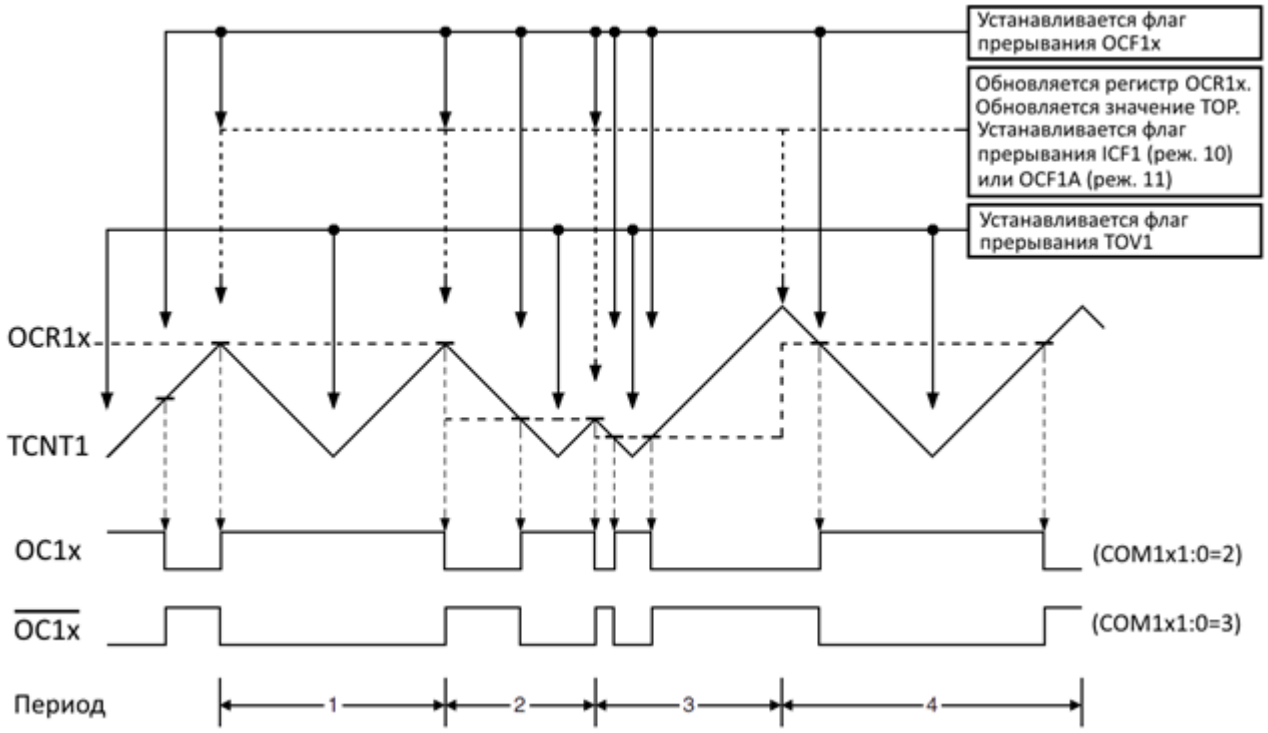


Рисунок 4.6 – Временные диаграммы формирования ШИМ-сигнала в режиме Phase Correct PWM таймера/счетчика TC1

В том же такте сигнала f_{TC1}, в котором обнуляется счетный регистр TCNT1, флаг прерывания по событию «Переполнение таймера/счетчика TC1» TOV1 устанавливается в лог. 1. При равенстве счетного регистра TCNT1 и регистра сравнения OCR1x флаг прерывания по событию «Совпадение x таймера/счетчика TC1» OCF1x устанавливается в лог. 1. Флаг прерывания по событию «Захват» ICF1 устанавливается в лог. 1 при изменении значения счетного регистра с верхнего предела счета, определяемого регистром ICR1, на значение 0x0000. Таким образом, в режимах 1–3 и 11 могут быть реализованы только прерывания по переполнению, совпадению A и совпадению B, а в режиме 10 – дополнительно и прерывание по захвату.

Режим работы Phase and Frequency Correct PWM (ШИМ с точной фазой и частотой/фазовой и частотной коррекцией).

Режим Phase and Frequency Correct PWM (PFCPWM) почти полностью совпадает с режимом Phase Correct PWM, но имеет одно принципиальное отличие – обновление содержимого регистра сравнения OCR1A (OCR1B) происходит в момент достижения счетным регистром TCNT1 нижнего предела счета 0x0000. Это объясняется тем, что в режиме Phase Correct PWM частое изменение верхнего предела счета TOP во время работы таймера/счетчика TC1 приводит к появлению на выходах генератора ШИМ-сигнала несимметричных импульсов относительно середины периода. Это следует из рисунка 4.6, на котором видно, что обновление содержимого регистра сравнения OCR1x происходит в момент достижения счетчиком максимального значения, период ШИМ-сигнала равен времени между этими моментами. При этом время обратного

счета определяется предыдущим значением верхнего предела счета TOP, а время прямого счета – новым значением. Если эти значения различны, то время прямого и время обратного счета также отличаются. Результатом этого и являются несимметричные импульсы, например, в третьем периоде сигнала.

Таким образом, при обновлении содержимого регистра сравнения в момент достижения счетным регистром нижнего предела счета время прямого счета всегда равно времени обратного счета, выходные импульсы симметричны, и соответственно частота генерируемого ШИМ-сигнала остается постоянной (рисунок 4.7).

Если же используется постоянное значение TOP, то будет отсутствовать разница между режимами Phase Correct PWM и Phase and Frequency Correct PWM.

В режиме PFCPWM верхний предел счета TOP определяется только регистром ICR1 или OCR1A. Разрешающую способность генератора ШИМ-сигнала можно также рассчитать по формуле (4.5) (таблица 4.13).

Таблица 4.13 – Разрешающая способность ШИМ-генератора в PFCPWM

Номер режима	WGM13	WGM12	WGM11	WGM10	Разрешающая способность	Верхний предел счета (TOP)
8	1	0	0	0	Переменная (2–16)	ICR1 (0x0003–0xFFFF)
9	1	0	0	1	Переменная (2–16)	OCR1A (0x0003–0xFFFF)

Режимы 8 и 9 обеспечивают более широкий диапазон регулировки частоты генерируемых ШИМ-сигналов благодаря изменению в широких пределах разрешающей способности ШИМ-генератора, т. е. изменению значений в регистре ICR1 или OCR1A.

Режимы 8 и 9 отличаются между собой тем, что если при генерации ШИМ-сигнала его частота меняется очень часто, то для задания верхнего предела счета необходимо использовать регистр сравнения OCR1A. В этом случае за счет двойной буферизации записи в регистр сравнения (для ICR1 буферизация не выполняется) исключается появление несимметричных импульсов сигнала на выходе ШИМ-генератора.

В режиме 9 формирование ШИМ-сигнала с переменой скважностью будет возможно только на выводе OC1B.

Режимы управления состоянием вывода OC1A (OC1B) в режиме Phase Correct PWM (PCPWM) представлены в таблице 4.14, а временные диаграммы формирования ШИМ-сигнала показаны на рисунке 4.7, где x означает канал A (B) ШИМ-генератора.

Таблица 4.14 – Режимы управления состоянием вывода OC1x в PFCPWM

Номер режима	COM1x1	COM1x0	Описание
0	0	0	Таймер/счетчик TC1 отключен от вывода OC1x
1	0	1	В режиме 9 состояние вывода OC1A меняется на противоположное при каждом равенстве счетного регистра TCNT1 и регистра сравнения OCR1A
2	1	0	Состояние вывода OC1x сбрасывается в лог. 0 при равенстве регистров TCNT1 и OCR1x во время прямого счета и устанавливается в лог. 1 при равенстве во время обратного счета (прямой режим)
3	1	1	Состояние вывода OC1x устанавливается в лог. 1 при равенстве регистров TCNT1 и OCR1x во время прямого счета и сбрасывается в лог. 0 при равенстве во время обратного счета (инверсный режим)

Частота генерируемого ШИМ-сигнала в герцах будет определяться по формуле

$$f_{OC1x.PFCPWM} = \frac{f_T}{2 \cdot k_{дел} \cdot TOP}, \quad (4.8)$$

где TOP – верхний предел счета, определяемый в соответствии с таблицей 4.13.

Вывод OC1x будет иметь устойчивое состояние в соответствии с таблицей 4.15, где x означает канал A (B) ШИМ-генератора.

Таблица 4.15 – Устойчивые состояния вывода OC1x в режиме PFCPWM

Номер режима	COM1x1	COM1x0	Регистр OCR1x	Состояние вывода OC1x
2	1	0	Нижний предел счета (0x0000)	0
2	1	0	Верхний предел счета (TOP)	1
3	1	1	Нижний предел счета (0x0000)	1
3	1	1	Верхний предел счета (TOP)	0

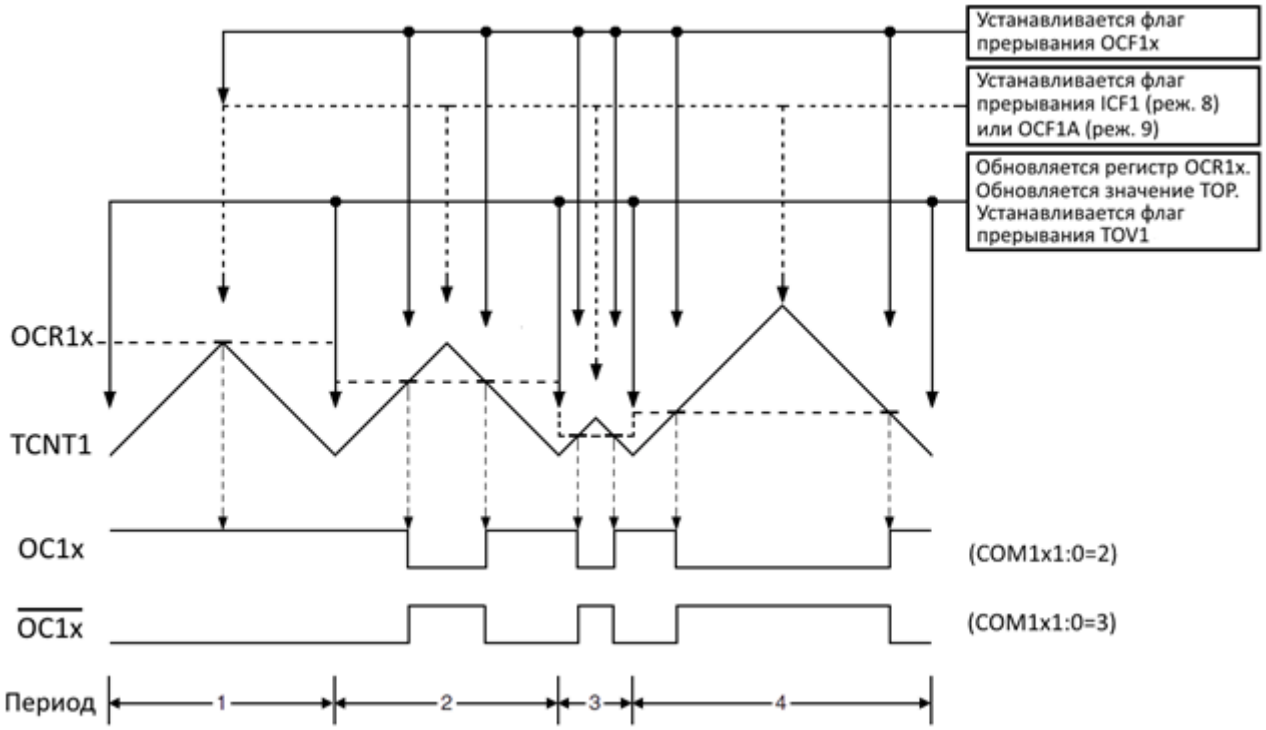


Рисунок 4.7 – Временные диаграммы формирования ШИМ-сигнала в режиме Phase and Frequency Correct PWM таймера/счетчика TC1

В том же такте сигнала f_{TC1}, в котором обнуляется счетный регистр TCNT1, флаг прерывания по событию «Переполнение таймера/счетчика TC1» TOV1 устанавливается в лог. 1. При равенстве счетного регистра TCNT1 и регистра сравнения OCR1x флаг прерывания по событию «Совпадение x таймера/счетчика TC1» OCF1x устанавливается в лог. 1. Флаг прерывания по событию «Захват» ICF1 устанавливается в лог. 1 при изменении значения счетного регистра с верхнего предела счета, определяемого регистром ICR1, на значение 0x0000. Таким образом, в режиме 9 могут быть реализованы только прерывания по переполнению, совпадению A и совпадению B, а в режиме 8 – дополнительно и прерывание по захвату.

1.3 Инициализация таймеров/счетчиков на формирование ШИМ-сигналов

Все три таймера/счетчика микроконтроллера ATmega16 имеют различные режимы формирования ШИМ-сигналов, которые могут быть получены без использования прерываний. Каждый из двух 8-разрядных таймеров/счетчиков поддерживает работу генератора ШИМ-сигналов только в одноканальном режиме. 16-разрядный таймер/счетчик содержит двухканальный ШИМ-генератор.

Ниже представлены примеры инициализации таймеров/счетчиков TC0 и TC1 для формирования ШИМ-сигнала без инверсии с фазовой коррекцией Phase Correct PWM и 8-разрядной разрешающей способностью на языке программирования C/C++ в среде разработки Atmel Studio.

Инициализация 8-разрядного таймера/счетчика TC0 имеет следующий вид:

```
1:    DDRB|=(1<<PB3);
2:    PORTB&=~(1<<PB3);
3:    TCNT0=0x00;
4:    OCR0=0x00;
5:    TCCR0=(0<<WGM01)|(1<<WGM00)|(1<<COM01)|(0<<COM00);
6:    TCCR0|=(0<<CS02)|(0<<CS01)|(1<<CS00);
```

Инициализация 16-разрядного таймера/счетчика TC1 имеет следующий вид:

```
1:    DDRD|=(1<<PD4)|(1<<PD5);
2:    PORTD&=~((1<<PD4)|(1<<PD5));
3:    TCNT1=0x00;
4:    OCR1A=0x00;
5:    OCR1B=0x00;
6:    TCCR1A=(1<<COM1A1)|(0<<COM1A0)|(1<<COM1B1)|(0<<COM1B0);
7:    TCCR1A|=(0<<WGM11)|(1<<WGM10);
8:    TCCR1B=(0<<WGM13)|(0<<WGM12)|(0<<CS12)|(0<<CS11)|(1<<CS10);
```

Так как ШИМ-сигнал таймера/счетчика TC0 будет генерироваться на выводе OC0 микроконтроллера, то необходимо третью линию порта B настроить на выход (вывод данных) с формированием на ней состояния лог. 0.

Так как ШИМ-сигналы таймера/счетчика TC1 будут генерироваться на выводах OC1A и OC1B микроконтроллера, то необходимо четвертую и пятую линии порта D соответственно настроить на выход (вывод данных) с формированием на них состояния лог. 0.

Счетный регистр и регистр сравнения таймеров/счетчиков должны быть обнулены.

Режим Phase Correct PWM задается разрядами WGM в соответствующем регистре управления согласно таблицам 3.7 и 3.13 (см. лабораторную работу №3). Для обоих таймеров/счетчиков это первый режим работы, т. е. WGM01:0=1 и WGM13:0=1.

ШИМ-сигнал без инверсии задается разрядами COM в соответствующем регистре управления согласно таблицам 4.4 и 4.11. Для обоих таймеров/счетчиков это второй режим, т. е. COM01:0=2, COM1A1:0=2 и COM1B1:0=2.

Частота формируемых ШИМ-сигналов будет зависеть от выбора тактовой частоты микроконтроллера и коэффициента деления в соответствии с формулами (4.3) и (4.7).

Значение коэффициента деления задается разрядами CSn2:0, где n – номер таймера/счетчика. Для получения максимальной частоты ШИМ-сигнала при заданной тактовой частоте микроконтроллера необходимо выбрать первый режим (без предделителя), т. е. CSn2:0=1.

В дальнейшем для формирования ШИМ-сигналов и управления их скважностью необходимо записывать требуемые значения в регистры сравнения OCR0, OCR1A, OCR1B. Эти значения должны находиться в диапазоне 0–255, потому что выбрана 8-разрядная разрешающая способность. Чем большим будет записанное значение, тем большую длительность будут иметь импульсы ШИМ-сигнала благодаря прямому режиму формирования ШИМ-сигнала. В инверсном режиме зависимость будет обратной.

1.4 Пример написания программы на C/C++

Требуется написать программу на языке программирования C/C++, выполнение которой микроконтроллером ATmega16 позволит сформировать ШИМ-сигнал с заданной частотой и скважностью с помощью канала A ШИМ-генератора 16-разрядного таймера/счетчика TC1. При этом необходимо предусмотреть выполнение следующих требований:

1) схема устройства в программе Proteus ISIS представлена на рисунке 4.8;

2) микроконтроллер должен работать от внутреннего источника тактирования;

3) параметры ШИМ-сигнала и тактовая частота микроконтроллера заданы в таблице 4.16;

4) необходимо использовать десятый режим работы таймера/счетчика TC1 и прямой режим управления состоянием вывода OC1A;

5) предусмотреть запуск и выключение генерации сигнала по тактовой кнопке SB1 с фиксацией своего состояния.

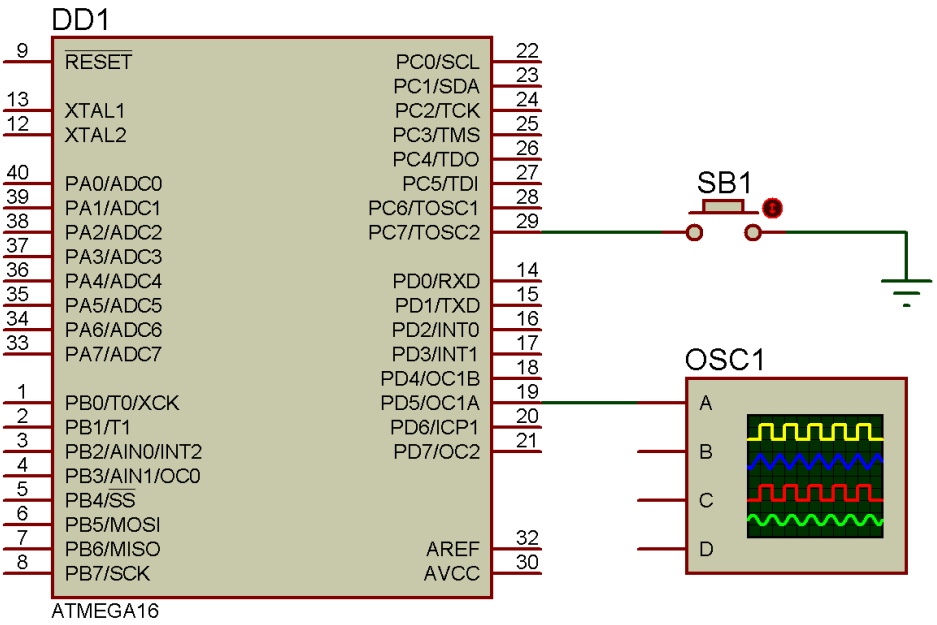


Рисунок 4.8 – Схема устройства, формирующего ШИМ-сигнал, с управлением по тактовой кнопке в программе Proteus ISIS

Таблица 4.16 – Исходные данные для примера

Частота ШИМ-сигнала, кГц	Скважность	Тактовая частота микроконтроллера, МГц
2,5	5	4

Десятый режим работы 16-разрядного таймера/счетчика ТС1 – это режим генерации ШИМ-сигнала с фазовой коррекцией (Phase Correct PWM), в котором в соответствии с таблицей 4.10 верхний предел счета TOP устанавливается регистром захвата ICR1, следовательно, частота ШИМ-сигнала зависит от значения ICR1, поэтому из формулы (4.7) выразим и рассчитаем значение регистра ICR1:

$$ICR1 = TOP = \frac{f_T}{2 \cdot k_{дел} \cdot f_{OC1A.PCPWM}} = \frac{4\,000\,000}{2 \cdot k_{дел} \cdot 2500}. \quad (4.9)$$

Как видно из формулы (4.9), значение регистра ICR1 может принимать разные значения в зависимости от коэффициента деления предделителя, который для таймера/счетчика ТС1 имеет фиксированные значения: 1, 8, 64, 256, 1024. Рассчитаем для каждого коэффициента значение регистра ICR1 (таблица 4.17).

Таблица 4.17 – Значения регистра ICR1 в зависимости от коэффициента деления

Коэффициент деления предделителя таймера/счетчика ТС1	1	8	64	256	1024
Значение регистра ICR1	800	100	12,5	3,125	0,78125

В соответствии с таблицей 4.17 выбираем коэффициент деления $k_{дел} = 8$ и значение регистра ICR1=100 из следующих соображений:

- все регистры таймера/счетчика ТС1 могут содержать только целочисленные значения;
- чем меньше тактовая частота f_{TC1}, тем меньше энергопотребление таймером/счетчиком ТС1.

Так как регистр ICR1 определяет верхний предел счета, то в соответствии с рисунком 4.6 этот регистр отвечает за период повторения импульсов T, а момент совпадения значений регистров ICR1 и OCR1x определяет спадающий и нарастающий фронты импульса ШИМ-сигнала, т. е. соответственно OCR1x отвечает за длительность импульса τ, поэтому скважность будет рассчитываться по формуле

$$s = \frac{T}{\tau} = \frac{ICR1}{OCR1x}. \quad (4.10)$$

Из формулы (4.10) выразим и рассчитаем значение регистра OCR1A:

$$\mathrm{OCR1A} = \frac{\mathrm{ICR1}}{\mathrm{s}} = \frac{100}{5} = 20. \qquad (4.11)$$

Блок-схема алгоритма работы программы представлена на рисунке 4.9.

Алгоритм работы программы не содержит процедуру обработки прерывания, т. е. генерация ШИМ-сигнала осуществляется без необходимости использования прерываний.

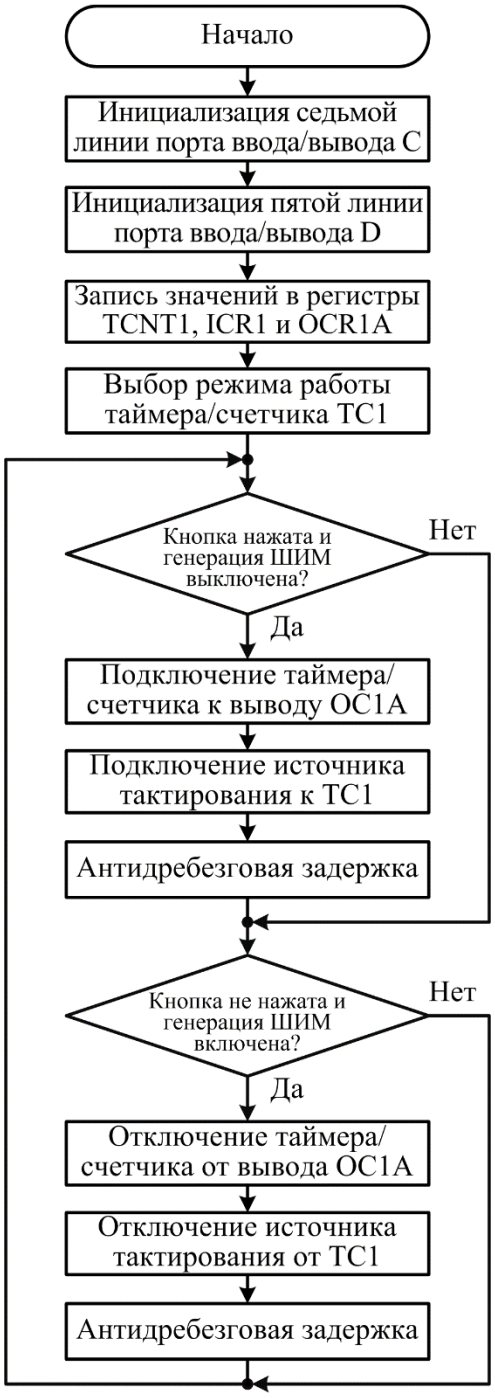


Рисунок 4.9 – Блок-схема алгоритма работы программы для примера

В соответствии с алгоритмом работы программа начинается с инициализации седьмой линии порта ввода/вывода C, которая заключается в настройке этой линии на вход (ввод данных) с подключением внутреннего подтягивающего резистора МК, потому что к этой линии подключено устройство ввода – тактовая кнопка без внешнего подтягивающего резистора, который необходим для проверки работы кнопки. Затем выполняется инициализация пятой линии порта ввода/вывода D, которая заключается в настройке этой линии на выход (вывод данных) с формированием на ней состояния лог. 0, потому что на этой линии в дальнейшем будет сформирован требуемый ШИМ-сигнал с помощью 16-разрядного таймера/счетчика TC1. Дальше выполняется запись значений в счетный регистр TCNT0, регистр захвата ICR1 и регистр сравнения OCR1A, которые обеспечат получение требуемых параметров ШИМ-сигнала.

Затем задается десятый режим работы таймера/счетчика TC1, а именно режим генерации ШИМ-сигнала с фазовой коррекцией (Phase Correct PWM).

Далее находится первый блок ветвления, в котором проверяется выполнение двух условий: нажата ли тактовая кнопка SB1 и выключена ли генерация ШИМ-сигнала. Только при выполнении обоих условий осуществляется переход по ветви «Да» на следующие три блока команд. В противном случае реализуется переход по ветви «Нет», и блоки команд будут пропущены.

При переходе по ветви «Да» вначале задается второй режим управления состоянием вывода OC1A. В результате чего таймер/счетчик TC1 будет подключен к выводу OC1A микроконтроллера, который является альтернативной функцией работы пятой линии порта ввода/вывода D. При этом базовая функция вывода МК, как линия порта ввода/вывода D, будет недоступна. В следующем блоке алгоритма осуществляется запуск таймера/счетчика TC1 в результате подключения к нему источника тактового сигнала. В результате выполнения этих команд на выводе OC1A с помощью таймера/счетчика TC1 будет сгенерирован требуемый ШИМ-сигнал. Затем осуществляется задержка по времени для борьбы с дребезгом контактов тактовой кнопки SB1 при ее нажатии.

Далее находится второй блок ветвления, в котором проверяется выполнение двух других условий: не нажата (отпущена) ли тактовая кнопка SB1 и включена ли генерация ШИМ-сигнала. Если оба условия выполнены, то осуществляется переход по ветви «Да». Вначале таймер/счетчик TC1 будет отключен от вывода OC1A, а затем таймер/счетчик TC1 будет остановлен в результате отключения от него источника тактирования. Затем выполняется задержка по времени для борьбы с дребезгом контактов тактовой кнопки SB1 при ее отпускании.

В обоих блоках ветвления проверяются два условия, для того чтобы блоки команд в ветви «Да» выполнялись только один раз после нажатия и отпускания тактовой кнопки SB1.

Исходный текст программы на языке программирования C/C++ для рассмотренной блок-схемы представлен в листинге 4.1 и заключается в следующем.

В строках 1–6 с помощью директивы *#define* определены символьные константы:

- F_CPU обозначает тактовую частоту микроконтроллера. В соответствии с заданием указана частота 4 МГц;

- F_PWM обозначает требуемую частоту ШИМ-сигнала. В соответствии с заданием указана частота 2,5 кГц;

- K_DEL обозначает коэффициент деления блока предделителя таймера/счетчика TC1. В соответствии с таблицей 4.17 указан коэффициент деления, равный 8;

- ICR1_VALUE обозначает значение, которое необходимо записать в регистр захвата ICR1. Для расчета значения использована формула (4.9). Величина значения составит

$$\text{ICR1\_VALUE} = \frac{\text{F\_CPU}}{2 \cdot \text{K\_DEL} \cdot \text{F\_PWM}} = \frac{4\,000\,000}{2 \cdot 8 \cdot 2500} = 100; \qquad (4.12)$$

- S обозначает требуемую скважность ШИМ-сигнала. В соответствии с заданием указано значение 5;

- OCR1A_VALUE обозначает значение, которое необходимо записать в регистр сравнения OCR1A. Для расчета значения использована формула (4.11). Величина значения составит

$$\text{OCR1A\_VALUE} = \frac{\text{ICR1\_VALUE}}{\text{S}} = \frac{100}{5} = 20. \qquad (4.13)$$

В строке 7 программы с помощью директивы *#include* подключается стандартный заголовочный файл, который в свою очередь подключает библиотеку идентификаторов для используемой модели микроконтроллера, выбранной при создании проекта в среде разработки Atmel Studio. В строке 8 подключается библиотека *delay.h*, содержащая различные функции задержки по времени.

В строке 9 начинается главная функция программы.

В строке 11 определена локальная переменная беззнакового символьного типа *pwmState*, которая играет роль флага состояния генерации ШИМ-сигнала. Переменная проинициализирована значением 0, которое означает, что ШИМ-сигнал не генерируется, а запись значения 1 – ШИМ-сигнал генерируется.

В строке 12 выполняется настройка седьмой линии порта C на вход с помощью записи лог. 0 в седьмой разряд регистра DDRC. В строке 13 осуществляется подключение внутреннего подтягивающего резистора МК к седьмой линии порта C с помощью записи лог. 1 в седьмой разряд регистра PORTC. Эти команды необходимы для правильной работы тактовой кнопки SB1.

В строке 14 осуществляется настройка пятой линии порта D на выход с помощью записи лог. 1 в пятый разряд регистра DDRD, а в строке 15 формирование на линии состояния лог. 0 с помощью записи лог. 0 в пятый разряд реги-

стра PORTD. Эти команды необходимы для отображения формируемого ШИМ-сигнала на выводе OC1A.

В строке 16 осуществляется запись нулевого значения в счетный регистр TCNT1.

В строке 17 выполняется запись значения символьной константы ICR1_VALUE в регистр захвата ICR1, которое отвечает за период повторения импульсов ШИМ-сигнала, а в строке 18 – запись значения символьной константы OCR1A_VALUE в регистр сравнения OCR1A, которое отвечает за длительность импульсов ШИМ-сигнала.

В строках 19 и 20 в результате записи в регистры управления A и B выбирается режим работы Phase Correct PWM (режим WGM13:0=10). При этом источник тактового сигнала (режим CS12:0=0) и способ управления состоянием вывода OC1A (режим COM1A1:0=0) не заданы, поэтому таймер/счетчик TC1 будет оставаться в выключенном состоянии и не будет подключен к выводу OC1A.

В строке 21 начинается бесконечный цикл, в котором в строке 23 проверяется двойное условие: если тактовая кнопка SB1 нажата и ШИМ-сигнал еще не генерируется, то выполняется блок команд в строках 24–29, иначе эти команды будут пропущены.

Выполнение команды в строке 25 подключает таймер/счетчик TC1 к выводу OC1A микроконтроллера (режим COM1A1:0=2), а выполнение команды в строке 26 приводит к запуску таймера/счетчика на тактовой частоте, равной частоте микроконтроллера, деленной на 8 (режим CS12:0=2).

В строке 27 выполняется установка флага состояния генерации ШИМ-сигнала с помощью записи единицы в переменную *pwmState*. Это приведет к тому, что на следующей итерации бесконечного цикла одно условие в строке 23 не будет выполнено. Затем в строке 28 реализуется задержка по времени для борьбы с дребезгом контактов тактовой кнопки SB1. Величина 200 мс обусловлена работой тактовой кнопки в программе Proteus ISIS. Для каждой тактовой кнопки длительность дребезга контактов будет определяться конкретным конструктивным исполнением и в большинстве случаев составлять гораздо меньшую величину.

В строке 30 проверяется другое двойное условие: если тактовая кнопка SB1 отпущена и ШИМ-сигнал уже генерируется, то выполняется блок команд в строках 31–36, иначе команды будут пропущены. В этих строках реализуется отключение таймера/счетчика TC1 от вывода OC1A (строка 32), отключение источника тактирования, в результате чего таймер/счетчик TC1 будет остановлен (строка 33), сброс флага состояния генерации ШИМ-сигнала (строка 34) и антидребезговая задержка (строка 35).

В строках 23 и 30 проверяются по два условия, для того чтобы соответствующие блоки команд выполнялись только один раз после нажатия и отпускания тактовой кнопки SB1.

Листинг 4.1

```
1:   #define F_CPU          4000000UL
2:   #define F_PWM          2500
3:   #define K_DEL          8
4:   #define ICR1_VALUE     F_CPU/2/K_DEL/F_PWM
5:   #define S              5
6:   #define OCR1A_VALUE    ICR1_VALUE/S
7:   #include <avr/io.h>
8:   #include <util/delay.h>
9:   int main(void)
10:  {
11:      unsigned char pwmState=0;
12:      DDRC&=~(1<<PC7);
13:      PORTC|=(1<<PC7);
14:      DDRD|=(1<<PD5);
15:      PORTD&=~(1<<PD5);
16:      TCNT1=0x00;
17:      ICR1=ICR1_VALUE;
18:      OCR1A=OCR1A_VALUE;
19:      TCCR1A=(1<<WGM11)|(0<<WGM10);
20:      TCCR1B=(1<<WGM13)|(0<<WGM12);
21:      while (1)
22:      {
23:          if (((PINC & (1<<PC7))==0) && (pwmState==0))
24:          {
25:              TCCR1A|=(1<<COM1A1)|(0<<COM1A0);
26:              TCCR1B|=(0<<CS12)|(1<<CS11)|(0<<CS10);
27:              pwmState=1;
28:              _delay_ms(200);
29:          }
30:          if (((PINC & (1<<PC7))!=0) && (pwmState==1))
31:          {
32:              TCCR1A&=~((1<<COM1A1)|(0<<COM1A0));
33:              TCCR1B&=~((0<<CS12)|(1<<CS11)|(0<<CS10));
34:              pwmState=0;
35:              _delay_ms(200);
36:          }
37:      }
38:  }
```

На рисунке 4.10 представлена осциллограмма ШИМ-сигнала на выводе OC1A, полученная для рассмотренного листинга в программе Proteus ISIS.

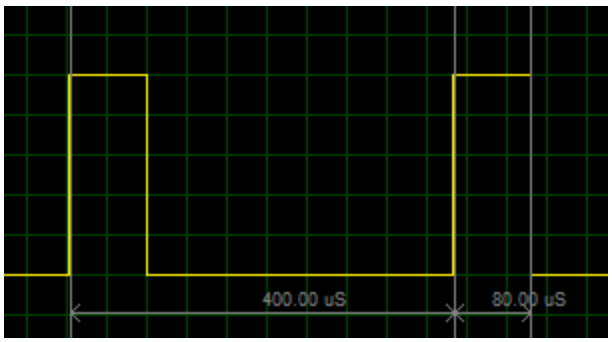


Рисунок 4.10 – Осциллограмма ШИМ-сигнала на выводе OC1A

Как видно из рисунка 4.10, период повторения импульсов T составляет 400 мкс, а длительность импульса τ – 80 мкс, поэтому рассчитаем частоту ШИМ-сигнала и скважность по следующим формулам:

$$f = \frac{1}{T} = \frac{1}{400 \cdot 10^{-6}} = 2500 \text{ (Гц)}, \tag{4.14}$$

$$s = \frac{T}{\tau} = \frac{400}{80} = 5. \tag{4.15}$$

Рассчитанные значения совпадают с требуемыми параметрами ШИМ-сигнала в таблице 4.16, следовательно, программа написана правильно.

2 Порядок выполнения работы

Задание 1

Требуется написать программу на языке программирования C/C++, выполнение которой микроконтроллером ATmega16 позволит управлять яркостью свечения одноцветного светодиода с помощью двух тактовых кнопок, отвечающих соответственно за увеличение и уменьшение яркости, и с использованием определенного таймера/счетчика, работающего в заданном режиме формирования ШИМ-сигналов. При этом необходимо предусмотреть выполнение следующих условий и требований:

1) исследуемая схема устройства в программе Proteus ISIS представлена в приложении Г. В качестве управляемого светодиода используется один из трех цветных светодиодов, находящихся в одном корпусе RGB-светодиода с общим катодом. В схеме использована пользовательская модель RGB-светодиода, библиотеку которую необходимо дополнительно установить в директории программы Proteus ISIS;

2) микроконтроллер должен работать от внутреннего источника тактирования;