Часть I. 8-разрядные

МИКРОКОНТРОЛЛЕРЫ

Руководство к практикуму

КФУ - Казань - 2014

УДК 681.3.068

Печатается по решению редакционно-издательского совета

Института физики

Казанского федерального универсистета

Рецензенты

Гумеров Р.И.

Программируемые микроэлектронные системы. Лабораторный практикум.

Часть I. 8-разрядные микроконтроллеры. Руководство. – Казань: КФУ, 2014, - 74 стр.

Аннотация:

Руководство предназначено для практического освоения современных микроконтроллеров с архитектурой AVR .Основное внимание уделяется разработке приложений, использующих периферийные устройства и порты ввода/вывода. Приводится описание архитектуры, структуры, специфики системы команд для контроллеров типа ATmega и XMEGA, а также инструментария разработки приложений AVRStudio4 и AmelStudio6. Даны примеры построения приложений. Для студентов, обучающихся по направлению 011800.62

Оглавление

ПРОГРАММИРУЕМЫЕ МИКРОЭЛЕКТРОННЫЕ СИСТЕМЫ

ВВЕДЕНИЕ

Одной из главных составляющих современных информационных технологий являются программируемые микроэлектронные системы. Их еще называют системами на кристалле - SoC - **S**ystem **o**n **C**hip. SoC – это система на кристалле (или однокристальная система), выполняющая функцию целого устройства и размещенная на одной интегральной схеме (ИС). В зависимости от назначения она может оперировать цифровыми сигналами, аналоговыми сигналами, их совокупностью, а также радиосигналами. Как правило, SoC применяются в портативных и встраиваемых системах.

Типичная SoC содержит:

- один или несколько микроконтроллеров, микропроцессоров или ядер цифровой обработки сигналов (DSP). SoC, содержащий несколько процессоров, называют *многопроцессорной системой на кристалле* (MPSoC);
- систему памяти, состоящую из блоков ОЗУ, СППЗУ или флэш-памяти;
- источники тактирования, например, кварцевые резонаторы и схемы ФАПЧ (фазовой автоподстройки частоты);
- таймеры, счетчики, широтно-импульсные модуляторы;
- блоки цифро-аналоговых и аналого-цифровых преобразователей.
- регуляторы напряжения и стабилизаторы питания;
- блоки, реализующие порты ввода/вывода, а также стандартные интерфейсы для подключения внешних устройств: USB, Ethernet, RS485, CAN...

В SoC часто включают блоки программируемых логических матриц (FPGA, например), а в программируемые аналого-цифровые системы еще и программируемые аналоговые блоки.

В рамках Программы развития КФУ учебная «Лаборатория программируемых микроэлектронных систем» кафедры радиоэлектроники

была оснащена целым набором современных устройств, относящихся к классу систем на кристалле. Это стенды с микроконтроллерами различных типов: с архитектурами AVR, ARM, xMOS, стенды с программируемыми логическими приборами (PLD) от фирмы ALTERA (Cyclone II Cyclone IV), а также с процессорами цифровой обработки сигналов (DSP) от Analog Devices (SHARC, Blackfin, TigerSharc).

Первый практикум посвящается изучению микроконтроллеров и среды разработки приложений, причем он разделен на две части: в первой части рассматриваются восьмиразрядные микроконтроллеры, а во второй – наиболее современные и интересные тридцати двух разрядные микроконтроллеры с архитектурами ARM и XMOS.

Второй практикум будет связан с освоением инструментария для разработки приложений ЦОС на цифровых сигнальных процессорах и PLD.

МИКРОКОНТРОЛЛЕРЫ

Микроконтроллеры отличаются от микропроцессоров тем, что они предназначены для управления различными системами. При относительно скромном вычислительном ядре микроконтроллеры включают в себя много дополнительных блоков, обеспечивающих взаимодействие с внешними устройствами: память, порты ввода/вывода, таймеры, контроллеры прерываний и прямого доступа к памяти, устройства аналогового ввода/вывода и многое другое. Для построения полностью функционирующего устройства достаточно единственной микросхемы микроконтроллера.

Часть I. Восьмиразрядные микроконтроллеры

Настоящий раздел практикума начинается с освоения микроконтроллеров AVR фирмы Atmel, как наиболее простых и, в то же время, весьма широко распространенных. Эти устройства построены на основе RISC микропроцессора с *Гарвардской* архитектурой и разнообразными ресурсами для ввода/вывода (в том числе и аналоговыми). В основе лабораторной установки - отладочный набор AS-megaM, который в сочетании с прилагаемым пакетом разработчика «AVR-Studio 4» и программатором позволяет создавать рабочие приложения. Язык ассемблера для данного типа микропроцессоров весьма нагляден и прост, и в сочетании со средой «AVR-Studio 4» дает возможность пользователю детально разобраться, как работают ядро процессора и периферийные устройства.

1.1. Общие сведения

Микроконтроллеры AVR представляют собой однокристальные 8-ми разрядные RISC-контроллеры (**R**educed **I**nstruction **S**et **C**omputer – компьютер с сокращённым набором команд), обладающие программируемой FLASH – памятью и EPROM памятью (**E**rasable **P**rogrammable **R**ead **O**nly **M**emory – стираемое программируемое постоянное запоминающее устройство),

выпускаемые фирмой ATMEL (**A**dvanced **T**echnology **Me**mory and **L**ogic). Название AVR они получили в честь двух студентов, участвовавших в разработке микропроцессоров и выдвинувших идею восьмиразрядного RISC ядра: Альфа Богена и Вергарда Воллена. **A**lf + **V**ergard +**R**isc.

Представители семейства AVR обладают ограниченным набором из 118 высокоэффективных команд. Благодаря особой архитектуре (рис.1.1.1), в вычислениях, кроме накапливающего сумматора, задействованы 32 равноправных рабочих регистра, напрямую связанных с арифметико-логическим устройством, что исключает необходимость после завершения вычислительных операций обращения к вспомогательным регистрам или промежуточным запоминающим устройствам. Микроконтроллеры AVR построены в соответствии с Гарвардской архитектурой, что подразумевает разделение памяти на две части: для программ и для данных; обращение к каждой части по своей шине. AVR-микроконтроллеры используют одноступенчатую конвейерную обработку. Это означает, что во время выполнения текущей команды выполняется загрузка следующей команды из памяти программ, благодаря чему достигается возможность выполнения команды в течение одного тактового цикла. Поэтому, при использовании в качестве системного тактирования тактовой частоты непосредственно сгенерированной кварцевым осциллятором (у большинства процессоров и контроллеров для получения собственно системного такта она делится на заданный коэффициент) достигается быстродействие равное частоте используемого кварца. Так при работе с частотой кварца 7 МГц получаем около 7 миллионов выполненных команд в течение одной секунды (0,14 микросекунды на команду).

За немногими исключениями, в которых нарушается конвейер (первая команда в программе, команды условного и безусловного перехода, входы и выходы в прерывания и процедуры …), микроконтроллеры семейства AVR

действительно обрабатывают все команды в течение единственного системного такта.

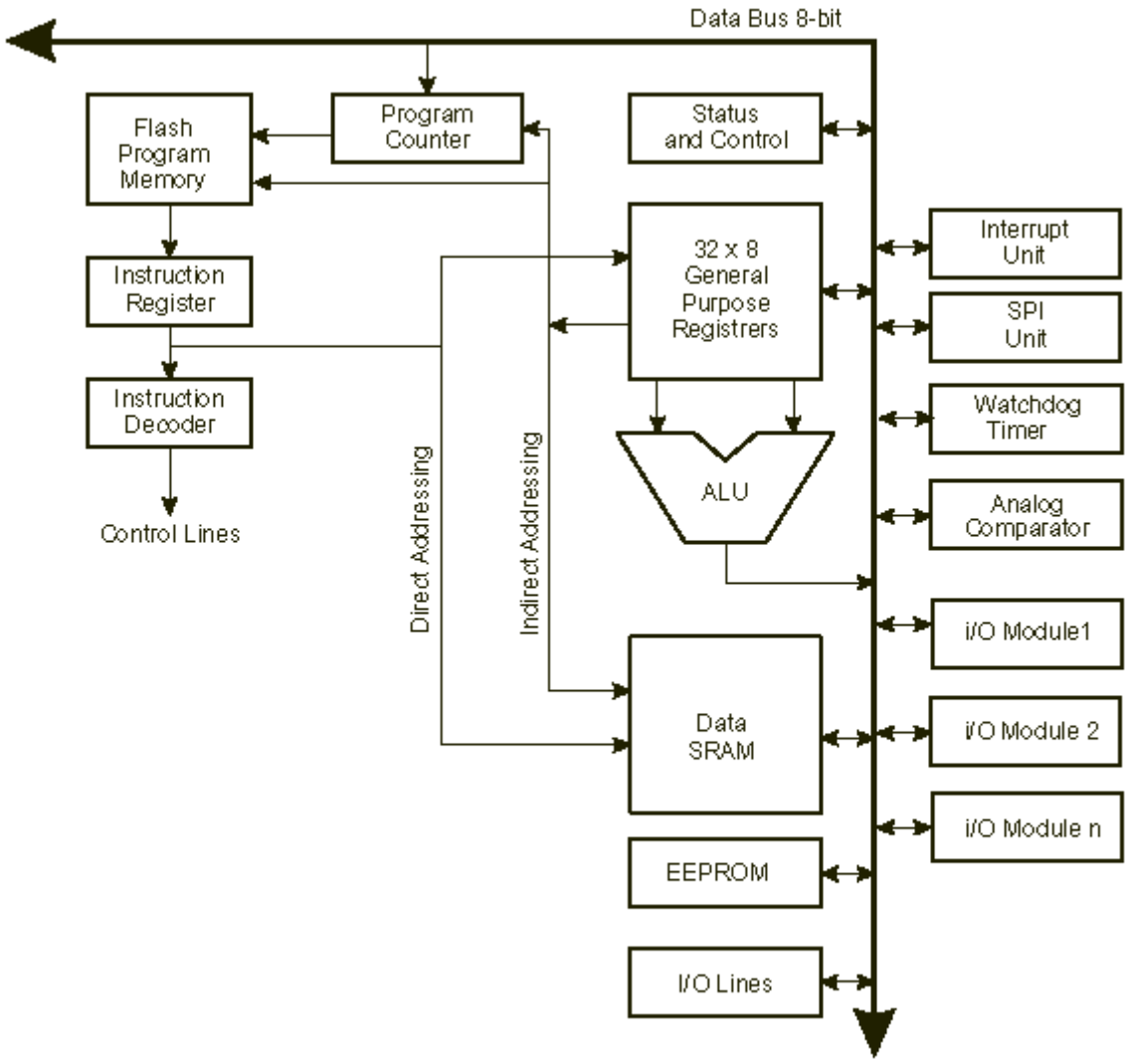


Рис.1.1.1. Архитектура ядра контроллера AVR

В рамках единой базовой архитектуры (рис.1.1.1) микроконтроллеры AVR подразделяются на несколько семейств. Мы остановимся на наиболее функциональных и производительных: *MEGA и XMEGA*. Представители различных семейств AVR различаются степенью развития периферии, объёмом памяти программ и данных, причём контроллеры семейства *MEGA* имели лучшие показатели, а впоследствии наиболее более функционально продвинутыми стали контроллеры *XMEGA (*но об этом ниже). Микроконтроллеры каждого семейства поддерживают несколько режимов

энергопотребления, имеют блок прерываний, сторожевой таймер и допускают программирование непосредственно в готовом устройстве.

1.2. Микроконтроллеры семейства Mega

Как и все микроконтроллеры AVR представители семейства MEGA являются 8-ми разрядными микроконтроллерами, предназначенными для встраиваемых приложений. Они имеют электрически стираемую память программ (FLASH) и данных (EEPROM), а также разнообразные периферийные устройства. Изготавливаются они по мало потребляющей КМОП – технологии, которая в сочетании с усовершенствованной RISC – архитектурой позволяет достичь наилучшего соотношения быстродействие/энергопотребление. Микроконтроллеры данного семейства относятся к наиболее развитым представителям МК AVR (рис.1.2.1).

Отличительные особенности:

● FLASH – память объёмом 8…128 Кбайт (128Кбайт для ATmega128). Число циклов стирания/записи не менее 10000.

● Оперативная память (статическое ОЗУ) объёмом 1…4Кбайт (4Кбайт для ATmega128).

● Память данных на основе EEPROM объёмом 512байт…4Кбайт (4Кбайт для ATmega128).Число циклов стирания/записи не менее 100000.

● Возможность защиты от чтения и модификации памяти программ и данных.

● Возможность программирования непосредственно в системе через последовательные интерфейсы SPI и JTAG.

● Возможность самопрограммирования.

● Возможность внутрисхемной отладки в соответствии со стандартом IEEE 1149.1 (JTAG).

● Различные способы синхронизации: встроенный RC-генератор с внутренней или внешней времязадающей RC- цепочкой или с внешним резонатором (пьезокерамическим или кварцевым), внешний сигнал синхронизации.

● Наличие нескольких режимов пониженного энергопотребления.

● Наличие детектора снижения напряжения питания.

● Возможность программного снижения частоты тактового генератора.

Периферийные устройства микроконтроллера ATmega128:

● два 8-ми разрядных таймера/счётчика;

● два 16-ти разрядных таймера/счётчика;

● сторожевой таймер;

● аналоговый компаратор;

● многоканальный 10-ти разрядный АЦП;

● полнодуплексный универсальный синхронный/асинхронный приёмопередатчик;

● последовательный синхронный интерфейс;

● последовательный двухпроводной интерфейс.

1.2.1. Центральный процессор и память

Характеристики процессора:

● полностью статическая архитектура (минимальная тактовая частота равна нулю);

● АЛУ подключено непосредственно к регистрам общего назначения;

● большинство команд выполняется за один машинный цикл;

● многоуровневая система прерываний;

● поддержка очереди прерываний;

● 27 источников прерываний, из них 8 внешних;

● наличие программного стека;

● наличие аппаратного умножителя;

Как уже было отмечено, в микроконтроллерах AVR семейства MEGA реализована Гарвардская архитектура, в соответствии с которой разделены не только адресные пространства памяти программ и памяти данных, но и шины доступа к ним (рис.1.2.1). То есть способы адресации и доступа к этим областям памяти также различны. Такая структура позволяет центральному процессору работать одновременно и с памятью программ, и с памятью

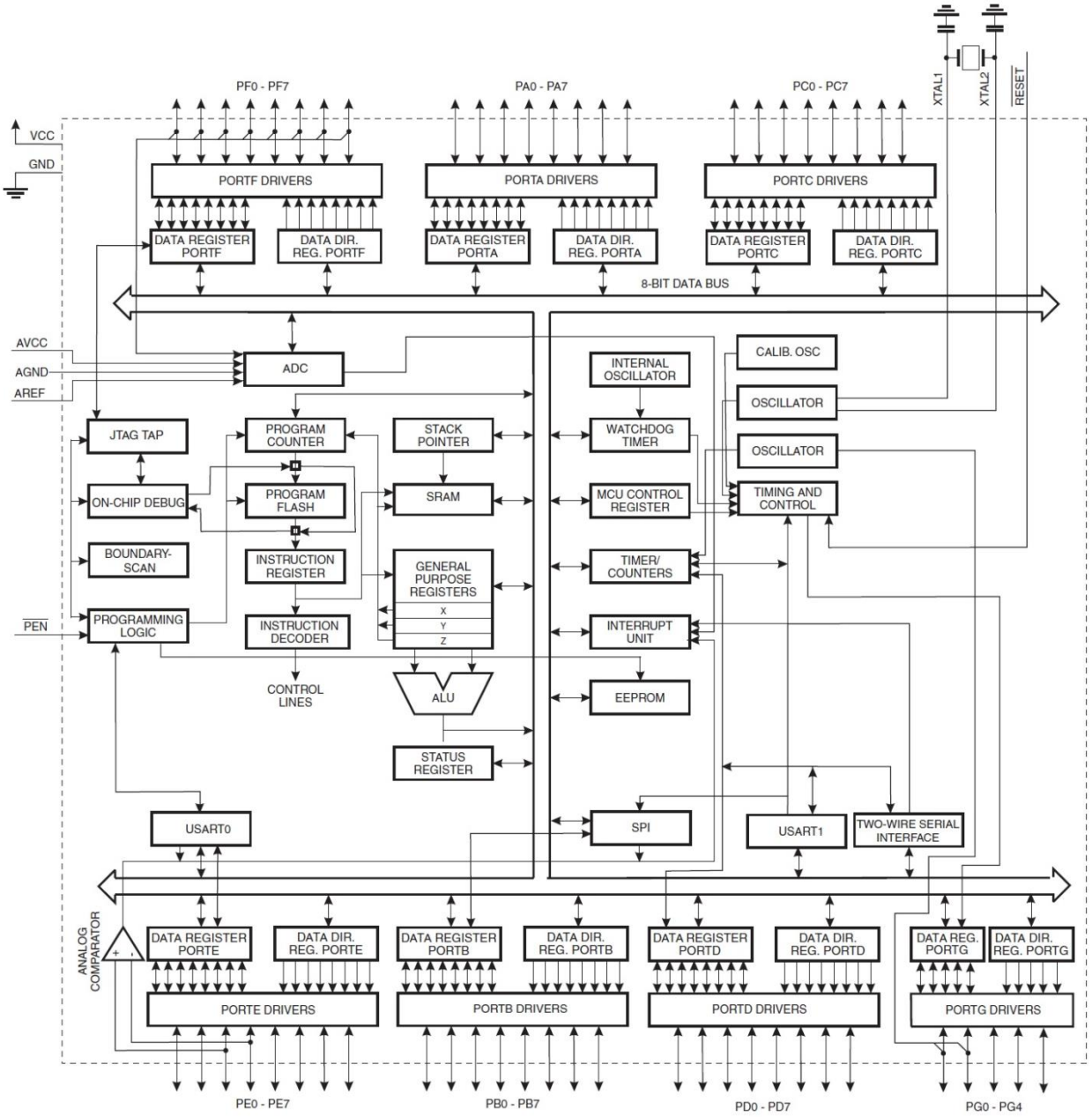


Рис. 1.2.1. Функциональная блок-схема МК ATmega128

данных, что существенно увеличивает производительность.

Память программ предназначена для хранения команд, управляющих функционированием микроконтроллера, и таблиц констант, не меняющихся во время работы. Как уже было сказано, она представляет собой электрически стираемое ППЗУ (FLASH – ПЗУ). В связи с тем что длина всех команд кратна одному 16-ти разрядному слову, память программ имеет 16-разрядную организацию. Соответственно, объём памяти микроконтроллера

ATmega128 составляет 64×1024 16-разрядных слов. Карта памяти приведена на рис.1.2.2.

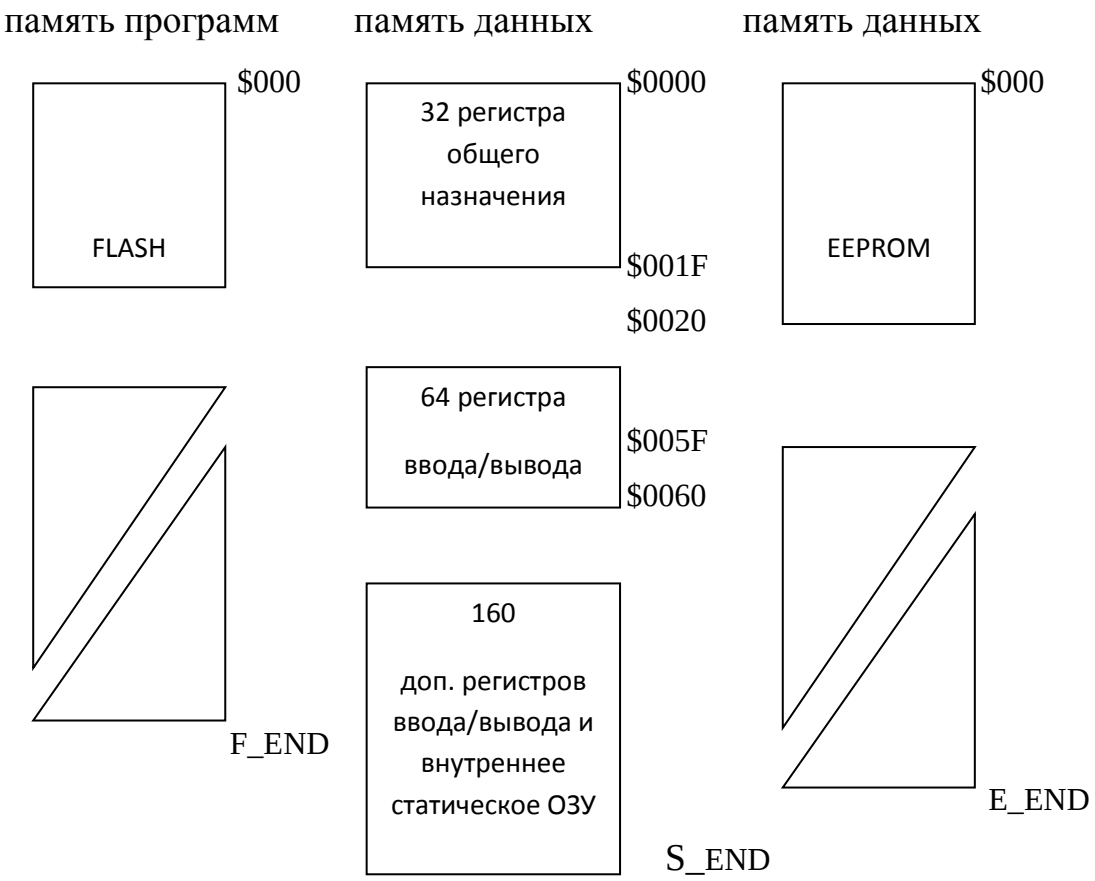


Рис.1.2.2. Карта памяти ATMega 128.

Для адресации памяти программ используется счётчик команд (PC – Program Counter). В ATmega128 размер счётчика команд составляет 16 бит. При нормальном выполнении программ содержимое счётчика команд автоматически увеличивается на 1 или на 2 (в зависимости от размера выполняемой команды) в каждом машинном цикле. Этот порядок нарушается при выполнении команд перехода, вызова и возврата из подпрограмм, а также при возникновении прерываний. По адресу $0000 памяти программ находится вектор «сброса», то есть после сброса микроконтроллера выполнение программы начинается с этого адреса, и по нему должна размещаться команда перехода к инициализационной части программы. Начиная с адреса $0001 памяти программ, располагается таблица

векторов прерываний. При возникновении прерывания после сохранения в стеке текущего значения счётчика команд происходит выполнение команды, расположенной по адресу соответствующего вектора. По этим адресам должны располагаться команды перехода к подпрограммам обработки соответствующих прерываний.

Память данных микроконтроллеров семейства MEGA организована линейно и разделена на три части: регистровую память, оперативную память (статическое ОЗУ) и энергонезависимую ЭСППЗУ – электрически стираемое программируемое постоянное запоминающее устройство (EEPROM). Регистровая память включает 32 регистра общего назначения (РОН), объединённых в файл, служебные регистры ввода/вывода (РВВ) и область дополнительных регистров ввода/вывода (ДРВВ). Под РВВ в памяти контроллера отводится 64 байт, под ДРВВ – 160 байт.

Все регистры общего назначения объединены в регистровый файл быстрого доступа. Как уже было сказано, в микроконтроллерах AVR все 32 РОН непосредственно доступны АЛУ, а это позволяет использовать любой РОН практически во всех командах и как операнд-источник и как операнд-приёмник. *Такое решение (в сочетании с конвейерной обработкой) позволяет АЛУ выполнять действия, связанные с извлечением операндов из регистрового файла, выполнением команды и записью результата обратно в регистровый файл за один машинный цикл.*

В обеих областях регистров ввода/вывода располагаются различные служебные регистры (регистр управления микроконтроллером, регистр состояния и т.п.), а также регистры управления периферийными устройствами, входящими в состав микроконтроллера. Эти регистры располагаются в так называемом пространстве ввода/вывода, то есть существуют команды быстрого доступа к ним, использующие отдельную адресацию. Введение дополнительных РВВ связано с тем, что для поддержки всех периферийных устройств ATmega128 обычных 64-х РВВ не хватает.

Для хранения переменных прикладной программы помимо РОН может использоваться статическое ОЗУ объёмом 4 КБ (для ATmega128). В адресном пространстве ОЗУ также расположены все регистры микроконтроллера, под них отведены младшие 256 адресов. Это позволяет обращаться к РОН и РВВ как к ячейкам ОЗУ, хотя физически они ими не являются. Такое решение увеличивает эффективность подсистемы адресации.

Для долговременного хранения различной информации в микроконтроллере используется EEPROM – память, размер которой у ATmega128 равен 4Кб. Эта память расположена в отдельном адресном пространстве, а доступ к ней осуществляется с помощью определённых РВВ.

1.2.2. Порты ввода/вывода

Характеристики подсистемы ввода/вывода:

- программное конфигурирование и выбор портов ввода/вывода;
- выводы могут быть запрограммированы как входные или как выходные независимо друг от друга;
- входные буферы с триггером Шмидта на всех выводах;
- возможность подключения ко всем входам внутренних подтягивающих резисторов (35…120 кОм).

Каждый порт микроконтроллера состоит из определённого числа выводов, через которые он может осуществлять приём и передачу цифровых сигналов. Задание направления передачи данных через любой контакт ввода/вывода может быть произведено программно в любой момент времени.

Выходные буферы всех портов, имея симметричные нагрузочные характеристики, обеспечивают высокую нагрузочную способность при любом уровне сигнала.

Входные буферы построены по схеме триггера Шмидта. Для всех входов имеется возможность подключения внутреннего подтягивающего резистора между входом и шиной питания. *Отличительной особенностью портов микроконтроллеров AVR при использовании их в качестве цифровых*

портов ввода/вывода общего назначения является возможность выполнять действия с любым выводом (с помощью команд CBI и SBI), не влияя на другие выводы порта. Это относится к изменению режима работы контакта ввода/вывода, к изменению состояния выходного буфера (для выходов) и к изменению состояния внутреннего подтягивающего резистора (для входов).

Микроконтроллер ATmega128 имеет шесть 8-разрядных портов ввода/вывода и один 5-разрядный, то есть всего 53 контакта ввода/вывода. Управление, конфигурирование и обращение к портам ввода/вывода осуществляется с помощью соответствующих регистров ввода/вывода (РВВ).

В ATmega128 предусмотрены 8 внешних прерываний, связанных с контактами ввода/вывода и несколько условий их генерации: по низкому уровню, спаду или фронту сигнала на выводе. Управление обработкой прерываний от контактов ввода/вывода осуществляется при помощи соответствующих РВВ.

Необходимо отметить, что линии портов ввода/вывода могут выполнять несколько функций в зависимости от режима работы контроллера и периферийных устройств, подключенных к этим линиям.

1.2.3. Таймеры/счётчики.

В микроконтроллере ATmega128 есть 4 таймера/счётчика, и обозначаются они как T/C0, T/C1, T/C2, T/C3. Кроме того, в составе микроконтроллера имеется еще и, так называемый, сторожевой таймер, позволяющий выйти из «зависшей» программы.

Восьми разрядные таймеры/счётчики T/C0 и T/C2 идентичны, равно как идентичны и 16-ти разрядные T/C1 и T/C2. Функционально таймер/счетчик представляет собой:

1. счетный регистр, который инкрементируется (или декрементируется), в зависимости от режима работы таймера в каждом такте счётчика;

2. несколько вспомогательных регистров, необходимых для задания режимов работы счётчика (например, регистр блока сравнения или блока захвата);
3. линии входа и выхода;
4. механизм генерации прерываний на события, связанные с изменением состояния счётного регистра.

В таймерах/счетчиках T/C0 и T/C2 счётные регистры имеют по 8 разрядов и соответственно максимальное разрешение этих счётчиков – 256 тактов. Счётные регистры T/C1 и T/C3 16-ти разрядные и максимальный временной интервал, который они могут измерить непосредственно - 65536 тактов (таймера). Для получения тактового сигнала таймеров/счётчиков используются предделители, которые делят системный тактовый сигнал на программно-задаваемый коэффициент. В ATmega128 имеется 2 предделителя: один для T/C0, другой используется совместно таймерами T/C1,T/C2,T/C3. В качестве тактового сигнала таймеров может быть использован либо непосредственно системный тактовый сигнал микропроцессора clc_0, либо внешний сигнал с частотой не более, чем $clc_0/2$.

В ATmega128 предусмотрены несколько режимов работы таймеров/счётчиков и связанных с ними прерываний:

1. Normal (обычный)
2. CTC (сброс при совпадении)
3. Fast PWM (быстродействующий ШИМ)
4. Phase Correct PWM (ШИМ с точной фазой).

В режиме Normal по каждому импульсу тактового сигнала осуществляется инкрементирование счётчика (счетного регистра). При переходе через максимально возможное значение ($FF- для T/C0 и T/C2, $FFFF- для T/C1 и T/C2) возникает переполнение и счёт продолжается с $0000. В том же такте сигнала, в котором обнуляется счётный регистр, генерируется запрос на прерывание по переполнению. В этом режиме также возможна генерация прерываний по совпадению, то есть, когда значение счётного регистра

совпадает со значением, находящимся в регистре сравнения. В случае 16-разрядных таймеров существует несколько прерываний по совпадению, так как в них реализованы несколько блоков сравнения (точнее 3). Блоки сравнения таймеров могут также использоваться для изменения состояний линий выходов соответствующих таймеров/счётчиков.

В режиме CTC, также как и в предыдущем случае, инкремент счётного регистра осуществляется по каждому импульсу тактового сигнала, однако максимальное значение счётного регистра и, следовательно, разрешающая способность счётчика определяется регистром сравнения (в случае 16-ти разрядных таймеров регистром сравнения первого (A) блока). После достижения значения, записанного в регистре сравнения, счёт продолжается со значения $0000. В этом же такте сигнала, происходит генерация запросов на прерывания по переполнению и по совпадению соответствующего таймера/счётчика. Одновременно с генерацией прерываний может изменяться состояние выхода соответствующего таймера.

Режимы Fast PWM и Phase Correct PWM предназначены для генерации сигналов с широтно-импульсной модуляцией.

Доступ к таймерам/счётчикам и управление связанными с ними прерываниями и предделителями осуществляется с помощью соответствующих РВВ.

1.2.4. Универсальный синхронный/асинхронный приёмопередатчик

Микроконтроллер ATmega128 имеет два модуля последовательной связи, называемые универсальными синхронно/асинхронными приёмопередатчиками (USART). Оба модуля USART обеспечивают полнодуплексный обмен по последовательному каналу, при этом скорость передачи данных может варьироваться в довольно широких пределах. Длина посылки может составлять от 5 до 9 разрядов с контролем четности, или без него. Кроме того, модули USART могут обнаруживать различные внештатные ситуации, например, переполнение буфера приёмника, ошибка

кадрирования, неверный старт бит. Для уменьшения вероятности сбоев в модулях реализована функция фильтрации помех.

Для взаимодействия с программой в модулях предусмотрены 3 прерывания, запрос на генерацию которых генерируется при наступлении событий: передача завершена, регистр данных передатчика пуст, приём завершён.

Выводы микроконтроллера, используемые модулями USART, являются линиями портов ввода/вывода общего назначения, функционирование которых переопределяется при включении приёмника или передатчика.

Каждый модуль USART состоит из трёх основных частей: блока тактирования, блока передатчика и блока приёмника.

Блок тактирования включает в себя схему синхронизации, которая используется при работе в синхронном режиме, а также контроллер скорости передачи.

Блок передатчика включает одноуровневый буфер, сдвиговый регистр, схему формирования бита чётности и схему управления.

Блок приёмника, в свою очередь, включает схему восстановления тактового сигнала, схему восстановления данных, схему контроля чётности, двухуровневый FIFO-буфер, сдвиговый регистр, а также схему управления.

Доступ и управление модулями USART, как обычно, осуществляется с помощью соответствующих РВВ.

1.2.5. Система команд

Система команд МК AVR семейства Mega весьма развита и насчитывает в различных моделях от 130 до 135 разнообразных инструкций. И из-за этого, несмотря на то, что МК AVR имеют RISC архитектуру, их система команд по количеству реализуемых инструкций сопоставима с CISC архитектурой. При этом практически каждая команда, за исключением команд с абсолютной адресацией занимает одну ячейку памяти программ. Ниже приводятся обозначения и таблица команд для МК AVR, а описание

команды, ее формат, количество тактов и примеры доступны по гиперссылке на сайте http://www.gaw.ru/

Перед тем, как приступить к рассмотрению системы команд вспомним некоторые основные архитектурные особенности микроконтроллера. Микроконтроллер имеет своем составе 32 регистра. Первая их половина (**R0-R15**) не может быть использована в операциях с непосредственным операндом. Во второй половине есть специфические регистровые пары, которые могут использоваться в операциях пересылки данных между регистрами и памятью и некоторых других действий (**X**,**Y** и **Z**). Заметим к тому же, что "возможности" этих регистровых пар различны!

Кроме регистров, микроконтроллер имеет память данных (ОЗУ), обращение к которой производится при помощи регистровых пар (индексная адресация) или указанием 16-ти разрядного адреса (абсолютная адресация). Микроконтроллер может только прочесть память данных в регистр или записать туда из регистра, никакие арифметические или логические операции с памятью данных невозможны.

Для работы с периферией предназначены регистры ввода-вывода (**I/O**). Можно прочитать данные из **I/O** в регистр общего назначения и записать из регистра общего назначения в **I/O**. Кроме этого, у части регистров ввода-вывода, а точнее - у тех, чей адрес не превышает 0x1F, возможна установка отдельных бит в состояние 0 или 1.

1.2.5.1. Принятые обозначения

Регистр статуса (SREG)

SREG:	Регистр статуса
C:	Флаг переноса
Z:	Флаг нулевого значения
N:	Флаг отрицательного значения
V:	Флаг-указатель переполнения дополнения до двух
S:	NEV, Для проверок со знаком
H:	Флаг полупереноса
T:	Флаг пересылки, используемый командами BLD и BST
I:	Флаг разрешения/запрещения глобального прерывания

Регистры и операнды	
Rd:	Регистр назначения (и источник) в регистровом файле
Rr:	Регистр источник в регистровом файле
R:	Результат выполнения команды
K:	Литерал или байт данных (8 бит)
k:	Данные адреса константы для счетчика программ
b:	Бит в регистровом файле или I/O регистр (3 бита)
s:	Бит в регистре статуса (3 бита)
X, Y, Z:	Регистр косвенной адресации (X=R27:R26, Y=R29:R28, Z=R31:R30)
P:	Адрес I/O порта
q:	Смещение при прямой адресации (6 бит)
I/O регистры	
RAMPX, RAMPY, RAMPZ:	Регистры связанные с X, Y и Z регистрами, обеспечивающие косвенную адресацию всей области СОЗУ микроконтроллера с объемом СОЗУ более 64 Кбайт
Стек:	
STACK:	Стек для адреса возврата и опущенных в стек регистров
SP:	Указатель стека
Флаги:	
Ы	Флаг, на который воздействует команда
0:	Очищенный командой Флаг
1:	Установленный командой флаг
-:	Флаг, на который не воздействует команда

1.2.5.2. Команды

Команды детализируются по гиперссылке.

Обозначение	Функция
ADC	Сложить с переносом
ADD	Сложить без переноса
ADIW	Сложить непосредственное значение со словом
AND	Выполнить логическое AND
ANDI	Выполнить логическое AND с непосредственным значением
ASR	Арифметически сдвинуть вправо
BCLR	Очистить флаг
BLD	Загрузить T флаг в бит регистра
BRBC	Перейти если бит в регистре статуса очищен
BRBS	Перейти если бит в регистре статуса установлен
BRCC	Перейти если флаг переноса очищен
BRCS	Перейти если флаг переноса установлен
BREQ	Перейти если равно

BRGE	Перейти если больше или равно (с учетом знака)
BRHC	Перейти если флаг полупереноса очищен
BRHS	Перейти если флаг полупереноса установлен
BRID	Перейти если глобальное прерывание запрещено
BRIE	Перейти если глобальное прерывание разрешено
BRLO	Перейти если меньше (без знака)
BRLT	Перейти если меньше чем (со знаком)
BRMI	Перейти если минус
BRNE	Перейти если не равно
BRPL	Перейти если плюс
BRSH	Перейти если равно или больше (без знака)
BRTC	Перейти если флаг T очищен
BRTS	Перейти если флаг T установлен
BRVC	Перейти если переполнение очищено
BRVS	Перейти если переполнение установлено
BSET	Установить флаг
BST	Переписать бит из регистра во флаг T
CALL	Выполнить длинный вызов подпрограммы
CBI	- Очистить бит в регистре I/O
CBR	Очистить биты в регистре
CLC	Очистить флаг переноса
CLH	Очистить флаг полупереноса
CLI	Очистить флаг глобального прерывания
CLN	Очистить флаг отрицательного значения
CLR	Очистить регистр
CLS	Очистить флаг знака
CLT	Очистить флаг T
CLV	Очистить флаг переполнения
CLZ	Очистить флаг нулевого значения
COM	Выполнить дополнение до единицы
CP	Сравнить
CPC	Сравнить с учетом переноса
CPI	Сравнить с константой
CPSE	Сравнить и пропустить если равно
DEC	Декрементировать
EOR	Выполнить исключающее OR
ICALL	Вызвать подпрограмму косвенно
IJMP	Перейти косвенно
IN	Загрузить данные из порта I/O в регистр

INC	Инкрементировать
FMUL	Дробное незнаковое умножение
FMULS	Дробное умножение со знаком
FMULSU	Дробное умножение знакового с незнаковым
JMP	Перейти
LD Rd,X	Загрузить косвенно
LD Rd,X+	Загрузить косвенно инкрементировав впоследствии
LD Rd,-X	Загрузить косвенно декрементировав предварительно
LDI	Загрузить непосредственное значение
LDS	Загрузить непосредственно из СОЗУ
LPM	Загрузить байт памяти программ
LSL	Логически сдвинуть влево
LSR	Логически сдвинуть вправо
MOV	Копировать регистр
MUL	Перемножить
NEG	Выполнить дополнение до двух
NOP	Выполнить холостую команду
OR	Выполнить логическое OR
ORI	Выполнить логическое OR с непосредственным значением
OUT	Записать данные из регистра в порт I/O
POP	Загрузить регистр из стека
PUSH	Поместить регистр в стек
RCALL	Вызвать подпрограмму относительно
RET	Вернуться из подпрограммы
RETI	Вернуться из прерывания
RJMP	Перейти относительно
ROL	Сдвинуть влево через перенос
ROR	Сдвинуть вправо через перенос
SBC	Вычесть с переносом
SBCI	Вычесть непосредственное значение с переносом
SBI	Установить бит в регистр I/O
SBIC	Пропустить если бит в регистре I/O очищен
SBIS	Пропустить если бит в регистре I/O установлен
SBIW	Вычесть непосредственное значение из слова
SBR	Установить биты в регистре
SBRC	Пропустить если бит в регистре очищен
SBRS	Пропустить если бит в регистре установлен
SEC	Установить флаг переноса
SEH	Установить флаг полупереноса

SEI	Установить флаг глобального прерывания
SEN	Установить флаг отрицательного значения
SER	Установить все биты регистра
SES	Установить флаг знака
SET	Установить флаг T
SEV	Установить флаг переполнения
SEZ	Установить флаг нулевого значения
SLEEP	Установить режим SLEEP
ST X,Rr	Записать косвенно
ST Y,Rr	Записать косвенно из регистра в СОЗУ с использованием индекса Y
ST Z,Rr	Записать косвенно из регистра в СОЗУ с использованием индекса Z
STS	Загрузить непосредственно в СОЗУ
SUB	Вычесть без переноса
SUBI	Вычесть непосредственное значение
SWAP	Поменять нибблы местами
TST	Проверить на ноль или минус
WDR	Сбросить сторожевой таймер

Следует обратить внимание на следующие тонкости. Многие команды могут обращаться только к 16-старшим РОН и не имеют доступа к 16-младшим. Значения смещений и констант могут быть ограничены и оказаться не в том диапазоне, который вы ожидали. Но прежде чем приступить к изучению команд, целесообразно рассмотреть различные способы адресации данных.

Основным способом доступа к данным является прямое обращение к РОН – это прямая (регистровая адресация). На рис.1.2.3 операнд команды содержится в регистре Rd, а КОП обозначает часть слова команды, соответствующую **К**оду **Оп**ерации. Обычно в формате команды отводится пять бит, которые позволяют адресоваться к любому регистру.

Команды, оперирующие с двумя регистрами, действуют, в основном, аналогичным образом. В этих командах регистр-приемник Rd (destination) указывается перед регистром-источником Rr (resource), то есть является первым параметром (см. рис.1.2.4). Таким образом, команда:

ADD R0, R1; реально выполняется так: R0 = R0 + R1.

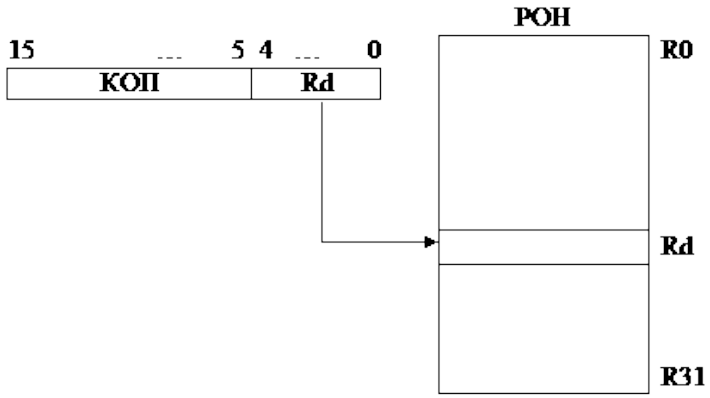


Рис. 1.2.3. Прямая адресация одного регистра.

Результат сохраняется в регистре Rd.

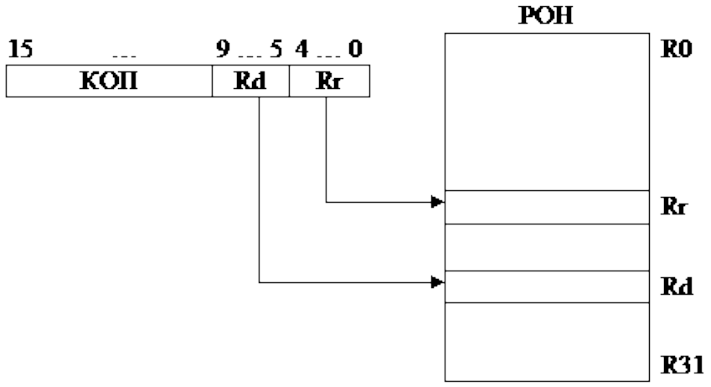


Рис. 1.2.4. Прямая адресация к двум РОН.

1.2.5.3. Прямая адресация к регистрам ввода/вывода

На рис. 1.2.5 адрес операнда содержится в 6 битах слова команды (ячейка Р).

Rd – определяет адрес регистра источника или регистра приемника.

Например, этот тип адресации могут использовать команды IN или OUT:

IN R0, SREG ; сохранить регистр состояния в регистре R0

*OUT PORTB, R1 ;*записать данные из регистра R1 в PORTB

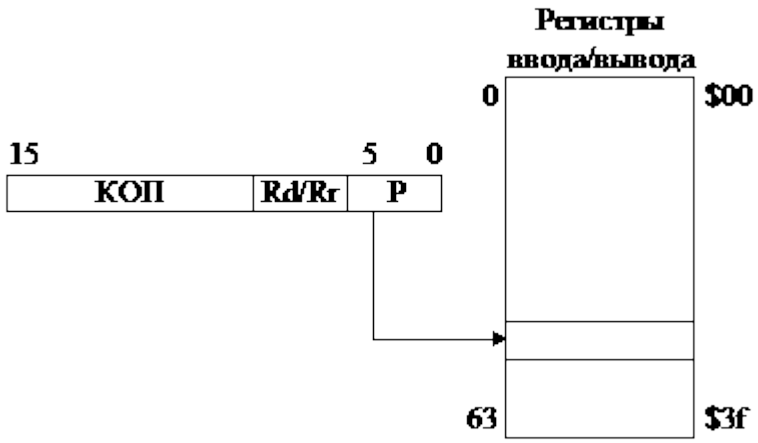


Рис. 1.2.5. Прямая адресация регистров ввода/вывода.

Прямая (абсолютная) адресация данных.

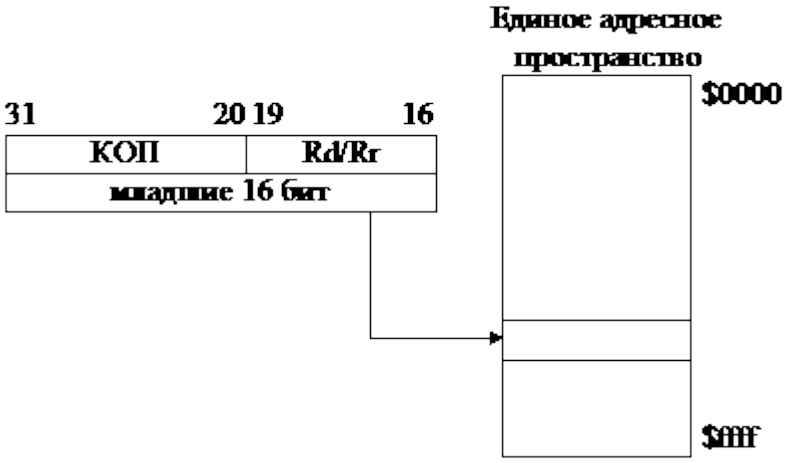


Рис. 1.2.6. Прямая (абсолютная) адресация данных.

Единое адресное пространство является пространством данных, включая РОН, регистры ввода/вывода, внутренняя память и внешняя память (если есть). 16-разрядный адрес данных содержится в 16 младших разрядах 32-х разрядной команды. Rd/Rr определяет адрес регистра источника или регистра приемника. Такой тип адресации, к примеру, могут использовать команды LDS и STS:

LDS PORTB, R1 ;записать данные из регистра R1 в PORTB

STS PORTB, R1 ;записать данные из регистра R1 в PORTB

1.2.5.4. Косвенная адресация данных.

Существует четыре типа косвенной адресации данных: простая, с пост инкрементом, с пред декрементом, со смещением. Для первых трех типов косвенной адресации данных адрес операнда содержится в регистре X, Y или Z. Для последнего типа (со смещением) адрес операнда вычисляется сложением содержимого регистров Y или Z с шестью битами адреса, содержащимися в слове команды. Сами же регистры остаются неизмененными. Смысл косвенной адресации с пост инкрементом (с пред декрементом) заключается в следующем. После (до) выполнения операции регистр X, Y и Z инкрементируется (декрементируется). Рассмотрим некоторые примеры:

1) *ST X, Rn ;* поместить данные из Rn в по адресу указанному в регистре X;
2) 2) *ST X+, Rn ;*тоже что и в 1), но впоследствии X инкрементируется;
3) 3) *ST -X, Rn* ;предварительно X декрементируется. В предыдущих трех примерах вместо X могут быть Y или Z.
4) 4) *STD Y+c, Rn* ; к значению Y прибавляется константа смещения ’c’ и по этому адресу записываются данные из Rn. Y при это остается неизменен. Вместо Y может быть только Z.

1.2.5.5. Команды пересылки данных

Пересылка данных из одного места в другое является для AVR очень простой операцией, так как имеется очень большое количество команд, предназначенных для выполнения этой задачи. Ни одна команда пересылки данных не оказывает влияния на биты регистра состояния. Команда LPM – загрузка данных из таблиц, хранящихся в памяти программ. В этой команде младший бит индексного регистра Z используется для указания байта, который будет читаться, если 0 – читается младший байт, 1 – старший. Оставшиеся 14 байт используются для указания адреса слова. Команда MOV – копирует содержимое одного РОН в другой. LDI – загружает в один из 16 старших РОН байт, содержащийся в команде. IN и

OUT – доступ к регистрам ввода/вывода, начиная с нулевого адреса.
1.2.5.6. Команды арифметических и логических операций

Основными арифметическими операциями являются сложение и вычитание двух чисел. Эти команды по большей части очевидны. Сложение и вычитание содержимого двух регистров производиться при помощи команд ADD и SUB. Модификации этих команд, которые учитывают значение флага переноса, позволяют выполнить операции над 8-, 16-, 24- и даже 32-разрядными числами со знаком, хранящимися в регистрах.

Поясним функции флагов отрицательного результата N (negative), переполнения V (overflow) и знака S (sign), так как они имеют некоторые особенности. Флаг отрицательного результата N просто копирует значение бита 7 результата, который показывает, является результат положительным или отрицательным числом.

Флаг переполнения V в регистре SREG указывает на переполнение во время сложения или вычитания чисел со знаком. Рассмотрим пример:

ADD R1, R2;

Флаг V будет установлен в 1, если в регистрах R1 и R2 содержаться положительные числа, а результат их сложения окажется больше 127, или оба числа отрицательны, а результат будет меньше –128. Рассмотрим пример с конкретными значениями:

```
LDI   R1, 100        ;100 = 0b01100100
LDI   R2, 100        ;Занести 0b01100100 R1 и R2
ADD   R1, R2         ;R1 = R1 + R2 = 200 = 0b11001000
```

Десятичное число 200 в двоичной записи имеет значение бита 7 равное 1, что указывает на получение отрицательного результата. Следовательно, после выполнения операции сложения флаг N будет установлен в 1. Но в данном случае вместе с флагом N будет так же установлен в 1 флаг V, указывая, что произошло переполнение при обработке чисел со знаком.

Если содержимое R1 = R2 = - 100, то результатом сложения этих чисел будет 0b00111000 в двоичной системе счисления, что является

положительным числом. При этом флаг N будет сброшен в 0,показывая, что результата положителен, однако будет установлен флаг V, означающий, что на самом деле это не так.

Использование флага S = N ^ V позволяет рассматривать результат как 9-разрядное число со знаком, где старшим (знаковым) разрядом как раз и является флаг S. Как было отмечено при описании флага V, он устанавливается в 1, когда бит 7 результата имеет неправильное значение, то есть результат не представлен правильным числом со знаком в дополнительном коде. Выполнив операцию «ИСКЛЮЧАЮЩЕЕ ИЛИ» над значением флага V и бита 7 результата, который храниться в бите N, вы получите реальный знак результата. В первом примере (100 + 100) происходит установка в 1 флагов V и N, в результате флаг S будет равняться нулю (1 ^ 1 = 0). Во втором примере (- 100 - 100) флаг N сбрасывается в 0, а флаг V устанавливается в 1, поэтому флаг S будет равняться единице, указывая на то, что результат отрицательный.

Флаг S должен использоваться со старшим байтом числа. При операциях с 16-, 24- и 32-разрядными числами значение флага S надо проверять только после завершения последней операции со старшим байтом числа. При операциях с младшими байтами используется флаг переноса C, как обычно при выполнении сложения и вычитания.

1.2.5.7. Команды ветвления

Команды относительного перехода RJMP и вызова подпрограммы RCALL являются основными для изменения выполнения последовательности команд в МК. При этом содержимое программного счетчика изменяется на величину смещения, которое задается в 12 младших битах кода команды.

МК может выполнять команды ветвления по значению определенных битов в регистре состояния SREG. Поскольку номер бита и его значение должны быть указаны в коде команды, то диапазон возможных адресов перехода составляет +/- 63 относительно текущего адреса. Это означает, что применение команд условных ветвлений весьма ограничено, но эту проблему

можно решить при помощи ветвления к команде, которая затем выполнит необходимый безусловный переход.

Еще один класс команд ветвления – это команды пропуска. После проверки указанного условия, данные команды либо выполняют следующую команду, либо пропускают ее.

1.2.5.8. Битовые команды и команды тестирования битов

Команды сброса (очистки) и установки битов предназначены для модификации регистров ввода/вывода. Но некоторые из них могут работать только с частью регистров ввода/вывода. Это значит, что для некоторой части регистров ввода/вывода вы должны сначала переписать их содержимое в РОН, модифицировать, а затем снова сохранить в регистре ввода/вывода. Для выполнения этой процедуры можно написать специальную макрокоманду (макрос).

Часто необходимо переслать бит из одного регистра или переменной в другой. Это можно сделать следующим образом:

BST r1,5 ;поместить бит 5 регистра r1в T-бит регистра SREG

BLD r1,2 ;сохранить T-бит регистра SREG в разряде 2 регистра r1

Команда SWAP меняет местами старший и младший полубайт регистра. Это полезно когда вы храните в регистре две цифры, а не одно восьмибитовое число. Команды сдвигов и циклических сдвигов LSL, LSR, ROL, ROR и ASR полезны как для выполнения сдвигов данных в процессе их ввода вывода, так и для проверки значения определенного бита в РОН без необходимости выполнения 8 отдельных операций тестирования битов. С помощью циклического сдвига можно произвести индивидуальную проверку любого бита в заданном месте байта.

1.2.6. Программирование AVR

Микроконтроллер ATmega128 поддерживает 3 режима программирования:

1. последовательное программирование при низком напряжении (по интерфейсу SPI);

2. параллельное программирование при высоком напряжении;
3. программирование по интерфейсу JTAG.

Кроме того, имеется возможность самопрограммирования. Под этим термином понимается изменение содержимого памяти программ, управляемое самим микроконтроллером.

В процессе программирования могут выполняться следующие операции:

- стирание кристалла;
- чтение/запись памяти программ;
- чтение/запись памяти данных;
- чтение/запись ячеек защиты;
- чтение/запись конфигурационных ячеек;
- чтение ячеек идентификатора;
- чтение калибровочного байта.

Содержимое памяти программ и данных может быть защищено от чтения и/или записи посредством программирования соответствующих ячеек защиты. После программирования этих ячеек память контроллера блокируется для соответствующих операций. Ячейки защиты могут быть перезаписаны только после выполнения команды стирания кристалла, что также уничтожит содержимое памяти программ и данных микроконтроллера.

Конфигурационные ячейки определяют некоторые параметры конфигурации микроконтроллера. Эти ячейки расположены в отдельном адресном пространстве, доступном только при программировании. Команда стирание кристалла на состояние этих ячеек не влияет.

Все микроконтроллеры AVR имеют три 8-ми разрядные ячейки, позволяющие идентифицировать устройство (ячейки идентификатора). Они доступны только в режиме программирования и только для чтения.

В калибровочную ячейку при изготовлении микроконтроллера заносится калибровочная константа RC-генератора. При работе от внутреннего RC-генератора необходимо воспользоваться содержимым этой ячейки для подстройки внутреннего генератора на номинальную частоту.

В лабораторном практикуме для программирования микроконтроллеров ATmega используется программатор AS-2, работающий по интерфейсу SPI, а для написания и отладки программ на ассемблере - интегрированная среда разработки «AVR-Studio», разработанная компанией ATMEL для поддержки выпускаемых устройств.

Программа «AVR-Studio» (рис.1.2.7), предназначенная для операционных систем Windows, позволяет создавать, редактировать и отлаживать программы для микроконтроллеров AVR на языках ассемблера и Си. Она свободно доступна на сайте компании ATMEL. Благодаря наличию для каждой модели микроконтроллеров своих заголовочных файлов, где определены характерные константы, такие как максимальный адрес памяти программ и данных, адреса регистров ввода/вывода и т.д., процесс программирования существенно упрощается.

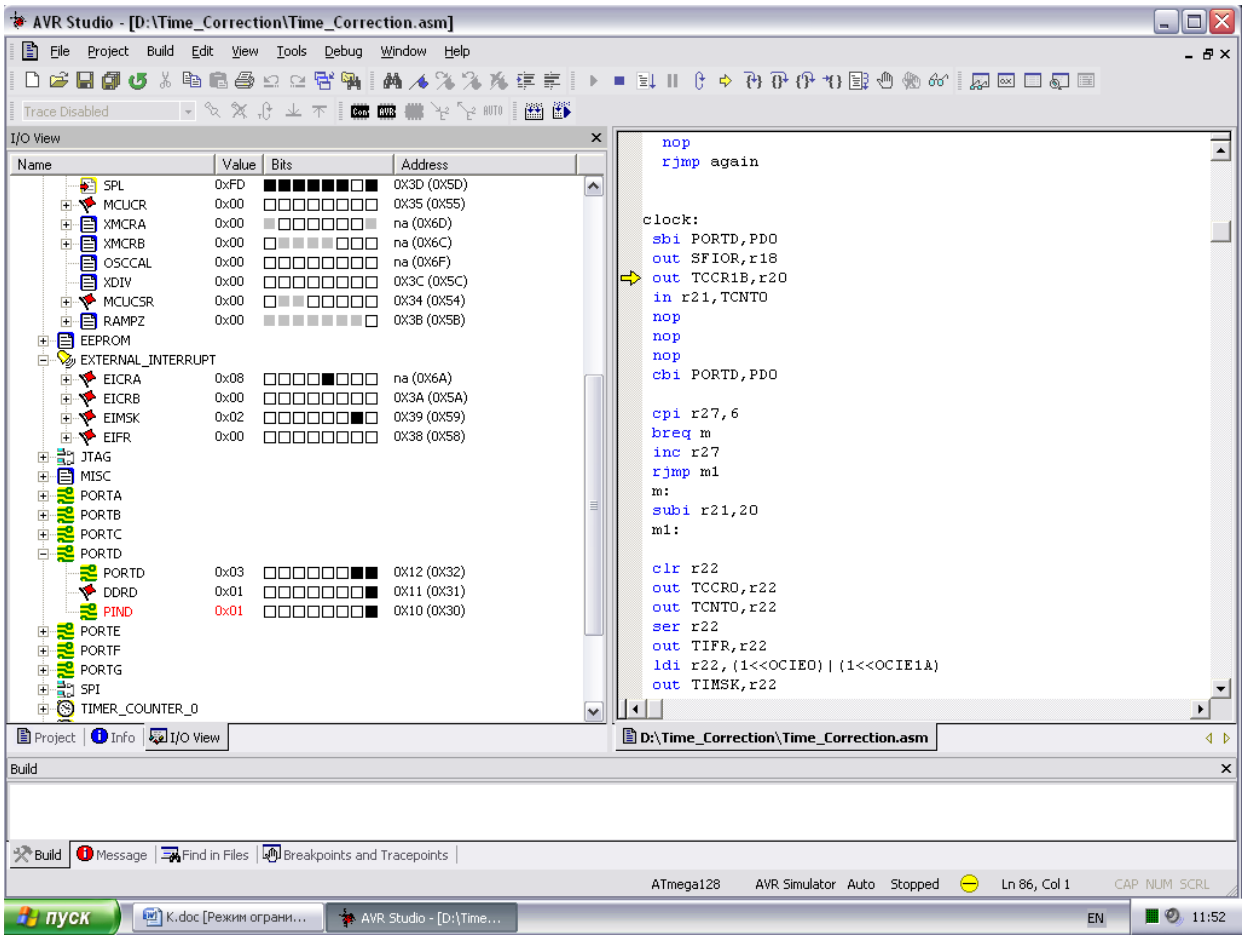


Рис. 1.2.7. Консоль программы «AVR-Studio».

Программный симулятор работает с большинством моделей МК AVR и позволяет моделировать поведение микроконтроллера при выполнении отлаживаемого приложения, что делает возможным поиск ошибок в нём ещё до записи непосредственно в микроконтроллер. Результатом работы «AVR-Studio» является HEX файл с кодами команд для микроконтроллера. Для записи этого файла в память микроконтроллера можно использовать программатор AS-2 от фирмы Аргуссофт.

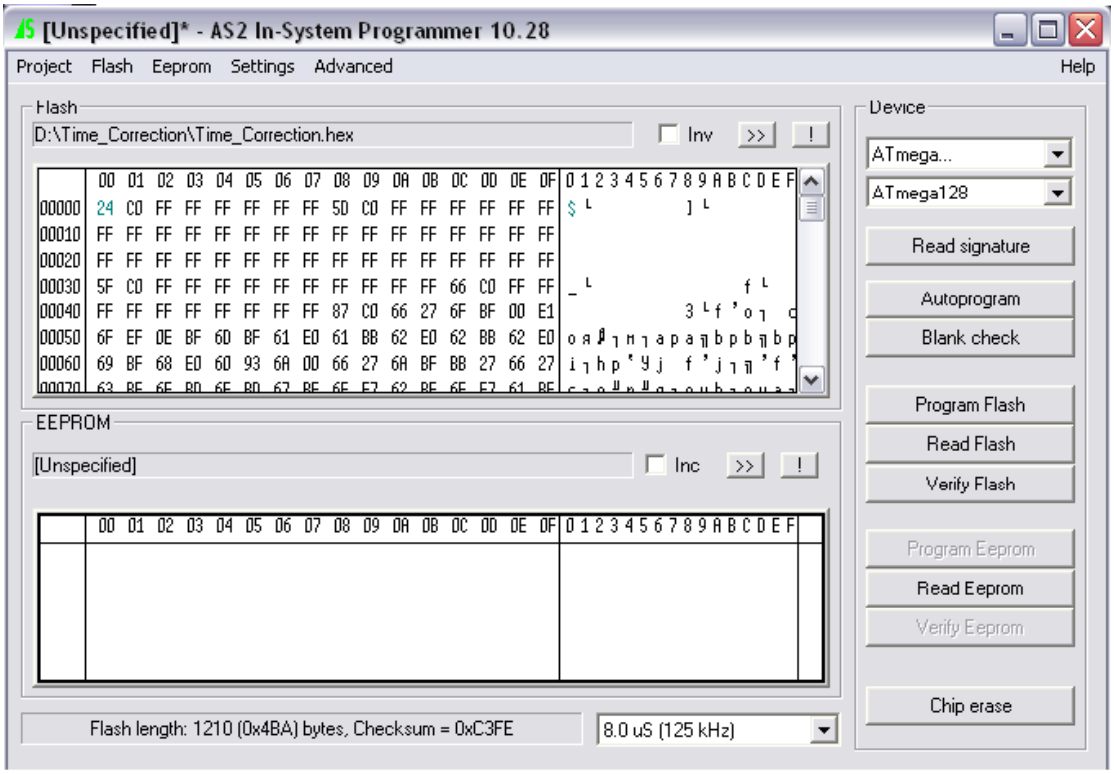


Рис. 1.2.8. Рабочее окно программатора AS-2.

1.2.7. Пример построения приложения

Работу пакета «AVR-Studio» удобно рассматривать на примере решения какой-либо конкретной задачи. В качестве иллюстрации рассмотрим создание проекта простой программы, реализующей поочередное включение/выключение двух светодиодов. Для отладочной платы AS-megaM подключим два светодиода к разрядам 6 и 7 порта D, то есть к выводам 31 и 32 микросхемы ATmega128. AVR-контроллеры имеют мощные выходные каскады, типовое значение тока каждого вывода составляет 20 мА, максимальный ток вывода – 40 мА, причем это относится

как к втекающему, так и к вытекающему току. В нашем примере светодиоды подключены анодами к выводам контроллера, а катоды через гасящие резисторы соединены с землей. Это означает, что светодиод зажигается подачей логической «1» на соответствующий вывод порта. Принципиальная схема используемых в данном примере элементов приведена на рисунке1.2.9. На схеме также показаны две кнопки, которые будут использованы в программе.

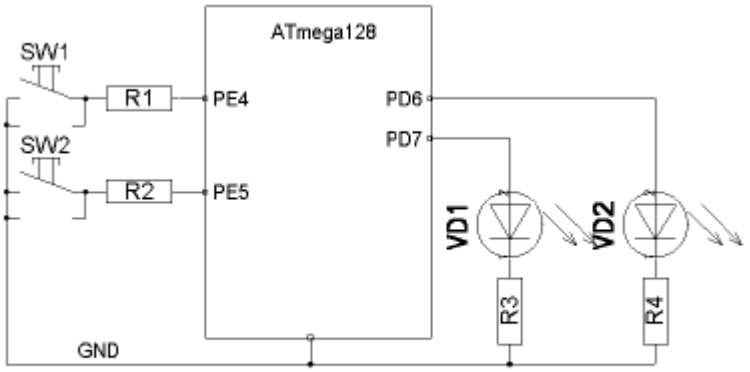


Рис. 1.2.9. Схема подключения светодиодов

Здесь уместно сделать небольшое отступление о выборе типа микросхемы для простейшего примера. Действительно, с первого взгляда может показаться странным, зачем нужен такой мощный кристалл в 64-выводном корпусе там, где хватит и 8-ми выводной микросхемы ATtiny12? Однако в таком подходе есть логика. Известно, что в основе практически любого AVR-контроллера находится однотипное ядро. В основном контроллеры различаются объемом памяти, количеством портов ввода/вывода и набором периферийных модулей. Особенности каждого конкретного контроллера – привязка логических имен регистров ввода/вывода к физическим адресам, адреса векторов прерываний, определения битов портов и т.д. описаны в “include” файлах (расширением .inc), которые входят в состав пакета «AVR-Studio». Следовательно, используя конкретный тип кристалла, можно отлаживать программу как собственно для него, так и для любого младшего кристалла. И если использовать в качестве отладочного старший кристалл,

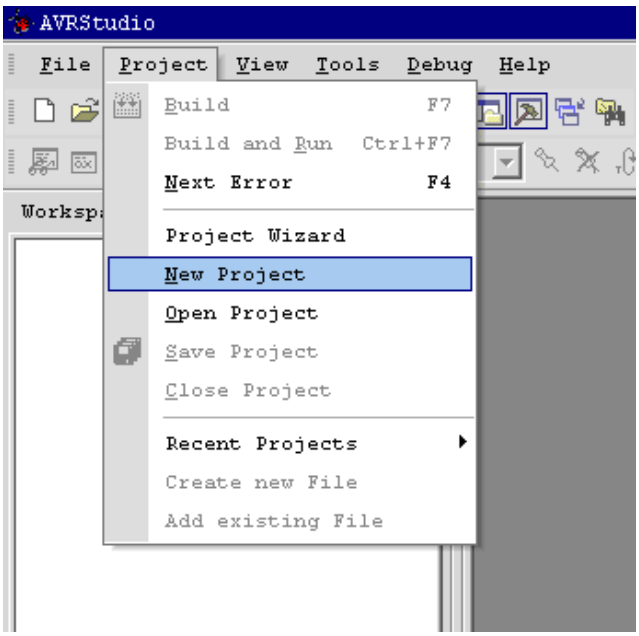
например, ATmega128, то можно отлаживать программу практически для любого AVR-контроллера. Просто не надо использовать аппаратные ресурсы, которые отсутствуют у целевого микроконтроллера. Таким образом, можно отлаживать на ATmega128 программу, которая будет выполняться, например, на ATtiny13. При этом исходный код останется практически тем же, изменится лишь имя подключаемого файла с 128def.inc на tn13def.inc. У такого подхода также есть свои преимущества. Например, «лишние» порты ввода/вывода можно использовать для подключения ЖК-индикатора, на который можно выводить отладочную информацию. Или, воспользоваться внутрисхемным эмулятором, который подключается к JTAG-порту микросхемы ATmega128 (контроллер ATtiny13 такой порт не имеет). Таким образом, можно использовать отладочную плату AS-megaM для отладки любых вновь разрабатываемых систем, естественно, базирующихся также на AVR-микроконтроллерах. На плате помимо контроллера ATmega128 имеется внешнее ОЗУ, два порта RS-232, порт для подключения ЖК-индикатора, внутрисхемного программатора и эмулятора AT JTAGICE. Кроме того, есть место для распайки микросхемы FLASH-ПЗУ серии AT45 в корпусах TSOP32/40/48 и двухканального ЦАП серии AD5302/12/22.

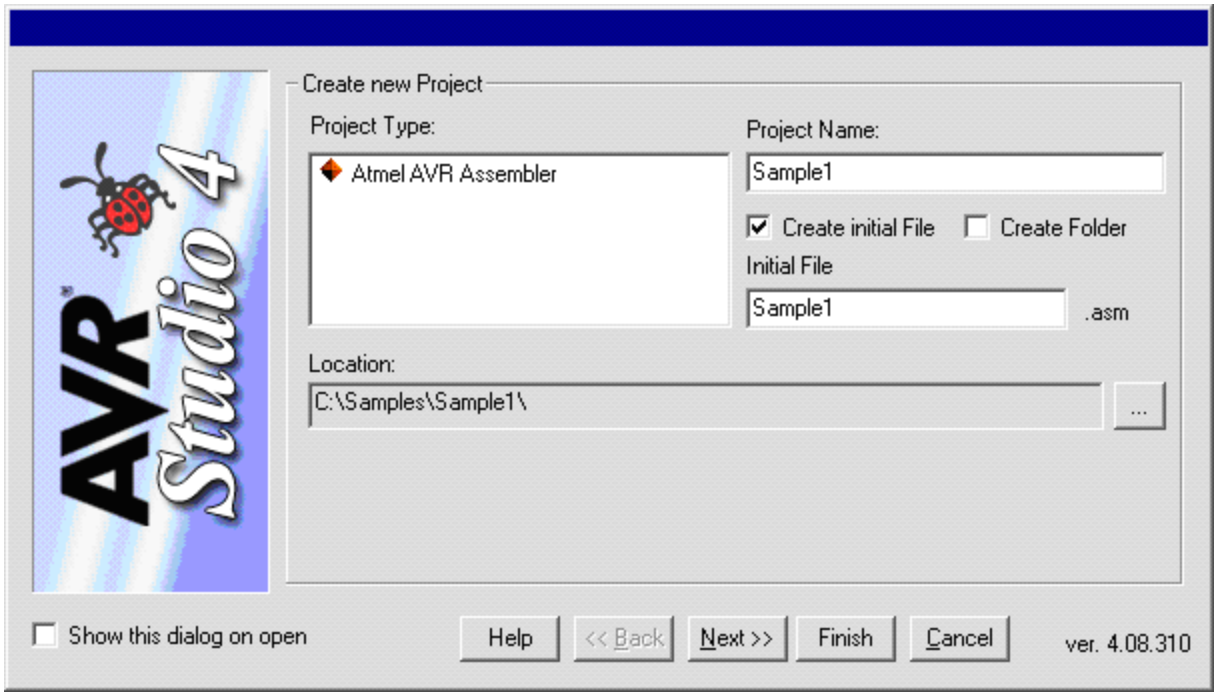
При программировании в среде «AVR-Studio» надо выполнить стандартную последовательность действий:

- *создание проекта*
- *загрузка файла*
- *компиляция*
- *симуляция*
- *загрузка hex-кода в микроконтроллер*

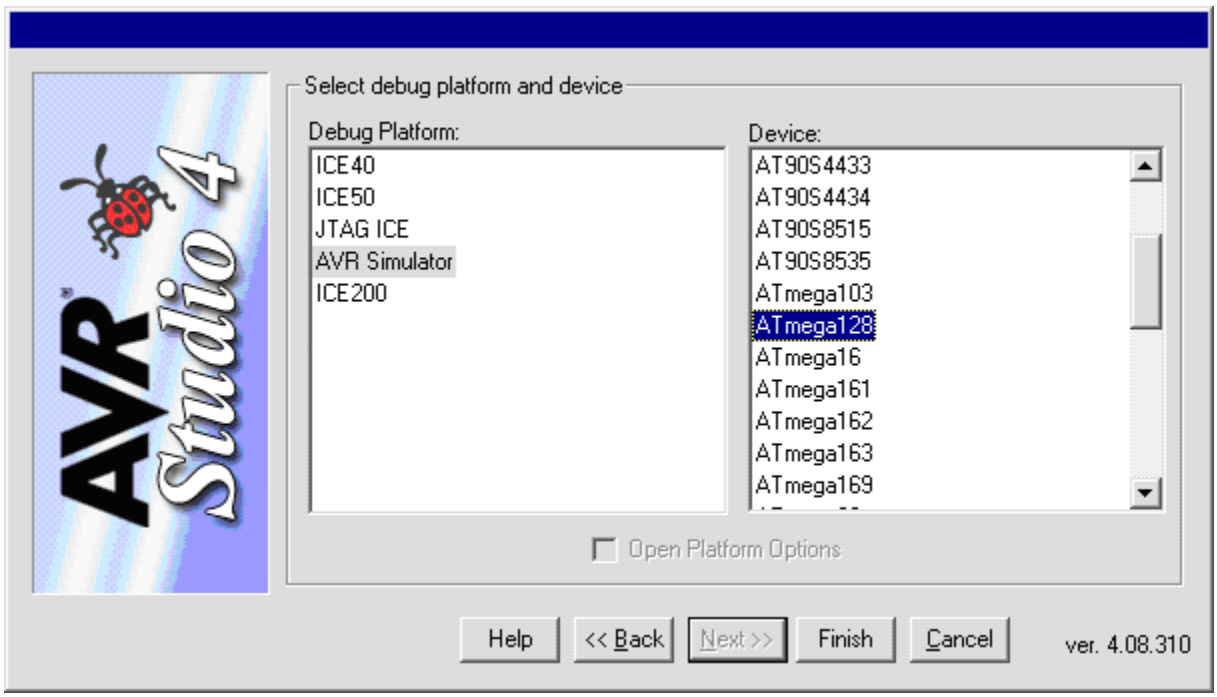
Эти действия выполняются в представленных ниже окнах и в особых пояснениях не нуждаются.

Открываем новый проект





Выбираем требуемый тип микроконтроллера



Создание проекта начинается с выбора строки меню Project\New Project. В открывшемся окне “Create new Project” надо указать имя проекта, (в нашем случае – sample1) и имя файла инициализации. После нажатия кнопки «Next» открывается окно «Select debug platform and device», где выбирается отладочная платформа (симулятор или эмулятор) и тип микроконтроллера.

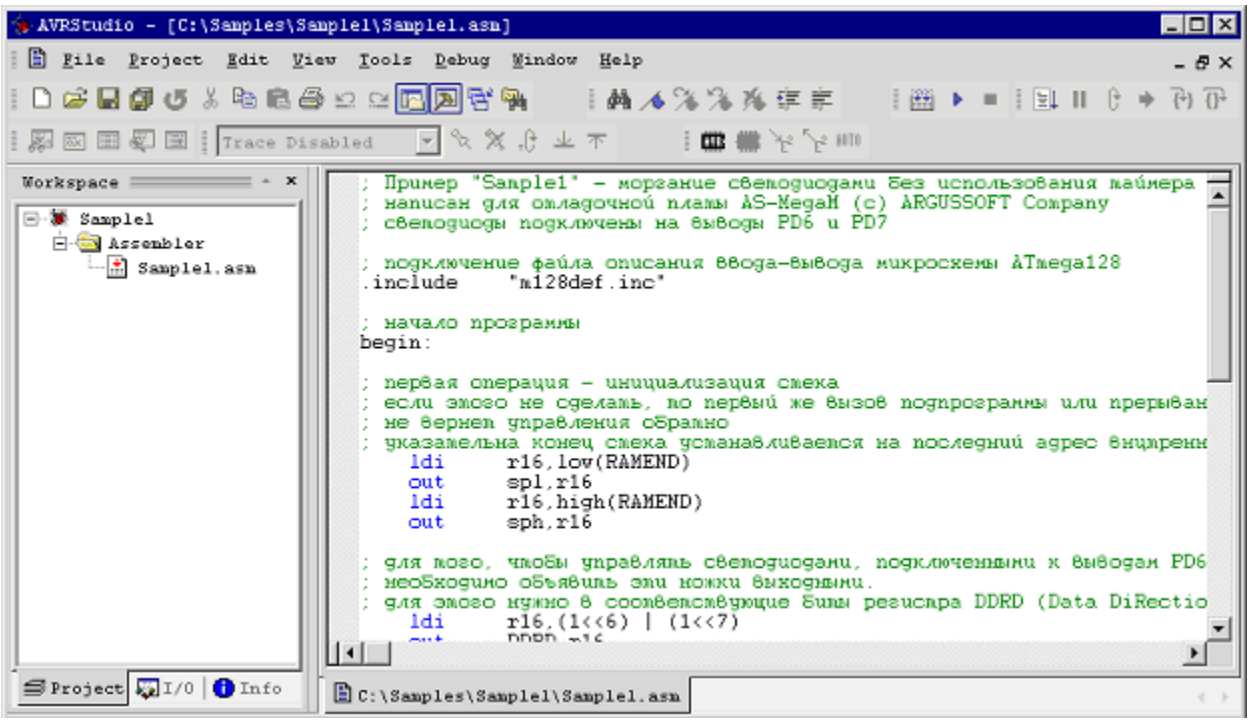
Можно выбрать один из предлагаемых внутрисхемных эмуляторов. Отметим, что у каждого эмулятора свой список поддерживаемых микросхем. Для рассматриваемого примера мы выбираем в качестве отладочной платформы AVR Simulator и микросхему ATmega128. После нажатия кнопки “Finish” мы увидим рабочие окна пакета «AVR-Studio», пока пустые. В правое окно следует поместить исходный текст программы. Это можно сделать двумя способами: либо набрать весь текст непосредственно в окне редактора, либо загрузить уже существующий файл. Ниже приведен полный текст простой программы с комментариями.

```
; Пример 1: «Управление светодиодами»
; написан для отладочной платы AS-MegaM
; Частота задающего генератора 7,37 МГц
; светодиоды подключены к выводам PD6 и PD7 и через резисторы на «землю».
```

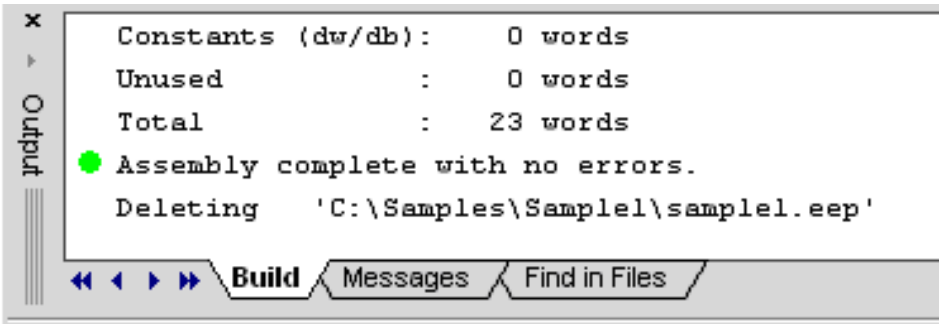
```
        ; подключение файла описания ввода-вывода микросхемы ATmega128

        .include "m128def.inc"
        ; начало программы
        begin:
        ; первая операция - инициализация стека
        ; если этого не сделать, то вызов подпрограммы или прерывания
        ; не вернет управление обратно
        ; указатель на конец стека устанавливается на последний адрес внутреннего ОЗУ –
        ;RAMEND

        ldi r16, low(RAMEND)
        out spl, r16
        ldi r16, high(RAMEND)
        out sph, r16

        ; для того, чтобы управлять светодиодами, подключенными к выводам PD6 и PD7,
        ; необходимо объявить эти выводы выходными.
        ; для этого нужно записать "1" в соответствующие биты регистра DDRD
        ;(DataDiRection)
        l
        ldi r16, (1<<PD6) | (1<<PD7)
        out DDRD, r16

        ; основной цикл программы
loop:
        ldi r16, (1<<PD6) ; светится один светодиод
        out PORTD, r16

        rcall delay ; задержка

        ldi r16, (1<<PD7) ; светится второй светодиод
        out PORTD, r16

        rcall delay ; задержка

        rjmp loop ; повторение цикла

        ; процедура задержки
        ; примерно полсекунды при частоте 7,37 МГц
        ; три пустых вложенных цикла соответственно
delay:
        ldi r16,30 ; 30
delay1:
        ldi r17, 200 ; 200

delay2:
        ldi r18, 200 ; и еще 200 циклов
delay3:
        dec  r18
        brne delay3
        dec  r17
```

```
brne delay2
dec  r16
brne delay1
ret ; возврат в главную программу
```

Проект может состоять из нескольких файлов, при этом один файл назначается основным. Все операции удобно производить, используя контекстную кнопку мыши. После подключения исходного файла окна имеют следующий вид.

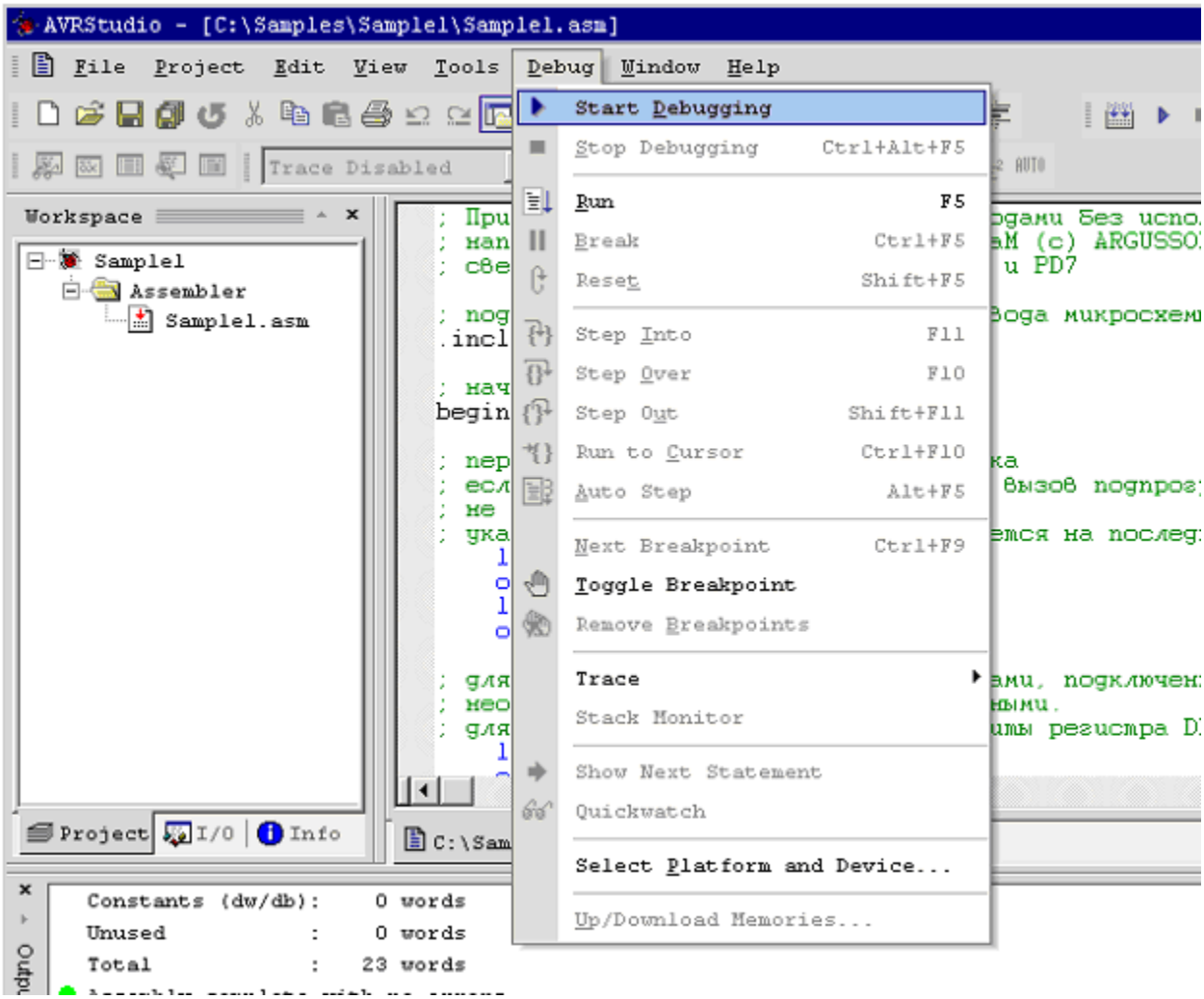


Компиляция проекта выполняется командой \Project\Build или нажатием клавиши F7. Процесс компиляции отображается в окне «Output». Это окно можно «вытащить» командой \View\Output.

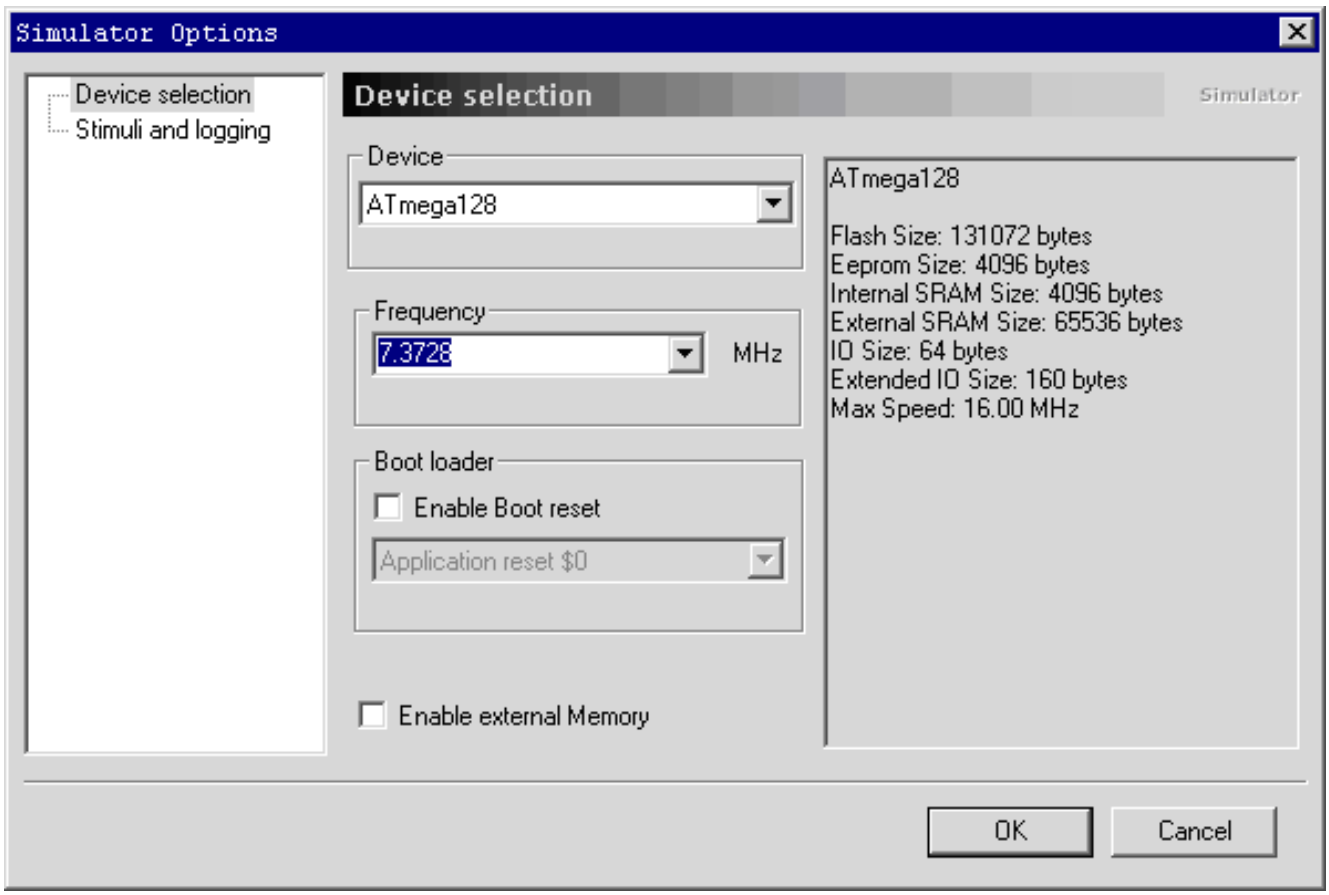


В принципе, мы уже получили выходной файл в формате .hex, который можно загружать во флэш-память микросхемы и наблюдать работу нашего приложения: перемигивание светодиодов. Однако нам нужно показать

полный цикл работы в среде «AVR-Studio», поэтому мы переходим к стадии отладки. Это делается при помощи команды \Debug\Start Debugging.

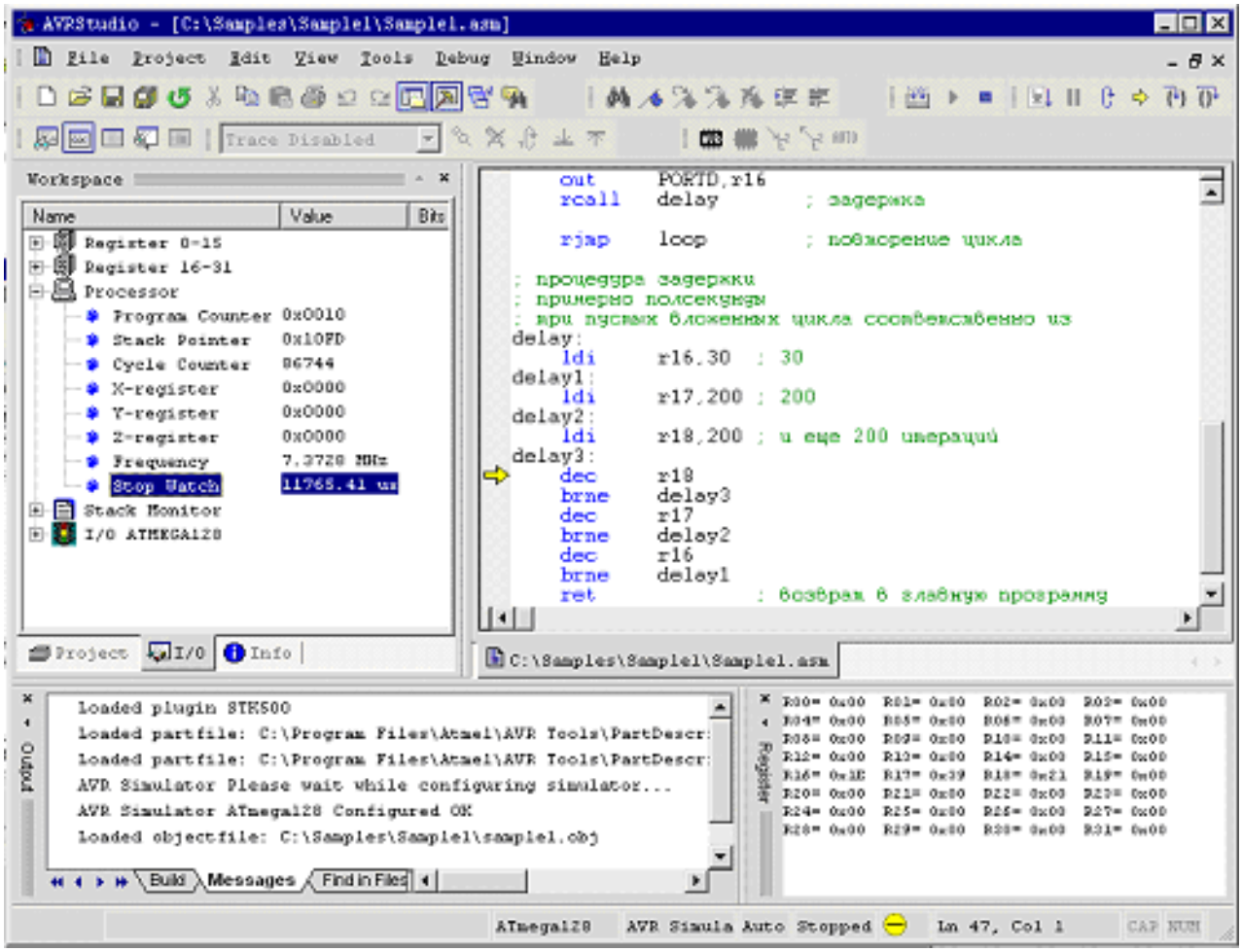


Теперь устанавливаем в окне Simulator Options” частоту кварца 7,3728 MHz для точного измерения времени выполнения программы .



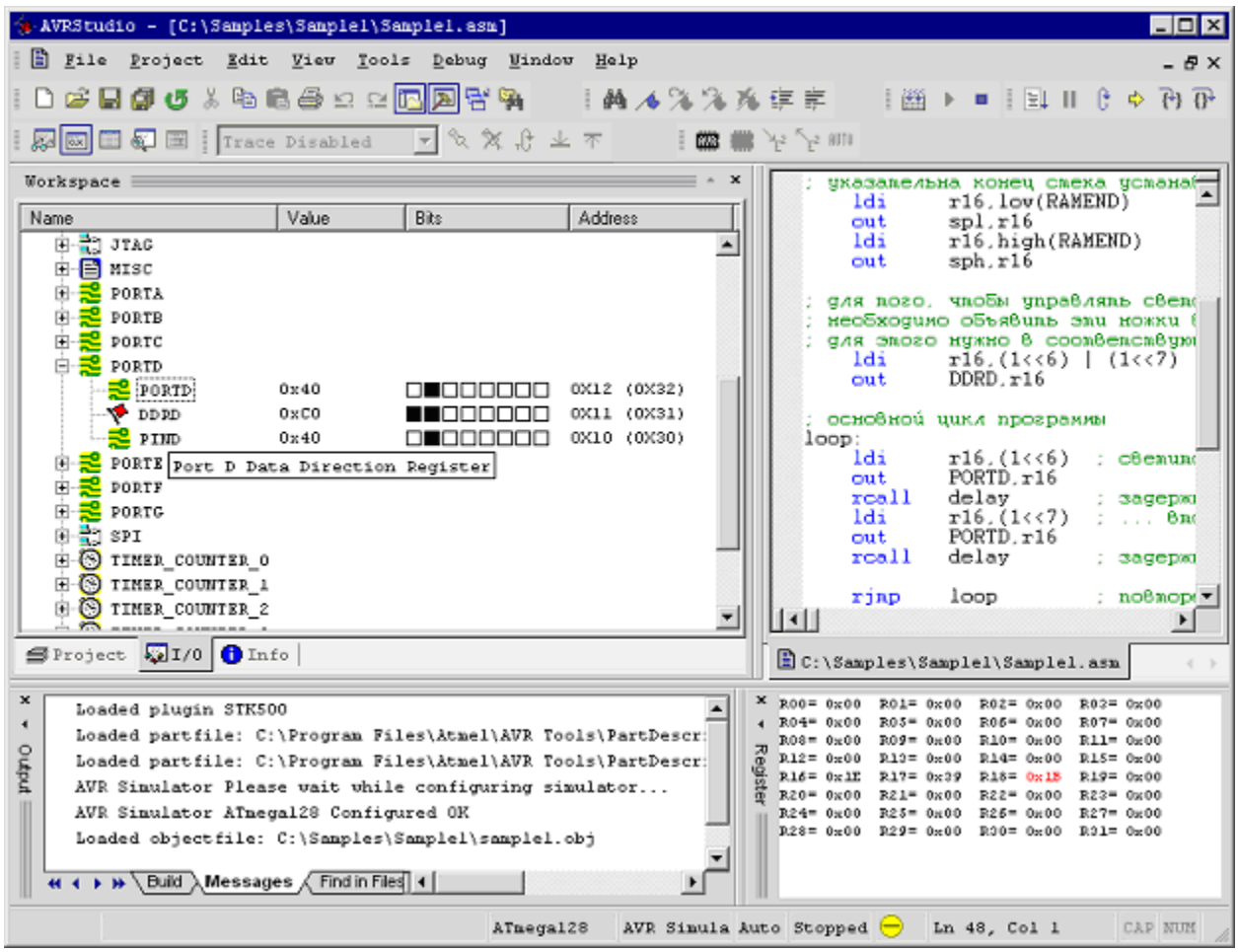
Остальные опции следует оставить без изменения. Программу удобно выполнять в пошаговом режиме при помощи мыши или клавиши F11.

Пакет «AVR-Studio» содержит мощные средства для просмотра и редактирования состояния внутренних регистров и портов ввода/вывода отлаживаемого микроконтроллера, а также время, выполнения программы . Доступ к ним осуществляется через окно “I/O”.



На самом деле, количество информации, доступное через окна просмотра пакета «AVR-Studio» весьма велико, и для получения достаточного комфорта рекомендуется использовать компьютер с большим монитором, а еще лучше в двухмониторной конфигурации.

Для отладки нашего примера, чтобы получить доступ к битам порта D, надо раскрыть строку I/O ATMEGA128 и затем строку PORTD. Теперь видны все три регистра этого порта, PORTD, DDRD и PIND. Чтобы увидеть поля Value, Bits и Address, придется расширить правую границу окна, потеснив при этом окно с исходным текстом программы.



Теперь, проходя программу в пошаговом режиме, можно видеть изменение текущих состояний этих регистров в поле Bits. Есть возможность оперативного изменения состояния любого бита регистров порта, причем это можно делать либо записью нового кода в поле Value, либо непосредственно, щелкнув мышью на нужном бите регистра.

Для самостоятельных упражнений, предлагается следующая программа, которая отличается от предыдущей тем, что зажиганием светодиодов управляют две кнопки.

; **Пример 2**: «Управление светодиодами от кнопок »
; написан для отладочной платы AS-MegaM
; светодиоды подключены к выводам PD6 и PD7 и через резисторы - на общий провод.
; кнопки - на PE4 и PE5

.include "m128def.inc"

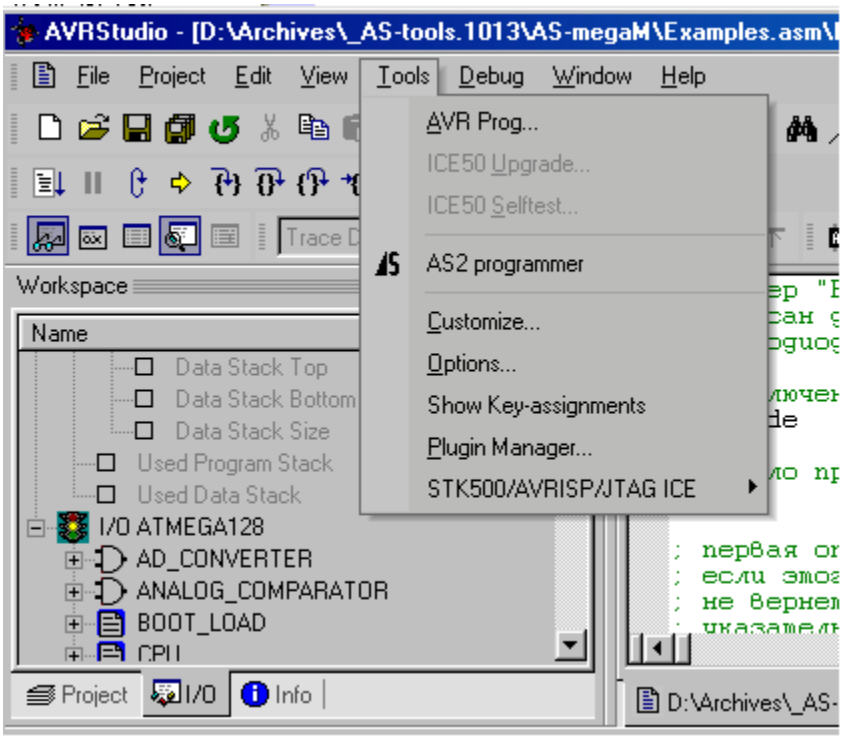
; основная программа
begin:

```
        ; инициализация стека
        ldi r16, low(RAMEND)
        out spl, r16
        ldi r16, high(RAMEND)
        out sph, r16

        ; инициализация светодиодов
        ldi r16, (1<<6) | (1<<7)
        out DDRD, r16

        ; инициализация выводов, к которым подключены кнопки (на вход)
        ; внутренние подтягивающие резисторы подключены
        ; для этого в PORTE нужно установить соответствующие биты в единицы
        ldi r16, (1<<4) | (1<<5)
        out PORTE, r16

        ; а в DDRE - в нули
        ldi r16, 0
        out DDRE, r16

        ; бесконечный цикл
forever:
        in r16, PINE ; теперь в r16 находится текущее "состояние" кнопок
        com r16 ; кнопка "нажимается" нулем, поэтому инвертируем регистр
        lsl r16 ; переносим биты 4,5 в позиции 6,7
        lsl r16 ; и обновляем "показания" светодиодов
         andi r16, (1<<6) | (1<<7)
        out PORTD, r16
        rjmp forever ; цикл выполняется бесконечно
```

Таким образом, на примере простейших программ показаны некоторые возможности пакета «AVR-Studio». Надо понимать, что это лишь первое знакомство, позволяющее быстрее освоиться с базовыми командами пакета. Между тем, возможности рассматриваемого пакета намного шире. Например, здесь можно отлаживать программы, написанные на языках высокого уровня. В частности, Си-компилятор фирмы ImageCraft пользуется отладчиком «AVR-Studio» «как родным». Для этого при компиляции исходного кода надо установить опцию генерации выходного файла в формате, совместимом с «AVR-Studio». При этом появляется возможность производить отладку в исходных кодах.

Еще одна из многих характеристик пакета «AVR-Studio» - возможность подключения внешних программ. Например, для обеспечения вызова

оболочки внутрисхемного программатора AS2 нужно выполнить несколько простых операций:



- В меню Tools главного окна «AVR-Studio» надо выбрать пункт Customize;
- в окне Customize выбрать пункт Tools;
- двойным нажатием кнопки мыши или нажав Insert на клавиатуре, добавить новую команду в список и назвать ее «Программатор AS2»;
- указать путь к исполняемому файлу программатора, введя его непосредственно в поле для ввода "Command", или нажав на кнопку "…" справа от этого поля.

Теперь в меню Tools появился пункт "Программатор AS2".

Средства пакета «AVR-Studio», начиная с версии 4.08, позволяют подключать вспомогательные программы – plugins. Первый plugin для «AVR-Studio» – это программа графического редактора, упрощающая процесс инициализации ЖК-индикатора, которым может непосредственно управлять AVR-контроллер ATmega169. Максимальный логический размер ЖК-индикатора составляет 100 сегментов, каждому элементу индикатора ставится в соответствие бит в специальном регистре контроллера. Чтобы

упростить рутинную процедуру привязки определенных битов к каждому сегменту, можно использовать вышеупомянутую программу.

1.2.8. Задания

1. Используя данное руководство, а также дополнительную информацию, представленную в [1-3], а также на сайте http://www.gaw.ru/html.cgi/txt/doc/micros/avr/arh128/index.htm выполнить следующие задания:

Задание 1. Управление светодиодами от кнопок, используя прерывание:

1. Взяв за основу *пример 2* составить блок-алгоритм программы.
2. Написать программу на ассемблере.
3. Провести отладку программы на симуляторе “AVR-Studio”.
4. Загрузить программу во флэш-память микроконтроллера с помощью программатора “AS-2”.
5. Проверить работу приложения.

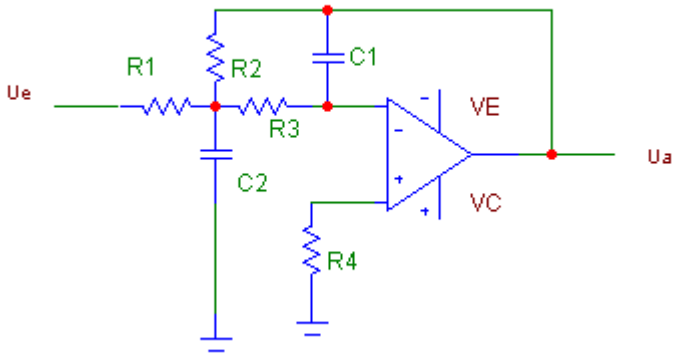
Задание 2. Управление частотой мигания светодиодов по интерфейсу RS-232:

1. В качестве консольной программы управления COM портом использовать утилиту «Com Port Toolkit» (ознакомиться с ее применением).
2. Для включения/выключения светодиодов за основу можно взять процедуры из *примера 2*.
3. Составить блок-алгоритм и написать программу на ассемблере.
4. Провести отладку программы на симуляторе “AVR-Studio”.
5. Загрузить программу во флэш-память микроконтроллера с помощью программатора “AS-2”.
6. Проверить работу приложения.

Задание 3. Формирование треугольного периодического сигнала с помощью таймера в режиме ШИМ:

1. Изучив работу счетчика-таймера в режиме широтно-импульсной модуляции (ШИМ), составить блок-алгоритм программы.
2. Написать программу на ассемблере.

3. Провести отладку программы на симуляторе “AVR-Studio”.
4. Загрузить программу во флэш-память микроконтроллера с помощью программатора “AS-2”.
5. Выход таймера подключить к фильтру Баттерворта (нижних частот, второго порядка), который нужно собрать на монтажной плате «WishBoard».



Усиление: $A_0 = -\frac{R_2}{R_1}$; граничная частота: $f_g{}^2 = \frac{1}{4\pi^2 C_1 C_2 R_2 R_3}$; $C2 \geq 4C_1$

Сигнал наблюдать с помощью цифрового осциллографа “GDS-2102”.

Задание 4. Управление 3-х фазным шаговым электродвигателем:

1. С помощью счетчика-таймера сформировать опорную частоту, определяющую скорость вращения шагового двигателя.
2. Получить 3-х фазную последовательность прямоугольных импульсов со сдвигом между фазами 120 градусов. В качестве тактовой частоты используется опорная частота со счетчика-таймера.
3. С помощью USART контроллера и консоли на основе утилиты «ComPort Toolkit» реализовать управление шаговым двигателем, задавая скорость, число шагов и направление вращения.
4. Составить блок-алгоритм работы приложения.
5. Написать программу на ассемблере.
6. Провести отладку программы на симуляторе “AVR-Studio”.

7. Загрузить программу во флэш-память микроконтроллера с помощью программатора “AS-2”.
8. Проверить управляющие сигналы с выходов порта (частоту, фазовый сдвиг) с помощью цифрового осциллографа «GDS-2102».
9. Подключить драйвер двигателя ШД 300/300 к выходам контроллера, к обмоткам двигателя и источнику питания.
10. Проверить работу привода.

Приложение 1. Описание платы AS-megaM.

Приложение 2. Ассемблер для 8-разрядных AVR-контроллеров.

1.3. Микроконтроллеры XMEGA

Дальнейшее развитие и усовершенствование восьмиразрядной архитектуры AVR привело к появлению на рынке новых микроконтроллеров, которые получили название XMEGA. Название XMEGA также подразумевает некоторое разнообразие типов данной архитектуры. Мы в нашем практикуме остановимся на семействе XMEGA A3, - семейство высокоэффективных, малопотребляющих и с обширным набором устройств ввода вывода 8/16-ти разрядные КМОП микроконтроллеры, выполненные на основе улучшенной RISC-архитектуры AVR. За счет выполнения большинства инструкций за один цикл синхронизации микроконтроллеры XMEGA A3 достигают производительности близкой к 1 MIPS/МГц, что дает возможность разработчику оптимизировать соотношение потребляемой мощности и быстродействия обработки.

Центральное процессорное устройство (ЦПУ, CPU) AVR сочетает набор инструкций (совпадающий, в основном, с системой команд семейства MEGA) с 32 рабочими регистрами общего назначения. Все 32 регистра напрямую подключены к арифметико-логическому устройству (ALU), что позволяет выполнить за один цикл синхронизации инструкцию, осуществляющую доступ к двум разным регистрам. Результирующая архитектура отличается более эффективным выполнением кода программы, добиваясь производительности в несколько раз превышающей производительность обычных CISC-микроконтроллеров или микроконтроллеров с одним аккумулятором.

МК XMEGA версии A3 интегрируют следующие функциональные блоки (рис.1.3.1): внутрисистемно-программируемая Flash-память с возможностями чтения во время записи, EEPROM, статическое ОЗУ (SRAM), четырехканальный контроллер ПДП, восьмиканальная система событий, программируемый многоуровневый контроллер прерываний, 50 линий ввода/вывода общего назначения, 16-ти разрядные часы реального времени (RTC), семь универсальных 16-ти разрядных таймеров-счетчиков с

режимами сравнения и возможностями широтно-импульсной модуляции (PWM), семь интерфейсов USART, два двухпроводных интерфейса (TWI), три последовательных интерфейса SPI, *ускоритель криптографических алгоритмов AES и DES*, два 8-ми канальных 12-ти разрядных АЦП с опциональным дифференциальным входом и программируемым усилением, один 2-канальный 12-ти разрядный цифро-аналоговый преобразователь (DAC), четыре аналоговых компаратора с поддержкой оконного режима, программируемый сторожевой таймер с отдельным внутренним генератором, точные внутренние генераторы с ФАПЧ и предделителем, а также программируемый супервизор питания.

Для программирования, отладки и тестирования предусмотрены двух проводной быстродействующий интерфейс PDI и IEEE 1149.1 (интерфейс совместимый с JTAG - **J**oint **T**est **A**ction **G**roup).

МК XMEGA A3 поддерживают пять программно-выбираемых экономичных режимов работы. В режиме IDLE останавливается ЦПУ, но продолжают работать SRAM, контроллер DMA, система событий, контроллер прерываний и все УВВ. Есть режим POWER-DOWN, когда сохраняется содержимое SRAM и регистров, но генераторы отключены, что в свою очередь блокирует работу всех встроенных УВВ вплоть до следующего прерывания модуля TWI, или прерывания по изменению состояния выводов, или же до сброса МК. В режиме POWER-SAVE остается в работе асинхронный счетчик реального времени (RTC), что позволяет микроконтроллеру продолжать счет времени в даже тогда, когда остальная его часть бездействует (спит). В режиме STANDBY сохраняется функционирование кварцевого генератора, а остальная часть МК отключена. Этот режим отличается наименьшим временем, которое требуется для возобновления нормальной работы МК. В режиме EXTENDED STANDBY, остаются в работе, как основной генератор, так и асинхронный таймер. Для предоставления еще более широких возможностей снижения потребляемой мощности, предусмотрена опциональная возможность

включения/отключения синхронизации каждого отдельного УВВ, как в активном режиме, так и в режиме IDLE. МК XMEGA выпускаются по разработанной Atmel технологии энергонезависимой памяти высокой плотности. Flash-память программ допускает внутрисистемное перепрограммирование через интерфейс PDI или JTAG. Программа «загрузчик» микроконтроллера может использовать любой интерфейс для загрузки программного кода в сектор прикладной программы Flash-памяти. Она хранится в загрузочном секторе Flash-памяти и продолжает исполняться даже во время обновления сектора прикладной программы, таким образом, обеспечивая полную поддержку «чтения во время записи». Благодаря сочетанию 8/16-ти разрядного RISC ЦПУ с внутристемно самопрограммируемой Flash-памятью, микроконтроллеры XMEGA A3 компании Atmel являются универсальным и выгодным в ценовом плане инструментом для реализации многих встраиваемых приложений.

Микроконтроллеры XMEGA A3 поддерживаются полным набором аппаратных и программных средств для проектирования: Си-компиляторы, макроассемблеры, программы отладчиков/симуляторов, программаторы и стартовые наборы.

1.3.1. Основные характеристики МК XMEGA

- Высокоэффективный и малопотребляющий 8/16-ти разрядный микроконтроллер AVR XMEGA
- Энергонезависимая память программ и данных:
    - 64…256 кбайт внутрисистемно-самопрограммируемой Flash-памяти
    - 4…8 кбайт загрузочного сектора с отдельными битами защиты
    - 2…4 кбайт EEPROM
    - 4…16 кбайт SRAM
- Особенности устройств для работы с периферией:

- Четырехканальный контроллер ПДП с поддержкой внешних запросов
- Восьмиканальная система событий
- Семь 16-ти разрядных таймеров-счетчиков:
- Четыре таймера-счетчика с 4 выходами блоков сравнения или 4 входами блоков захвата
- Три таймера-счетчика с 2 выходами блоков сравнения или 2 входами блоков захвата
- Возможность повышения разрешающей способности у всех таймеров-счетчиков
- Один таймер-счетчик с расширенными возможностями генерации прямоугольных импульсов
- Семь интерфейсов USART
- Поддержка IrDA у USART1
- Ускоритель криптографических алгоритмов AES и DES
- Два двухпроводных интерфейса с возможностями обнаружения совпадения двух адресов (I2C- и SMBus-совместимые)
- Три интерфейса SPI
- 16-ти разрядный таймер-счетчик с отдельным генератором
- Два восьмиканальных 12-и разрядных АЦП на частоту преобразования 2 МГц
- Один двухканальный 12-ти разрядный ЦАП на частоту преобразования 1 МГц
- Четыре аналоговых компаратора с функцией оконного компаратора
- Внешние прерывания на всех линиях ввода-вывода общего назначения
- Программируемый сторожевой таймер с отдельным встроенным сверхмаломощным генератором

- Специальные возможности микроконтроллера:

    - Схема сброса при подаче питания и супервизор питания
    - Внутренние и внешние источники синхронизации схемы ФАПЧ
    - Программируемый многоуровневый контроллер прерываний
    - Экономичные режимы работы: IDLE, POWER_DOWN, STANDBY, POWER-SAVE, EXTENDED STANDBY
    - Улучшенные интерфейсы программирования, тестирования и отладки
        - Интерфейс JTAG (IEEE 1149.1-совместимый) для тестирования, отладки и программирования
        - Интерфейс PDI для программирования, тестирования и отладки
- Количество линий ввода-вывода и корпуса:
    - 50 программируемых линий ввода/вывода
    - 64-выводной корпус TQFP
    - 64-контактный корпус QFN
- Рабочее напряжение:
    - 1.6…3.6В
- Градации быстродействия:
    - 0…12 МГц/1.6…3.6В
    - 0…32 МГц/2.7…3.6В

1.3.2. Архитектура

ЦПУ AVR XMEGA. Назначением ЦПУ является гарантированное и корректное выполнения программы. То есть ЦПУ должно быть способно осуществлять доступ к запоминающим устройствам, выполнять вычисления и управлять УВВ. Механизм обработки прерываний будет описан в отдельном разделе.

Особенности ЦПУ AVR XMEGA:

- 8/16-ти разрядная высокоэффективная RISC-архитектура AVR (рис.1.3.2)
- 138 инструкций
- Аппаратный умножитель
- 32 восьми разрядных регистра, напрямую подключенных к ALU
- Стек в RAM
- Указатель стека доступен в пространстве памяти ввода-вывода

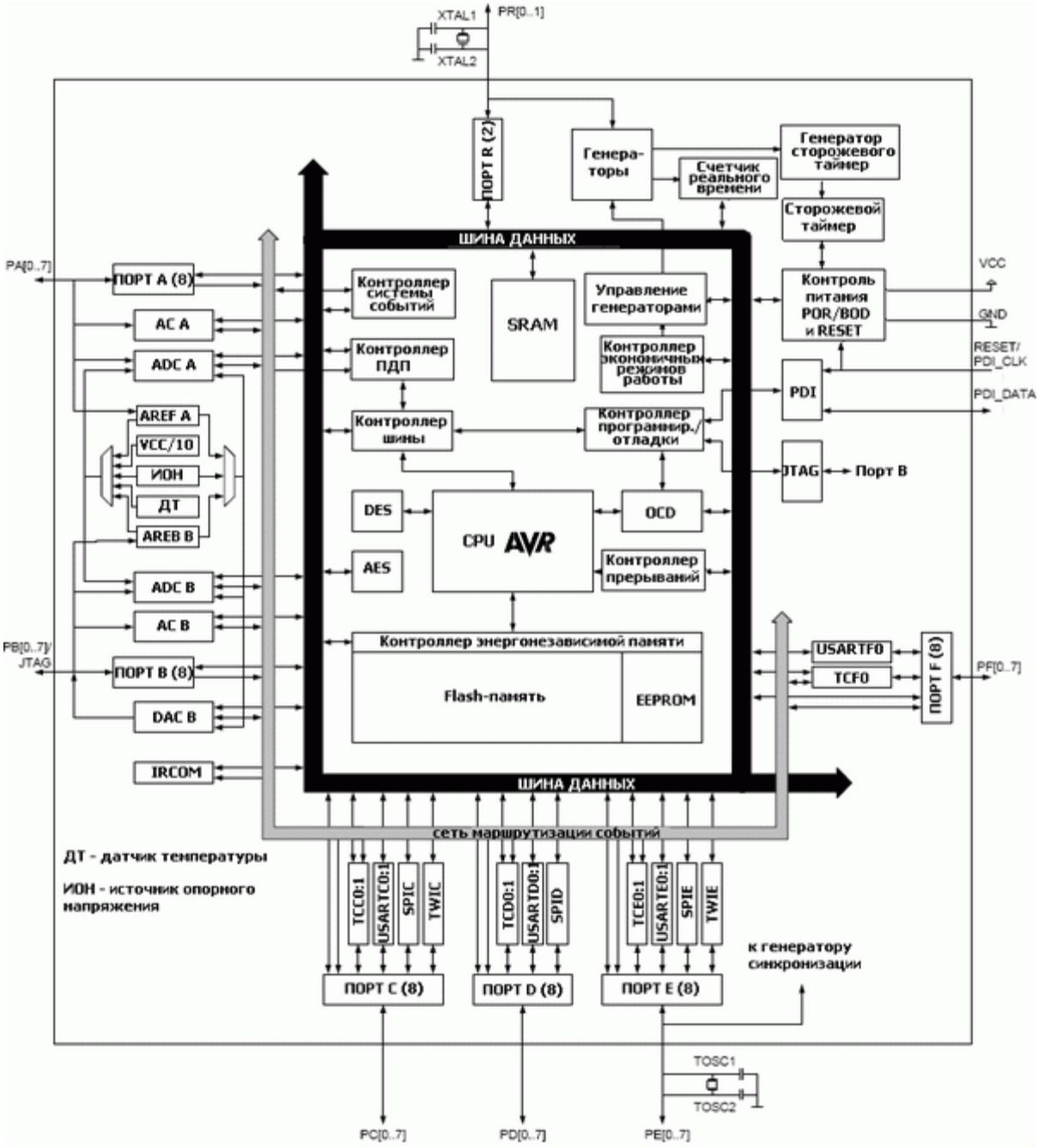


Рис. 1.3.1. Функциональная схема микроконтроллера

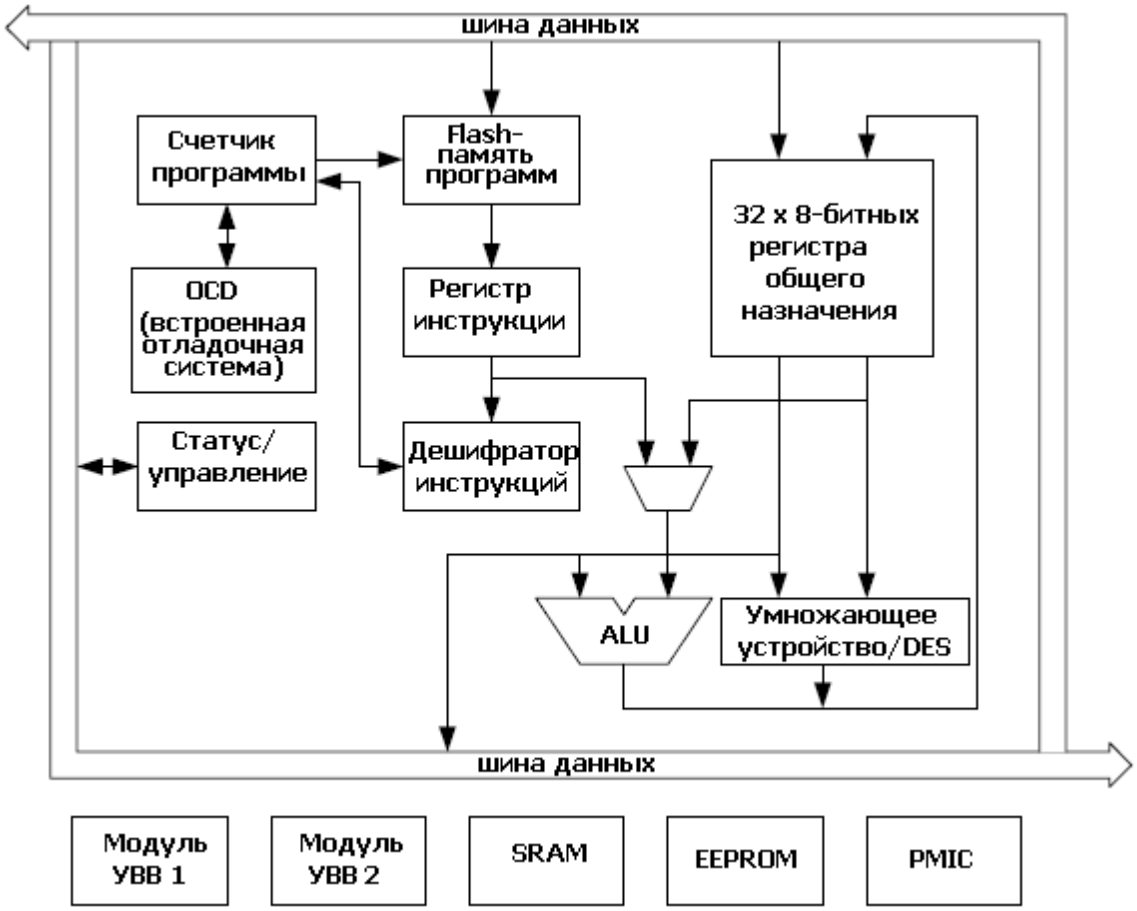


Рис. 1.3.2. Функциональная схема ЦПУ

Микроконтроллеры XMEGA также как и микроконтроллеры ATmega имеют Гарвардскую архитектуру. Инструкции, хранящиеся в памяти программ, выполняются с использованием одноуровневого конвейера. Это означает, что одновременно с выполнением текущей инструкции будет осуществляться выборка следующей инструкции.

Такой механизм позволяет выполнять по одной инструкции за каждый цикл синхронизации. В качестве памяти программ выступает внутрисистемно-перепрограммируемая Flash-память.

1.3.3. Память и ввод/вывод

Flash-память программ. Свойства:

- Одно линейное адресное пространство
- Внутрисистемное программирование

- Поддержка самопрограммирования под управлением программы загрузчика
- Сектор прикладной программы для хранения кода программы
- Сектор таблицы приложения для хранения данных или кода прикладной программы
- Загрузочный сектор для хранения прикладной программы или программы загрузчика
- Отдельные биты защиты и защита всех секторов
- Встроенная быстродействующая проверка CRC для выбранного сектора flash-памяти программ

Память данных:

- Одно линейное адресное пространство
- Одноцикловый доступ со стороны ЦПУ
- SRAM
- EEPROM
    - Побайтный и постраничный доступ
    - Опциональное распределение памяти для прямого чтения и записи

Ввод-вывод

в адресном поле памяти располагаются:

- Конфигурационные регистры и регистры статуса для всех УВВ и периферийных модулей
- Регистры общего назначения с 16-ти разрядным доступом для хранения глобальных переменных или флагов

Арбитраж шины:

- Безопасная и детерминистическая обработка приоритетов ЦПУ и контроллера DMA
- Отдельные шины для SRAM, EEPROM, памяти ввода-вывода и внешней памяти
- Одновременный доступ к шинам для ЦПУ и контроллера DMA

Область памяти с запрограммированным производителем сигнатурным кодом:

- Идентификационный код для каждого типа микроконтроллера
- Серийный номер для каждого микроконтроллера
- Калибровочные байты генератора
- Калибровочные данные АЦП, ЦАП и датчика температуры
- Сигнатурный код пользователя
- Размер равен одной странице flash-памяти
- Возможность программного считывания или программной записи
- Остается неизменным после операции стирания всей памяти

1.3.4. Система событий

Система событий - набор возможностей, предназначенных для организации внутренней связи. С ее помощью можно добиться автоматического запуска действий в одном или нескольких УВВ при изменении состояния в другом УВВ. Какие именно изменения в УВВ приводят к запуску действий в других УВВ, задается программно. Данная система, хотя и весьма простая, но достаточно эффективная. С ее помощью можно организовать автономную совместную работу нескольких УВВ, не используя для этого прерывания, ЦПУ или каналы DMA.

Факт изменений в УВВ определяется как событие. События обычно полностью совпадают с условиями прерываний УВВ. Для соединения между собой событий разных УВВ предусмотрена специальная сеть маршрутизации, которая называется сетью маршрутизации событий или матрицей межсоединений событий. На рис.1.3.3 представлена обобщенная функциональная схема системы событий с сетью маршрутизации событий и связываемых ею УВВ. Данная система отличается высокой гибкостью. Ее можно использовать для простой маршрутизации сигналов, функций выводов или для упорядочивания событий. Максимальная задержка между генерацией события в одном УВВ и запуском действий в другом или других УВВ составляет не более двух циклов синхронизации ЦПУ.

Свойства:

- Связь и передача сигналов между внутренними УВВ с минимальной задержкой.

- Независимая работа ЦПУ и DMA.
- 8 каналов событий позволяют пропускать до 8 сигналов одновременно.
- События могу генерироваться:
    - таймерами-счетчиками (TCxn);
    - счетчиком реального времени (RTC);
    - аналогово-цифровыми преобразователями (ADCx);
    - аналоговыми компараторами (ACx);
    - портами ввода-вывода (PORTx);
    - системой синхронизации (ClkSYS);
    - программой (ЦПУ);
- События могут использоваться:
    - таймерами-счетчиками (TCxn);
    - аналогово-цифровыми преобразователями (ADCx);
    - цифро-аналоговыми преобразователями (DACx);
    - портами ввода-вывода (PORTx);
    - DMA-контроллером (DMAC);
    - модулем инфракрасной связи (модуль IRCOM).
- Одно и то же событие может использоваться несколькими УВВ для синхронизированной их работы.
- Дополнительные возможности:
    - ручная программная генерация событий (ЦПУ);
    - квадратурная дешифрация;
    - цифровая фильтрация.
- Функционирует в активном режиме работы МК и в режиме IDLE.

Система событий функционирует в двух режимах работы МК: активном и IDLE. Сеть маршрутизации событий может напрямую связывать между собой аналогово-цифровые преобразователи, цифро-аналоговые преобразователи, аналоговые компараторы (ACx), порты ввода-вывода (PORTx), счетчик реального времени (RTC), таймеры-счетчики (T/C) и

модуль инфракрасной связи (IRCOM). События можно также генерировать программно.

Все события всех УВВ всегда передаются по сети маршрутизации событий. Она состоит из восьми программно-конфигурируемых мультиплексоров, позволяющих задать, какое именно событие необходимо соединить с каналом событий.

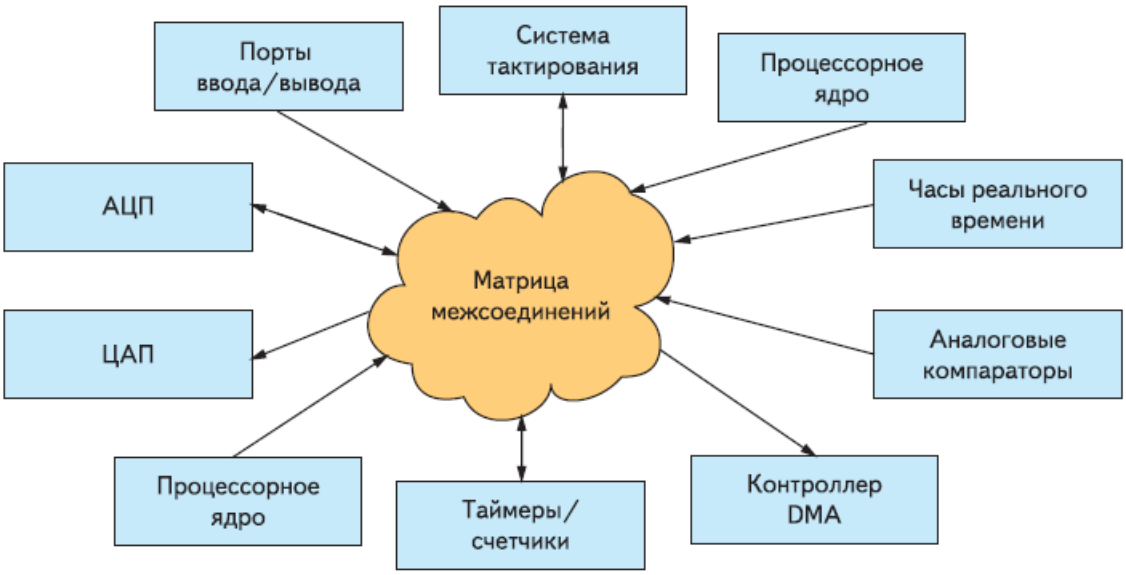


Рис. 1.3.3. Функциональная схема системы событий

Все восемь каналов событий соединены с УВВ, допускающих возможность использования событий. Кроме того, у каждого из этих УВВ предусмотрена возможность настройки использования событий из одного или нескольких каналов событий, приводящие к автоматическому запуску программно-заданного действия.

Полная структурная схема системы событий микроконтроллеров XMEGA приведена на рисунке 1.3.4. Здесь наглядно видно, как организована сеть маршрутизации, и каким образом мультиплексоры могут коммутировать события между генераторами и приемниками. Каждый мультиплексор имеет два регистра управления, доступные программисту. Регистр CHnMUX определяет, какое из входящих на мультиплексор событий коммутируется на соответствующий выходной канал событий Event Channel (n = 0 …7). Регистр

CHnCTRL используется на выходе мультиплексора для организации дополнительных функций — фильтрации и квадратурного декодирования. Наличие восьми мультиплексоров означает, что у разработчика имеется возможность «развести» до восьми событий одновременно. Также можно «распараллелить» одно и то же событие на несколько мультиплексоров. Структура сети маршрутизации событий в Event System одинакова для всех микроконтроллеров XMEGA. Конечно, представители разных подсемейств XMEGA имеют различные наборы периферии, но с точки зрения Event System это означает, что отсутствующий периферийный модуль не может генерировать и принимать события.

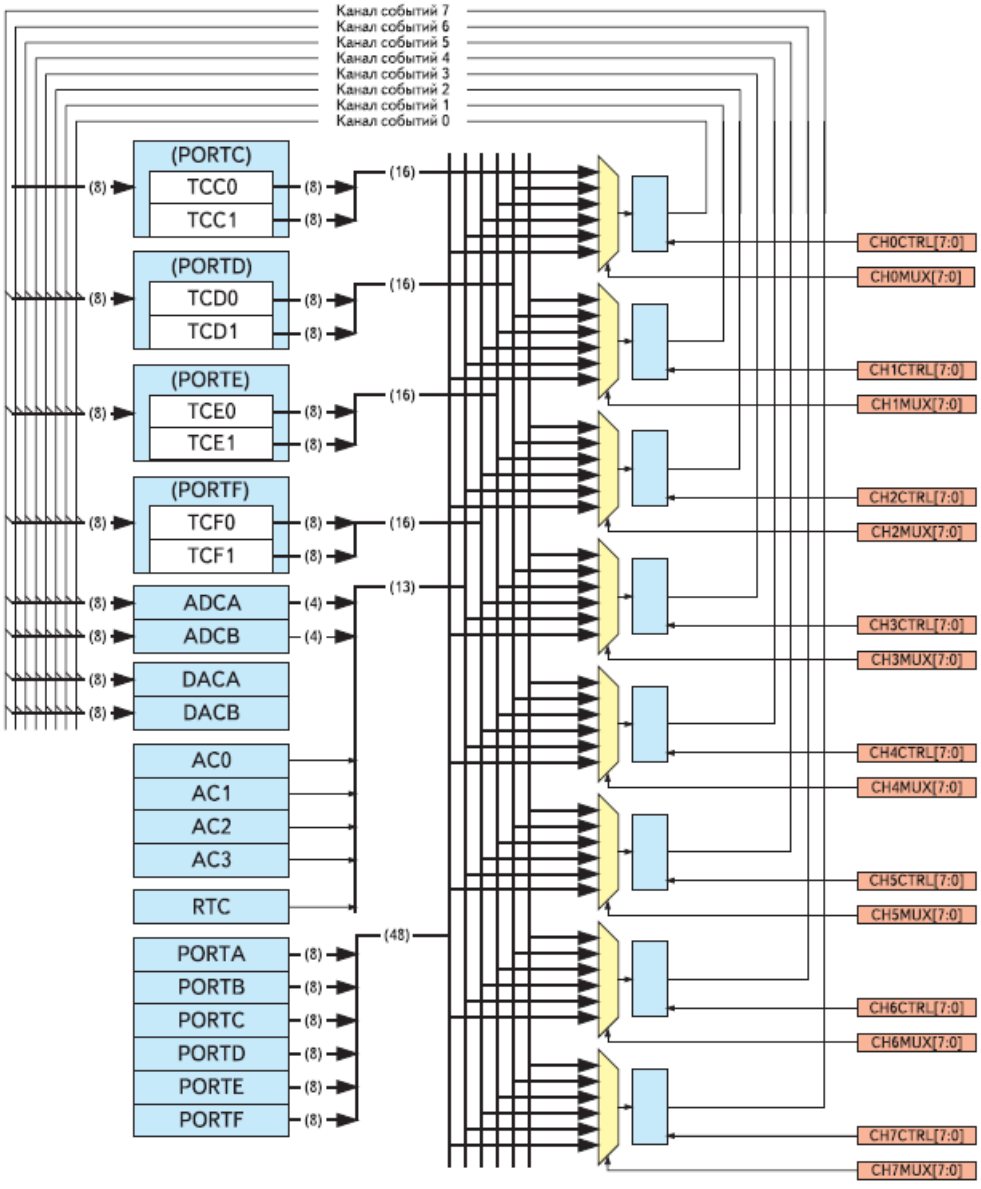


Рис. 1.3.4. Структурная схема системы событий XMEGA

1.3.5. Программирование

Atmel Studio 6.0

Программная среда *Atmel Studio* от компании Atmel — это очень полезный инструмент для разработки и отладки приложений для трех архитектур микроконтроллеров, выпускаемых корпорацией Atmel. «Atmel Studio 6» поддерживает кристаллы с ядрами AVR (в том числе XMEGA), AVR32 UC3 и ARM/Cortex-M3 (серий ATSAM3).

«Atmel Studio 6» включает в себя набор библиотек Atmel Software Framework, где в исходных кодах реализованы библиотеки для работы со всеми периферийными устройствами микроконтроллеров этих трёх архитектур, а для упрощения их освоения в состав «Atmel Studio 6.0» включено более 1000 примеров проектов, использующих эти библиотеки. В «Atmel Studio 6.0» также реализован функционал просмотра содержимого набора библиотек Atmel Software Framework (ASF Explorer) и интегрирована библиотека Qtouch composer, служащая для создания сенсорных интерфейсов. Программная среда Atmel Studio построена на базе Microsoft Visual Studio Shell 10 с надстройкой Visual Assist, что вкупе обеспечивает удобную разработку проектов как на Си, так и на C++. Пользователь на свое усмотрение вводит программу в действие или же тестирует ее и исправляет возможные ошибки. Для этого он может целенаправленно управлять ходом выполнения программы и приостанавливать ее в любых местах с целью контроля (например, проверять изменения, внесенные в содержимое регистров последними командами, или же просматривать ячейки памяти и содержимое регистров периферийных устройств, наподобие приемопередатчика UART, интерфейса SPI, таймера и т. д).

1.3.5.1. Работа в «Atmel Studio 6.0»

Запускаем Atmel Studio. После запуска наблюдаем окно со стартовой страницей, предлагающее создать новый проект или открыть существующий, либо открыть пример проекта из ASF.

1. Выбираем создание нового проекта. В открытом окне выбираем язык программирования, на котором будет написана программа.

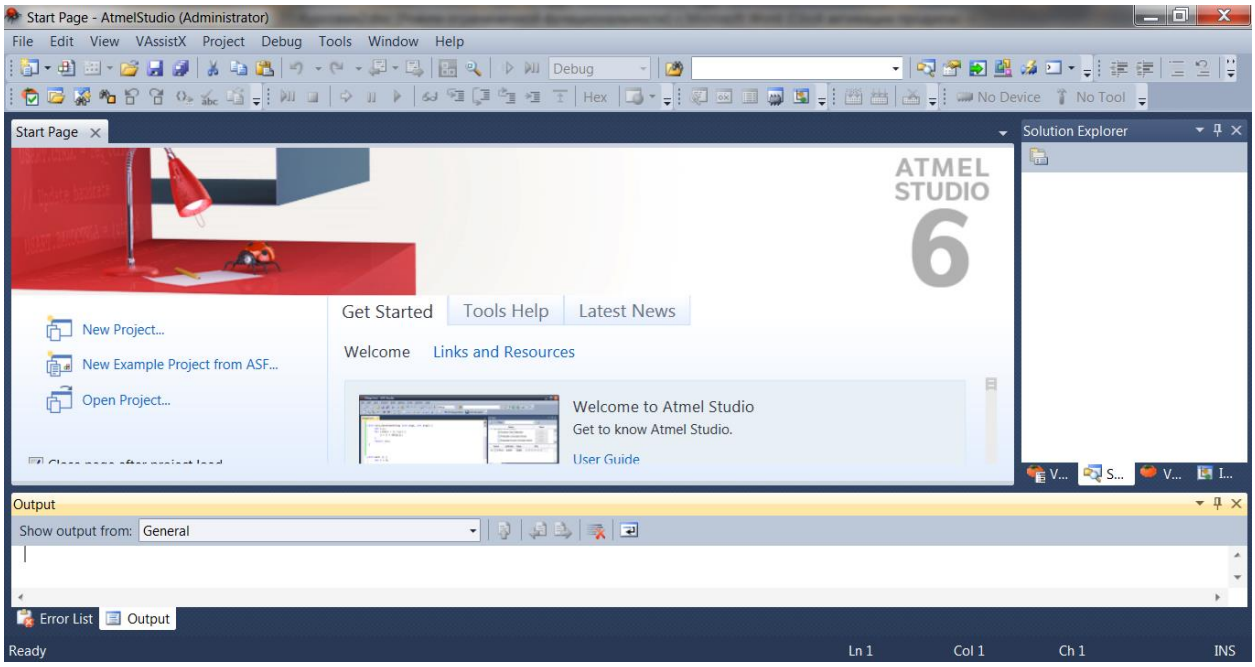


Рис. 1.3.5. Окно Atmel Studio со стартовой страницей

2. На вкладке «AtmelBoards» находим необходимую плату с микроконтроллером. Задаем имя проекта (Name) и расположение (Location). Нажимаем клавишу «OK».

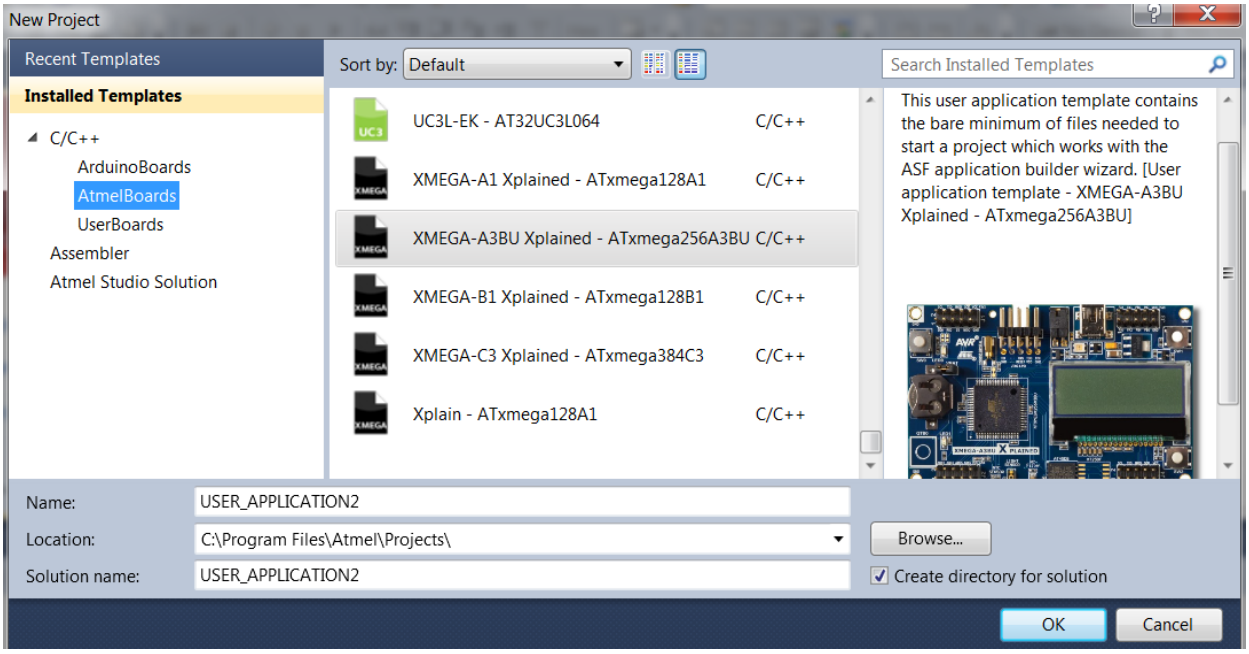


Рис. 1.3.6. Создание нового проекта

В окне исходного кода пишем текст программы на языке Си.

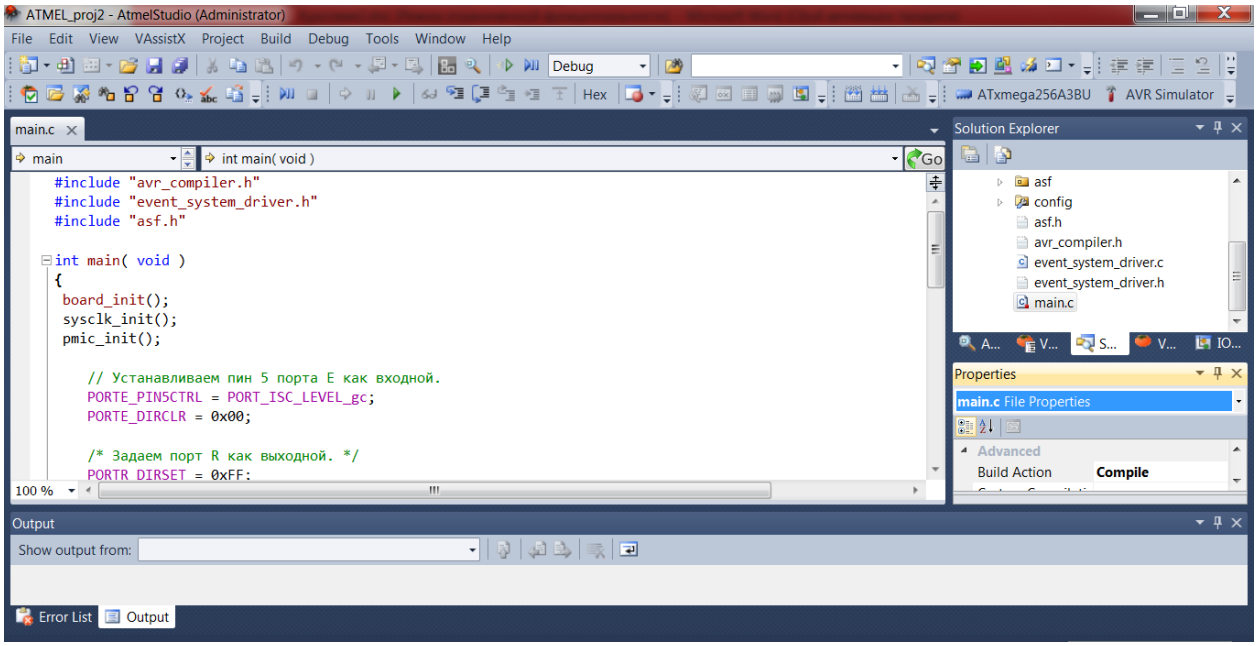


Рис. 1.3.7. Редактирование программы

3. Теперь запускаем программу на компиляцию командой меню «Build → Build Solution». В окне «Output»: получаем сообщение, что компиляция прошла успешно (Build succeeded). Далее можно посмотреть, как работает программа с помощью отладчика. Выполним команду меню «Debug → Continue»:

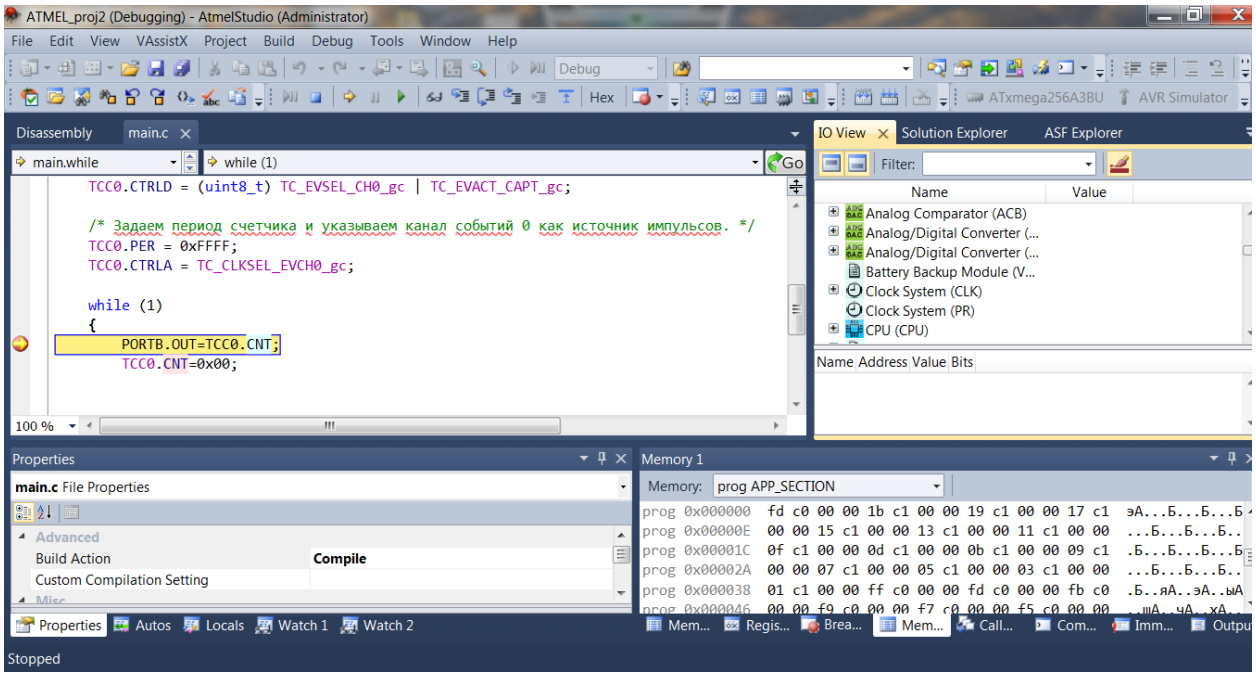


Рис. 1.3.8. Отладка

Также появилась возможность использовать команды отладки:

Start Debugging (F5) – Начать отладку

Stop Debugging (Ctrl+Shift+F5) Остановить отладку

Reset (SHIFT+F5) – Сброс

Step Into (F11) – Шаг программы

Step Over (F10) – Шаг вперед через инструкцию

Run to Cursor (CTRL+F10) – Запустить с позиции курсора

Для того, чтобы запрограммировать hex-файл в микроконтроллер, необходимо воспользоваться программой Flip 3.4.7 фирмы Atmel, которая позволяет выполнить «заливку прошивки» через USB-соединение.

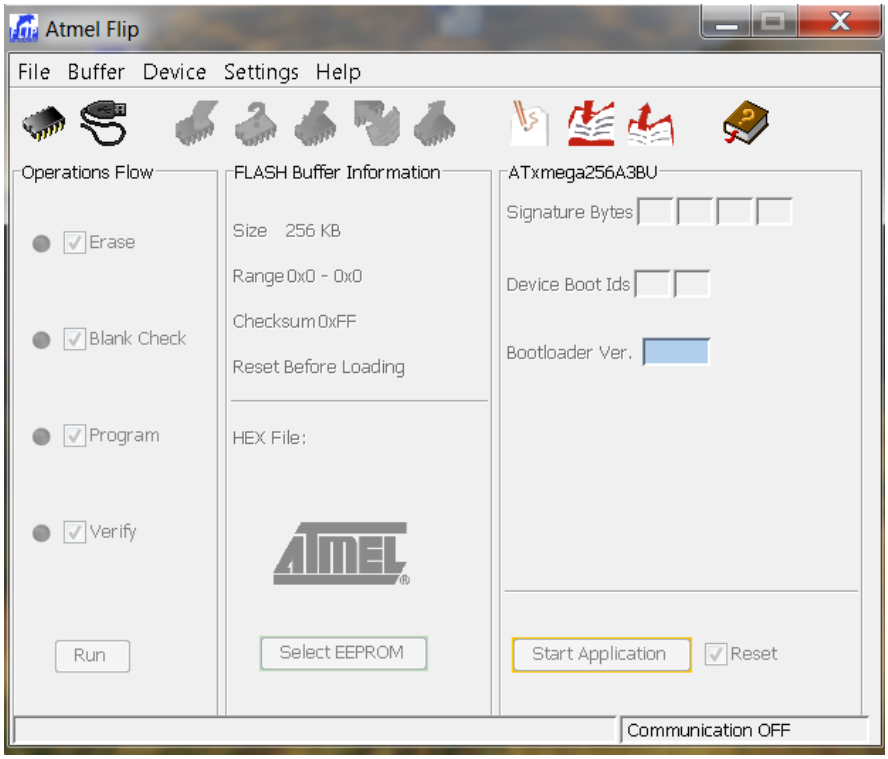


Рис.1.3.9. Flip

Как уже отмечалось, система команд и функции МК AVR XMEGA во многом схожи с функциями и соответствующими командами для МК AVR ATmega. Поэтому в лабораторном практикуме на основе этого типа МК мы будем изучать только то, что является новым для семейства AVR, а именно систему событий и, конечно, более «продвинутую» среду разработки «Atmel

Studio 6». В качестве аппаратной базы в данном практикуме используем модуль «XMEGA-A3BU XPLAINED».

1.3.6. Задания по XMEGA и «Atmel Studio 6»

Задание 1. В среде «Atmel Studio 6», используя пример «TC example 1 – XMEGA-A3BU-Xplaned» из Atmel Software Framework (ASF), выполнить:

a) загрузить проект и построить решение;

b) с помощью программатора «Atmel Flip 3.4.7» загрузить программу в модуль, запустить приложение и убедиться, что оно работает;

c) в окне редактора открыть код tc_exmple1.c, пояснить – каким образом происходит инициализация модуля (тактирование, контроллер прерываний, порты ввода/вывода, таймер …) и, изменяя режимы тактирования МК и содержимое регистров захвата таймера, изменить частоту и скважность свечения светодиодов.

Задание 2. На основе приложения «TC example 1 – XMEGA-A3BU-Xplaned» показать работу системы событий XMEGA: сконфигурировать кнопку SW1 (SW2, SW3) как источник событий (сигналов) поступающих на вход тактирования таймера; настроить таймер таким образом, чтобы нажимая на кнопку можно было видеть переключения светодиодов.

Для выполнения этого задания использовать ссылку [4] (раздел – система событий) и «XMEGA-A3BU XPLAINED», а для более детального изучения документ XMEGA AU MANUAL (doc8331.pdf).

К отчету о выполнении задания (коду программы) приложить блок-алгоритм, пример которого приводится ниже.

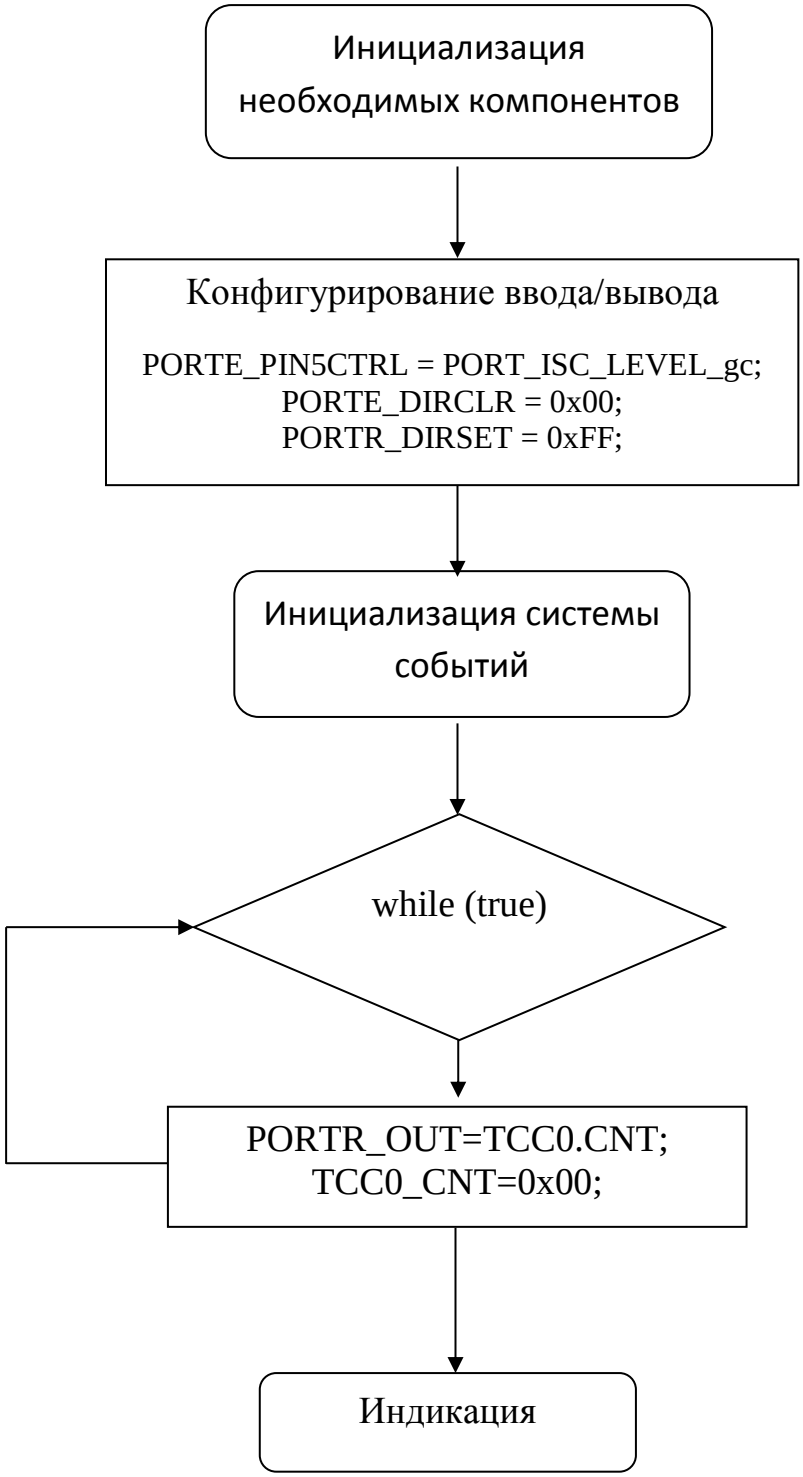


Рис. 1.3.10. Блок-схема программы

Пример представления кода.

Программа написана на языке программирования Си в среде «Atmel Studio 6.0»

```
#include "avr_compiler.h"
#include "event_system_driver.h"
#include "asf.h"

int main( void )
{
 board_init();
 sysclk_init();
 pmic_init();

	// Устанавливаем пин 5 порта E как входной.
	PORTE_PIN5CTRL = PORT_ISC_LEVEL_gc;
	PORTE_DIRCLR = 0x00;

	/* Задаем порт R как выходной. */
	PORTR_DIRSET = 0xFF;

	// Устанавливаем пин 5 порта E как событие канала 0 входа мультиплексора.
	EVSYS_SetEventSource( 0, EVSYS_CHMUX_PORTE_PIN5_gc );

	// Задаём событие канала 0 как источник событий TCC0 и захвата входа, как действие
по событию.
	TCC0_CTRLD = (uint8_t) TC_EVSEL_CH0_gc | TC_EVACT_CAPT_gc;

	/* Задаем период счетчика и указываем канал событий 0 как источник тактирования.*/
	TCC0_PER = 0xFFFF;
	TCC0_CTRLE = TC_CLKSEL_EVCH0_gc;

	while (1)
	{
		PORTR_OUT=TCC0.CNT;
		TCC0_CNT=0x00;


	}
}
```

Литература

1. *Евстифеев А.В.* Микроконтроллеры AVRсемейства Mega. Руководство пользователя. - М.: Издательский дом «Додэка-XXI», 2007.- 592 с.
2. *Ревич Ю.В.* Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера. – СПб.: БХВ-Петербург, 2008. – 384 с.
3. *Трамперт В.* AVR-RISC микроконтроллеры: Пер. с нем. – К.: “МК-Пресс”, 2006. – 464 с.
4. *Кривченко И., Ламберт Е.* Микроконтроллеры XMEGA — новые возможности проверенного решения. Части 1, 2 и 3: «Компоненты и технологии», 2008. №4, 5,6
5. Официальный сайт Atmel на английском языке http://www.atmel.com/
6. Сайт компании ЭФО о микроконтроллерах различных производителей http://www.mymcu.ru/
7. Руководство пользователя по AVR-микроконтроллерам XMEGA http://www.gaw.ru/html.cgi/txt/doc/micros/avr/arh_xmega/index.htm

Приложение. Арифметические и логические команды

Мнемонический код	Операнды	Описание	Операция
ADD	Rd, Rr	Сложение без учета переноса	Rd ← Rd + Rr
ADC	Rd, Rr	Сложение с учетом переноса	Rd ← Rd + Rr + C
ADIW	Rd, K	Сложение слова с константой	Rd ← Rd + 1:Rd + K
SUB	Rd, Rr	Вычитание без учета переноса	Rd ← Rd - Rr
SUBI	Rd, K	Вычитание константы	Rd ← Rd - K
SBC	Rd, Rr	Вычитание с учетом переноса	Rd ← Rd - Rr - C
SBCI	Rd, K	Вычитание константы с учетом переноса	Rd ← Rd - K - C
SBIW	Rd, K	Вычитание константы из слова	Rd + 1:Rd ← Rd + 1:Rd - K
AND	Rd, Rr	Логическое И	Rd ← Rd · Rr
ANDI	Rd, K	Логическое И с константой	Rd ← Rd · K
OR	Rd, Rr	Логическое ИЛИ	Rd ← Rd v Rr
ORI	Rd, K	Логическое ИЛИ с константой	Rd ← Rd v K
EOR	Rd, Rr	Исключающее ИЛИ	Rd ← Rd ⊕ Rr
COM	Rd	Инвертирование всех бит	Rd ← $FF - Rd
NEG	Rd	Вычисление двоичного дополнения	Rd ← $00 - Rd
SBR	Rd,K	Установка бит в регистре	Rd ← Rd v K
CBR	Rd,K	Сброс бит в регистре	Rd ← Rd · ($FFh - K)
INC	Rd	Инкрементирование	Rd ← Rd + 1
DEC	Rd	Декрементирование	Rd ← Rd - 1
TST	Rd	Проверка на ноль или минус	Rd ← Rd · Rd
CLR	Rd	Сброс регистра	Rd ← Rd ⊕ Rd
SER	Rd	Установка регистра	Rd ← $FF

MUL	Rd,Rr	Умножение беззнаковых переменных	R1:R0 ← Rd x Rr (UU)
MULS	Rd,Rr	Умножение знаковых переменных	R1:R0 ← Rd x Rr (SS)
MULSU	Rd,Rr	Умножение знаковой и беззнаковой переменных	R1:R0 ← Rd x Rr (SU)
FMUL2	Rd,Rr	Дробное умножение беззнаковых переменных	R1:R0 ← Rd x Rr<<1 (UU)
FMULS	Rd,Rr	Дробное умножение знаковых переменных	R1:R0 ← Rd x Rr<<1 (SS)
FMULSU	Rd,Rr	Дробное умножение знаковой и беззнаковой переменных	R1:R0 ← Rd x Rr<<<1 (SU)
DES	K	Шифрование данных	if (H = 0) then R15:R0 ← Encrypt(R15:R0, K) else if (H = 1) then R15:R0 ← Decrypt(R15:R0, K)

Команды переходов

Мнемонический код	**Операнды**	**Описание**	**Операция**
RJMP	k	Относительный переход	PC ← PC + k + 1
IJMP		Косвенный переход по адресу в регистре (Z)	PC(15:0) ← Z PC(21:16) ← 0
EIJMP		Расширенный косвенный переход по адресу в регистре (Z)	PC(15:0) ← Z PC(21:16) ← EIND
JMP	k	Переход	PC ← k
RCALL	k	Относительный вызов подпрограммы	PC ← PC + k + 1
ICALL		Косвенный вызов по адресу в регистре (Z)	PC(15:0) ← Z PC(21:16) ← 0
EICALL		Расширенный косвенный вызов по адресу в регистре (Z)	PC(15:0) ← Z PC(21:16) ← EIND
CALL	k	Вызов подпрограммы	PC ← k
RET		Выход из подпрограммы	PC ← STACK
RETI		Выход из процедуры обработки прерывания	PC ← STACK
CPSE	Rd,Rr	Сравнение и пропуск инструкции, если равно	if (Rd = Rr) PC ← PC + 2 or 3
CP	Rd,Rr	Сравнение	Rd - Rr
CPC	Rd,Rr	Сравнение с учетом переноса	Rd -Rr -C
CPI	Rd,K	Сравнение с константой	Rd - K
SBRC	Rr, b	Пропуск инструкции, если бит в регистре равен 0	if (Rr(b) = 0) PC ← PC + 2 or 3
SBRS	Rr, b	Пропуск инструкции, если бит в регистре равен 1	if (Rr(b) = 1) PC ← PC + 2 or 3
SBIC	A, b	Пропуск инструкции, если бит в регистре B/B равен 0	if (I/O(A,b) = 0) PC ← PC + 2 or 3
SBIS	A, b	Пропуск инструкции, если бит в регистре B/B равен 1	If (I/O(A,b) =1) PC ← PC + 2 or 3
BRBS	s, k	Переход, если флаг статуса	if (SREG(s) = 1) then PC ←

		равен 1	PC + k + 1
BRBC	s, k	Переход, если флаг статуса равен 0	if (SREG(s) = 0) then PC ← PC + k + 1
BREQ	k	Переход, если равно	if (Z = 1) then PC ← PC + k + 1
BRNE	k	Переход, если неравно	if (Z = 0) then PC ← PC + k + 1
BRCS	k	Переход, если перенос равен 1	if (C = 1) then PC ← PC + k + 1
BRCC	k	Переход, если перенос равен 0	if (C = 0) then PC ← PC + k + 1
BRSH	k	Переход, если больше или равно	if (C = 0) then PC ← PC + k + 1
BRLO	k	Переход, если меньше	if (C = 1) then PC ← PC + k + 1
BRMI	k	Переход, если минус	if (N = 1) then PC ← PC + k + 1
BRPL	k	Переход, если плюс	if (N = 0) then PC ← PC + k + 1
BRGE	k	Переход, если больше или равно (с учетом знака)	if (N ⊕ V= 0) then PC ← PC + k + 1
BRLT	k	Переход, если меньше (с учетом знака)	if (N ⊕ V= 1) then PC ← PC + k + 1
BRHS	k	Переход, если флаг полупереноса равен 1	if (H = 1) then PC ← PC + k + 1
BRHC	k	Переход, если флаг полупереноса равен 0	if (H = 0) then PC ← PC + k + 1
BRTS	k	Переход, если флаг T равен 1	if (T = 1) then PC ← PC + k + 1
BRTC	k	Переход, если флаг T равен 0	if (T = 0) then PC ← PC + k + 1
BRVS	k	Переход, если флаг переполнения равен 1	if (V = 1) then PC ← PC + k + 1
BRVC	k	Переход, если флаг переполнения равен 0	if (V = 0) then PC ← PC + k + 1
BRIE	k	Переход, если прерывания разрешены	if (I = 1) then PC ← PC + k + 1
BRID	k	Переход, если прерывания отключены	if (I = 0) then PC ← PC + k + 1

Команды передачи данных

Мнемонический код	**Операнды**	**Описание**	**Операция**
MOV	Rd, Rr	Копирование регистра	Rd ← Rr
MOVW	Rd, Rr	Копирование пары регистров	Rd+1:Rd ← Rr+1:Rr
LDI	Rd, K	Загрузка константы в регистр	Rd ← K
LDS	Rd, k	Загрузка регистра по прямому адресу памяти данных	Rd ← (k)
LD	Rd, X	Косвенная загрузка	Rd ← (X)
LD	Rd, X+	Косвенная загрузка с последующим инкрементированием	Rd ← (X) X ← X + 1

LD	Rd, -X	Косвенная загрузка с предварительным декрементированием	X ← X - 1 ← X – 1 Rd ← (X) ← (X)
LD	Rd, Y	Косвенная загрузка	Rd ← (Y) ← (Y)
LD	Rd, Y+	Косвенная загрузка с последующим инкрементированием	Rd ← (Y) Y ← Y + 1
LD	Rd, -Y	Косвенная загрузка с предварительным декрементированием	Y ← Y - 1 ← Y - 1 Rd ← (Y) ← (Y)
LDD	Rd, Y+q	Косвенная загрузка со смещением	Rd ← (Y + q)
LD	Rd, Z	Косвенная загрузка	Rd ← (Z)
LD	Rd, Z+	Косвенная загрузка с последующим инкрементированием	Rd ← (Z) Z ← Z + 1
LD	Rd, -Z	Косвенная загрузка с предварительным декрементированием	Z <-- Z - 1 <-- Z - 1 Rd ← (Z) ← (Z)
LDD	Rd, Z+q	Косвенная загрузка со смещением	Rd ← (Z + q)
STS	k, Rr	Запись по прямому адресу памяти данных	(k) ← Rd
ST	X, Rr	Косвенная запись	(X) ← Rr
ST	X+, Rr	Косвенная запись с последующим инкрементированием	(X) ← Rr X ← X + 1
ST	-X, Rr	Косвенная запись с предварительным декрементированием	X ← X – 1 (X) ← Rr
ST	Y, Rr	Косвенная запись	(Y) ← Rr
ST	Y+, Rr	Косвенная запись с последующим инкрементированием	(Y) ← Rr Y ← Y + 1
ST	-Y, Rr	Косвенная запись с предварительным декрементированием	Y ← Y - 1 (Y) ← Rr
STD	Y+q, Rr	Косвенная запись со смещением	(Y + q) ← Rr
ST	Z, Rr	Косвенная запись	(Z) ← Rr
ST	Z+, Rr	Косвенная запись с последующим инкрементированием	(Z) ← Rr Z ← Z + 1
ST	-Z, Rr	Косвенная запись с предварительным декрементированием	Z ← Z – 1 (Z) ← Rr
STD	Z+q,Rr	Косвенная запись со смещением	(Z + q) ← Rr
LPM		Чтение памяти программ	R0 ← (Z)

LPM	Rd, Z	Чтение памяти программ	Rd ← (Z)
LPM	Rd, Z+	Чтение памяти программ с последующим инкрементированием	Rd ← (Z) Z ← Z + 1
ELPM		Расширенное чтение памяти программ	R0 ← (RAMPZ:Z)
ELPM	Rd, Z	Расширенное чтение памяти программ	Rd ← (RAMPZ:Z)
ELPM	Rd, Z+	Расширенное чтение памяти программ с последующим инкрементированием	Rd ← (RAMPZ:Z) Z ← Z + 1
SPM		Запись в память программ	(RAMPZ:Z) <-- R1:R0
SPM	Z+	Запись в память программ с последующим инкрементированием на 2	(RAMPZ:Z) ← R1:R0 Z ← Z + 2
IN	Rd, A	Чтение из памяти ввода-вывода	Rd ← I/O(A)
OUT	A, Rr	Запись в память ввода-вывода	I/O(A) ← Rr
PUSH	Rr	Поместить содержимое регистра в стек	STACK ← Rr
POP	Rd	Извлечь содержимое регистра из стека	Rd ← STACK

Команды тестирования и обработки бит

Мнемонический код	**Операнды**	**Описание**	**Операция**
LSL	Rd	Логический сдвиг влево	Rd(n+1) ← Rd(n) Rd(0) ← 0 C ← Rd(7)
LSR	Rd	Логический сдвиг вправо	Rd(n) ← Rd(n+1) Rd(7) ← 0 C ← Rd(0)
ROL	Rd	Циклический сдвиг влево через перенос	Rd(0) ← C Rd(n+1) ← Rd(n) C ← Rd(7)
ROR	Rd	Циклический сдвиг вправо через перенос	Rd(7) ← C Rd(n) ← Rd(n+1) C ← Rd(0)
ASR	Rd	Арифметический сдвиг вправо	Rd(n) ← Rd(n+1), n=0..6
SWAP	Rd	Обмен тетрадами	Rd(3..0) ↔ Rd(7..4)
BSET	s	Установка флага	SREG(s) ← 1
BCLR	s	Сброс флага	SREG(s) ← 0
SBI	A, b	Установка бита в регистре B/B	I/O(A, b) ← 1
CBI	A, b	Сброс бита в регистре B/B	I/O(A, b) ← 0
BST	Rr, b	Запись бита регистра в бит T	T ← Rr(b)
BLD	Rd, b	Чтение бита T в бит регистра	Rd(b) ← T

SEC		Установка флага переноса	C ← 1
CLC		Сброс флага переноса	C ← 0
SEN		Установка флаг отрицательности	N ← 1
CLN		Сброс флаг отрицательности	N ← 0
SEZ		Установка флага нуля	Z ← 1
CLZ		Сброс флага нуля	Z ← 0
SEI		Общее разрешение прерываний	I ← 1
CLI		Общий запрет прерываний	I ← 0
SES		Установка флага проверки на знак	S ← 1
CLS		Сброс флага проверки на знак	S <-- 0
SEV		Установка флага переполнения двоичного дополнения	V ← 1
CLV		Сброс флага переполнения двоичного дополнения	V ← 0
SET		Установка флага T в SREG	T ? 1
CLT		Сброс флага T в SREG	T ? 0
SEH		Установка флага полупереноса в SREG	H ? 1
CLH		Сброс флага полупереноса в SREG	H ← 0

Команды управления микроконтроллером

Мнемонический код	**Операнды**	**Описание**	**Операция**
BREAK		Остановка	
NOP		Нет операции	
SLEEP		Переход в экономичный режим работы	
WDR		Сброс сторожевого таймера	