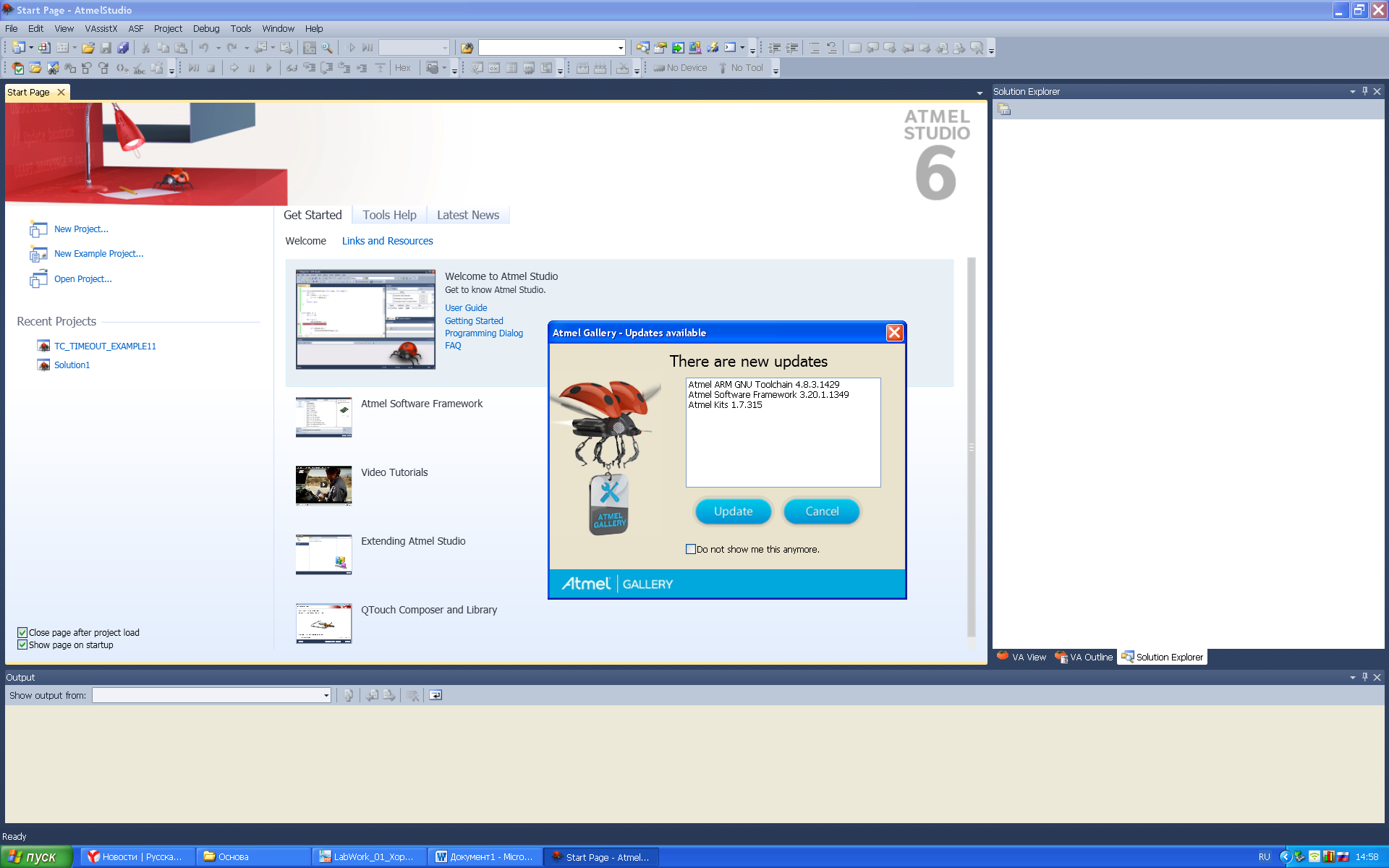
**Лабораторная работа №7.**

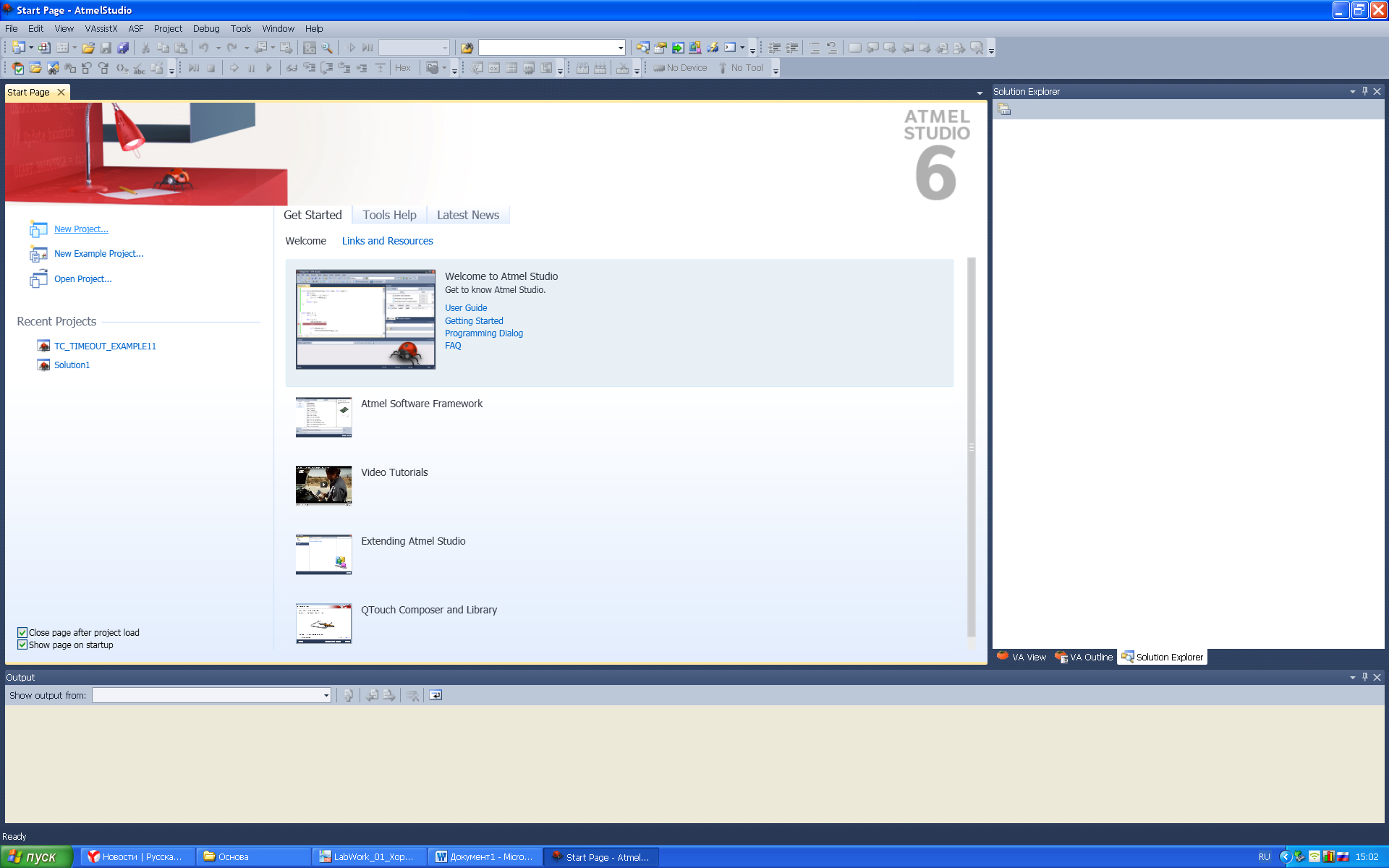
**Единая интегрированная среда разработки для микроконтроллеров ARM и AVR, позволяющая значительно упростить процесс разработки и отладки приложений на микроконтроллерах.**

**1. Создание проекта в AtmelStudio.**

*Запуск программы.* Запуск программы осуществляется с помощью иконки **AtmelStudio 6.2** на рабочем столе или через меню. После запуска появится окно, приведенное на рисунке 1.



*Рисунок 1 - Окно программы* *AtmelStudio 6.2 .*

При предложении апгрейда, нажмите кнопку отмены и приступайте к созданию нового проекта, нажатием на кнопку «Новый проект»  .

В появившимся окне выбираем язык программирования Assembler и сохраняем проект с именем, соответствующей фамилии студента, выполняющего работу, например, «Ivanov\_Application1».

В следующем окне выбираем, с каким типом микроконтроллера будет взаимодействовать созданная программа.

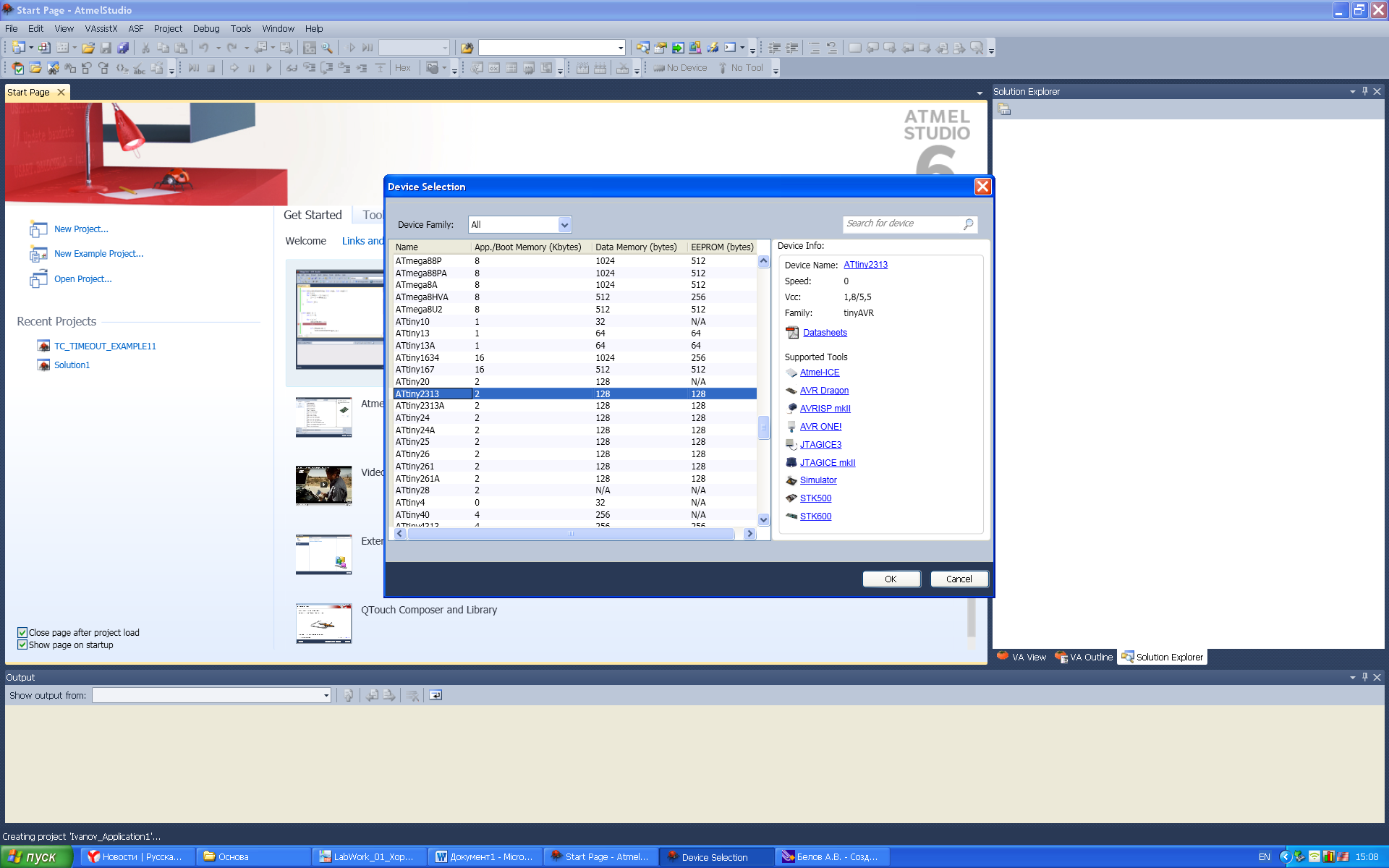


Рисунок 2 – Окно выбора микроконтроллера

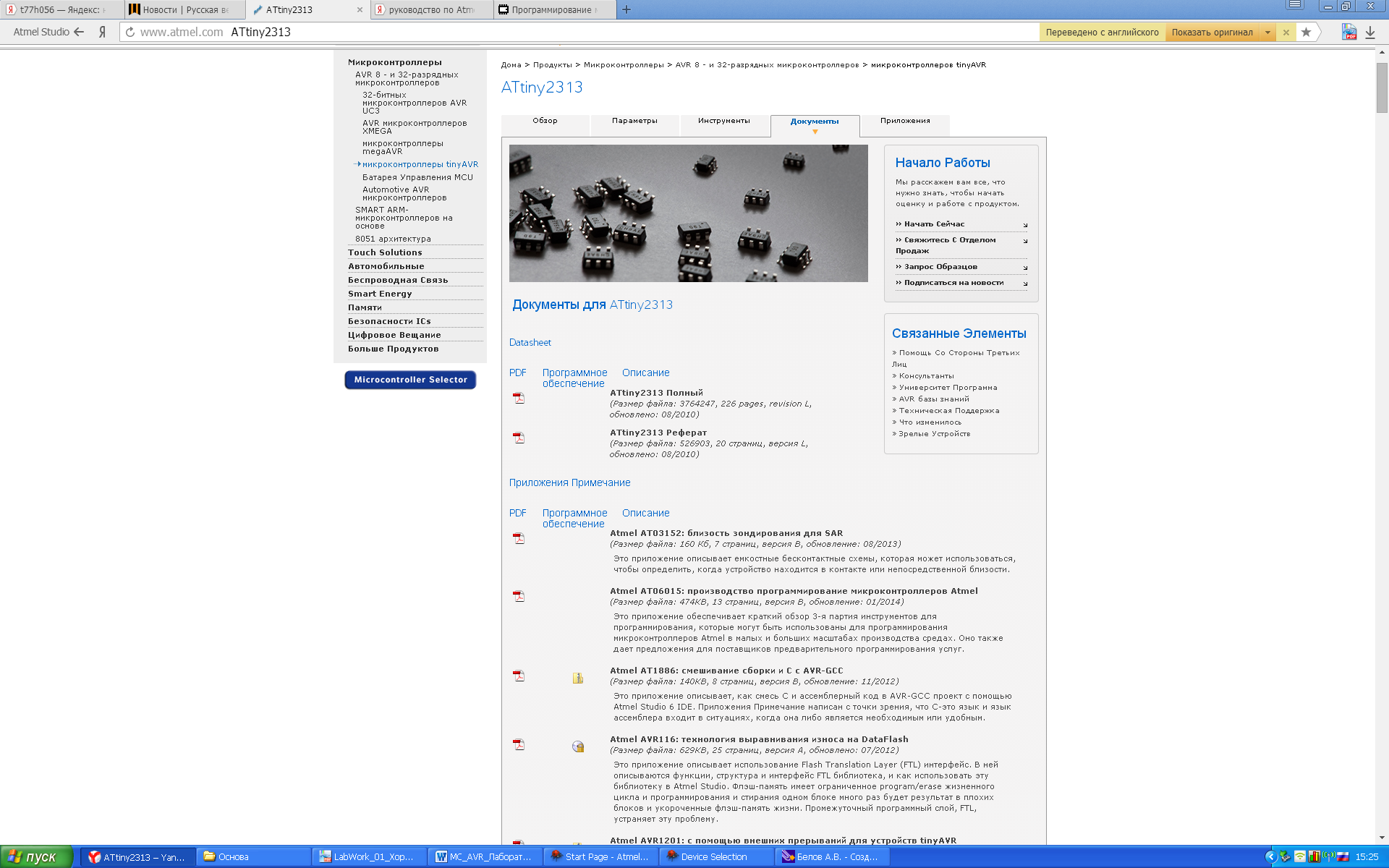


Рисунок 3 – Сайт производителя микроконтроллеров Atmel

Выберите микроконтроллер ATiny2313, как показано на рисунке 2. Также в данном окне разработчику проекта будет предложен перечень технических устройств по отладке и программированию, а также подробное описание данного микроконтроллера – Datasheets. При нажатии на «Datasheets» для пользователя будет открыт сетевой ресурс на портале производителя микроконтроллеров *http://www.atmel.com/devices/ATTINY2313.aspx?tab =documents*, как показано на рисунке 3.

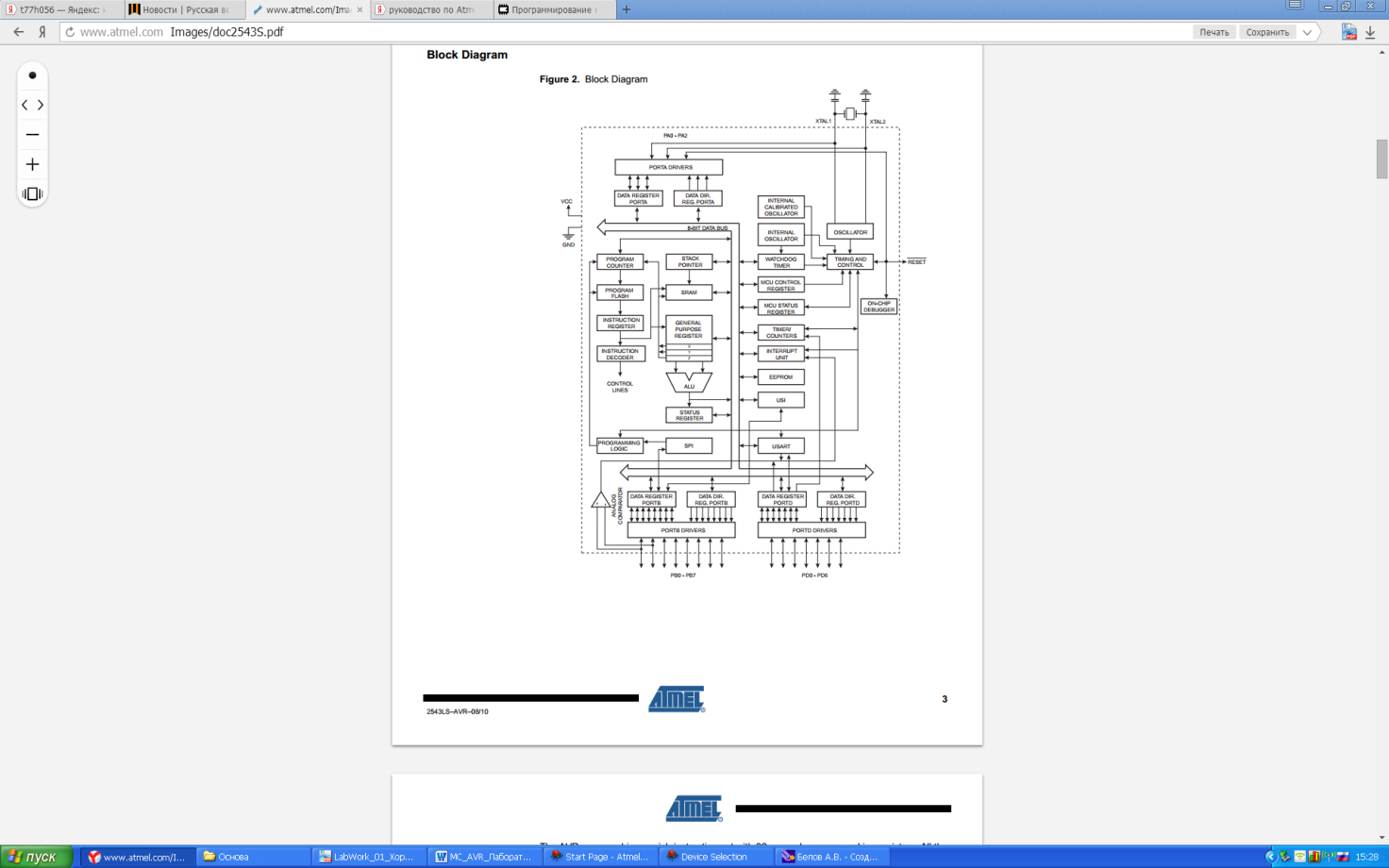
В описании приводится вся подробная информация о данном МК, начиная от внешнего вида и заканчивая адресацией конкретных регистров и системой команд. Структурная схема процессора ATiny2313, полученная с помощью Datasheets приведена на рисунке 4.

Рисунок 4 - Структурная схема процессора ATiny2313.

Эта микросхема, хотя и относится к семейству «Tiny», на самом деле занимает некое промежуточное место между семейством «Tiny» и семейством «Mega». Она не так перегружена внутренней периферией и не столь сложна, как микросхемы семейства «Mega». Но и не настолько примитивна, как все остальные контроллеры семейства «Tiny».

Эта микросхема содержит два основных и один дополнительный порт ввода—вывода, имеет не только восьмиразрядный, но и шестнадцатиразрядный таймер/счетчик. Имеет оптимальные размеры (20-выводной корпус, рисунок 5). К тому же эта микросхема имеет и еще одну привлекательную особенность.

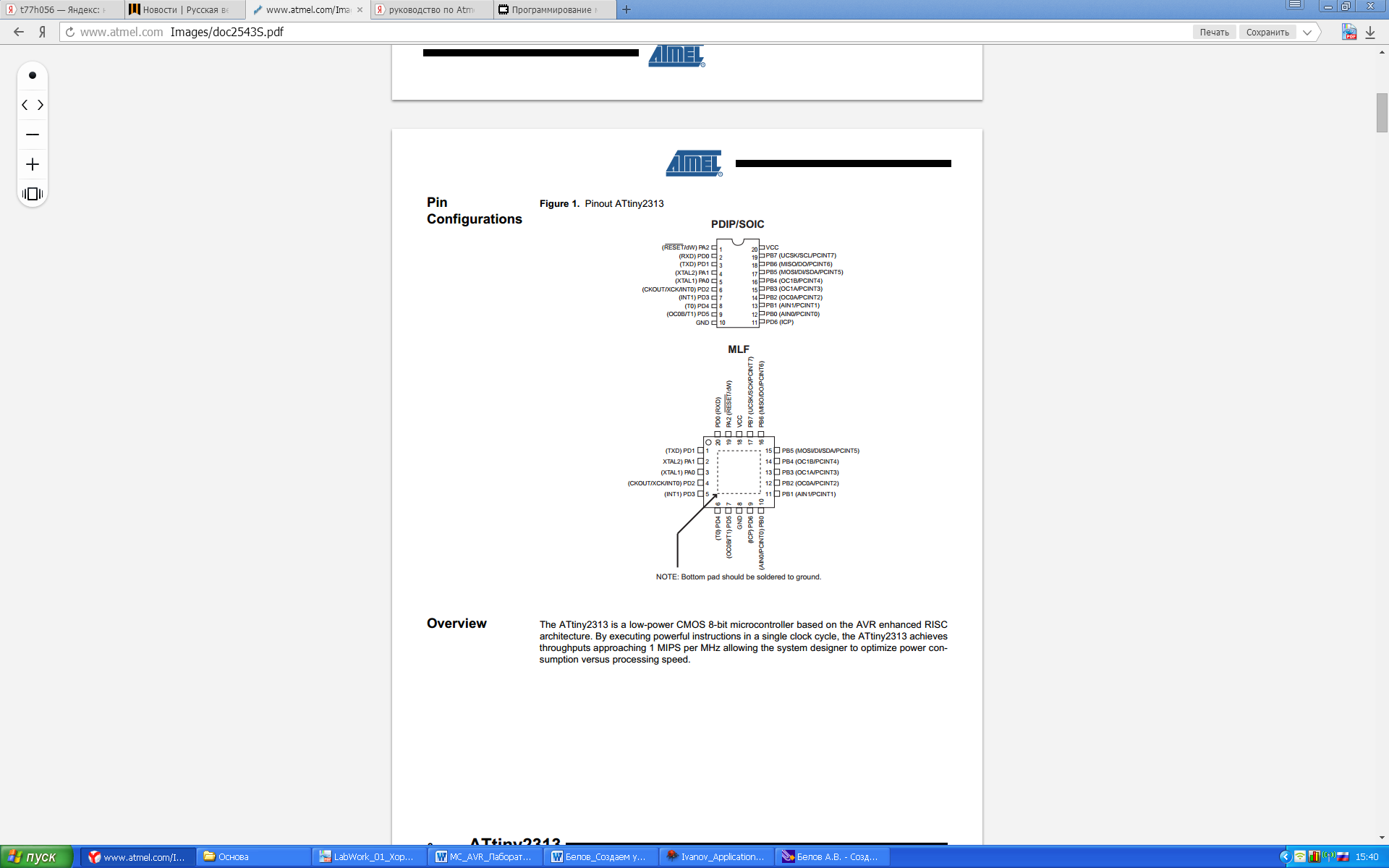


Рисунок 5 - Разновидности корпусов МК ATiny2313

**Этап №2. Создание программы.**

Самая простая задача, которую можно придумать для микроконтроллера, может звучать следующим образом:

«Разработать устройство управления одним светодиодным индикатором при помощи одной кнопки. При нажатии кнопки светодиод должен зажечься, при отпускании — погаснуть».

С практической точки зрения это совершенно бессмысленная задача, так как для ее решения проще обойтись без микропроцессора. Но будет использована в качестве примера для обучения.

Для подключения к микроконтроллеру AVR любых внешних устройств используются порты ввода—вывода. Причем каждый такой порт способен работать либо на ввод, либо и на вывод.

Удобнее всего светодиод подключить к одному из портов, а кнопку — к другому. В этом случае управляющая программа должна будет настроить порт, к которому подключен светодиод, на вывод, а порт, к которому подключена кнопка, на ввод. Других специальных требований к микроконтроллеру не имеется.

Микроконтроллер имеет два основных порта ввода—вывода (порт В и порт D). Договоримся, что для управления светодиодом мы будем использовать младший разряд порта В (линия РВ.0), а для считывания информации с кнопки управления используем младший разряд порта D (линия PD.0). Полная схема устройства, позволяющего решить поставленную выше задачу, приведена на рисунке 6.

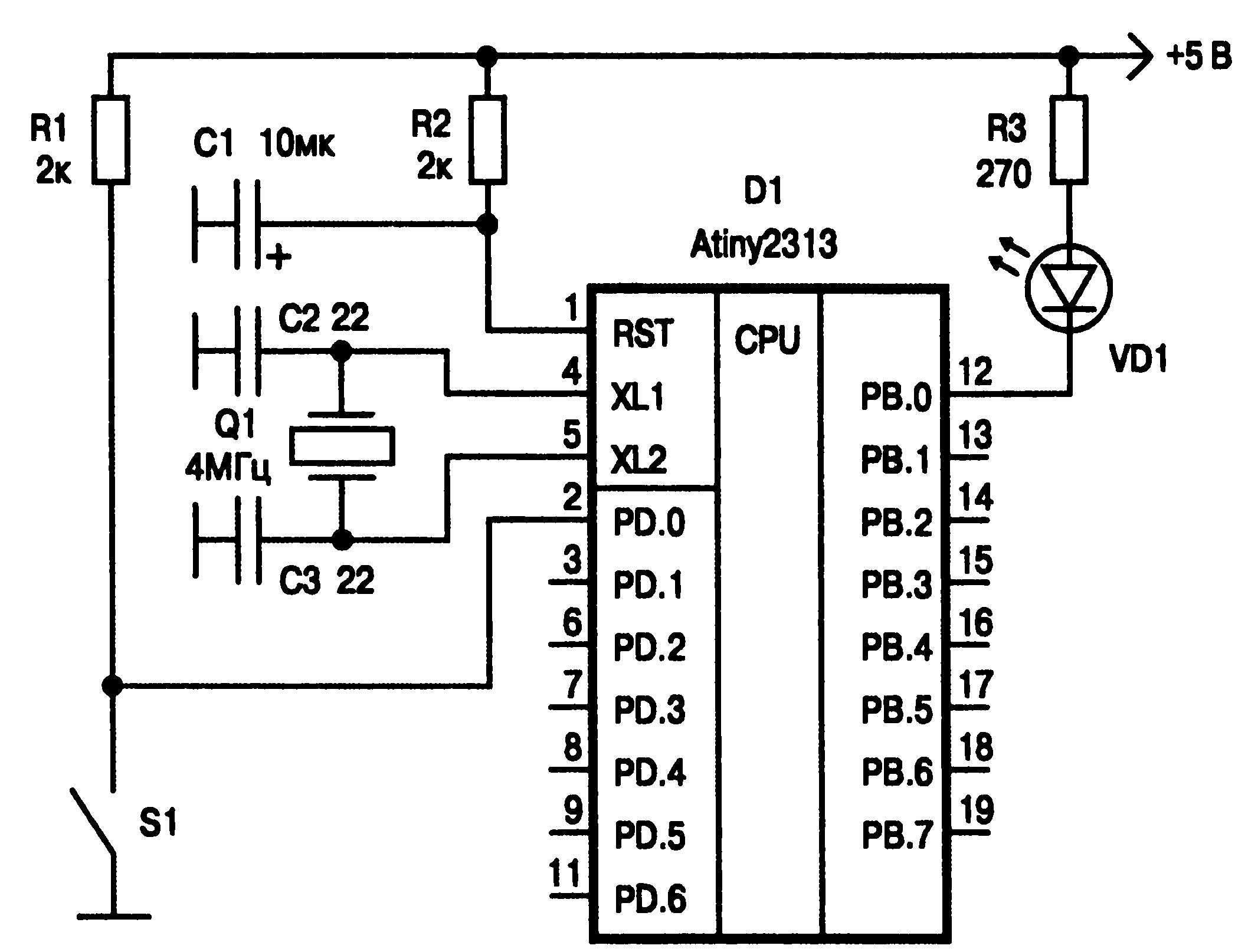


Рисунок 6 - Принципиальная схема МК с одним светодиодом и одной кнопкой.

Для подключения кнопки S1 использована классическая схема. В исходном состоянии контакты кнопки разомкнуты. Через резистор R1 на вход PD.0 микроконтроллера подается «плюс» напряжения питания, что соответствует сигналу логической единицы.

При замыкании кнопки напряжение падает до нуля, что соответствует логическому нулю. Таким образом, считывая значение сигнала на соответствующем выводе порта, программа может определять момент нажатия кнопки. Несмотря на простоту данной схемы, микроконтроллер AVR позволяет ее упростить. Резистор R1 можно исключить, заменив его внутренним нагрузочным резистором микроконтроллера. Как уже говорилось выше, микроконтроллеры серии AVR имеют встроенные нагрузочные резисторы для каждого разряда порта. Главное при написании программы — не забыть включить программным путем соответствующий резистор.

Подключение светодиода также выполнено по классической схеме. Это непосредственное подключение к выходу порта. Каждый выход микроконтроллера рассчитан на непосредственное управление светодиодом среднего размера с током потребления до 20 мА. В цепь светодиода включен токоограничивающий резистор R3.

Для того, чтобы зажечь светодиод, микроконтроллер должен подать на вывод РВ.0 сигнал логического нуля. В этом случае напряжение, приложенное к цепочке R2, VD1, окажется равным напряжению питания, что вызовет ток через светодиод, и он загорится. Если же на вывод PD.0 подать сигнал логической единицы, падение напряжения на светодиоде и резисторе окажется равным нулю, и светодиод погаснет.

Кроме цепи подключения кнопки и цепи управления светодиодом, на схеме вы можете видеть еще несколько цепей. Это стандартные цепи, обеспечивающие нормальную работу микроконтроллера. Кварцевый резонатор Q1 обеспечивает работу встроенного тактового генератора. Конденсаторы С2 и СЗ — это цепи согласования кварцевого резонатора.

Элементы Cl, R2 — это стандартная цепь начального сброса. Такая цепь обеспечивает сброс микроконтроллера в момент включения питания. Еще недавно подобная цепь была обязательным атрибутом любой микропроцессорной системы. Однако технология производства микроконтроллеров достигла такого уровня, что обе эти цепи (внешний кварц и цепь начального сброса) теперь можно исключить.

Большинство микроконтроллеров AVR, кроме тактового генератора с внешним кварцевым резонатором, содержат внутренний RC-генератор, не требующий никаких внешних цепей. Если вы не предъявляете высоких требований к точности и стабильности частоты задающего генератора, то микросхему можно перевести в режим внутреннего RC-генератора и отказаться как от внешнего кварца (Q1), так и от согласующих конденсаторов (С2 и СЗ).

Цепь начального сброса тоже можно исключить. Любой микроконтроллер AVR имеет внутреннюю систему сброса, которая в большинстве случаев прекрасно обеспечивает стабильный сброс при включении питания. Внешние цепи сброса применяются только при наличии особых требований к длительности импульса сброса. А это бывает лишь в тех случаях, когда микроконтроллер работает в условиях больших помех и нестабильного питания. Все описанные выше переключения производятся при помощи соответствующих fuse-переключателей.

Таким образом, схема может быть упрощена, как показано на рисунке 7.

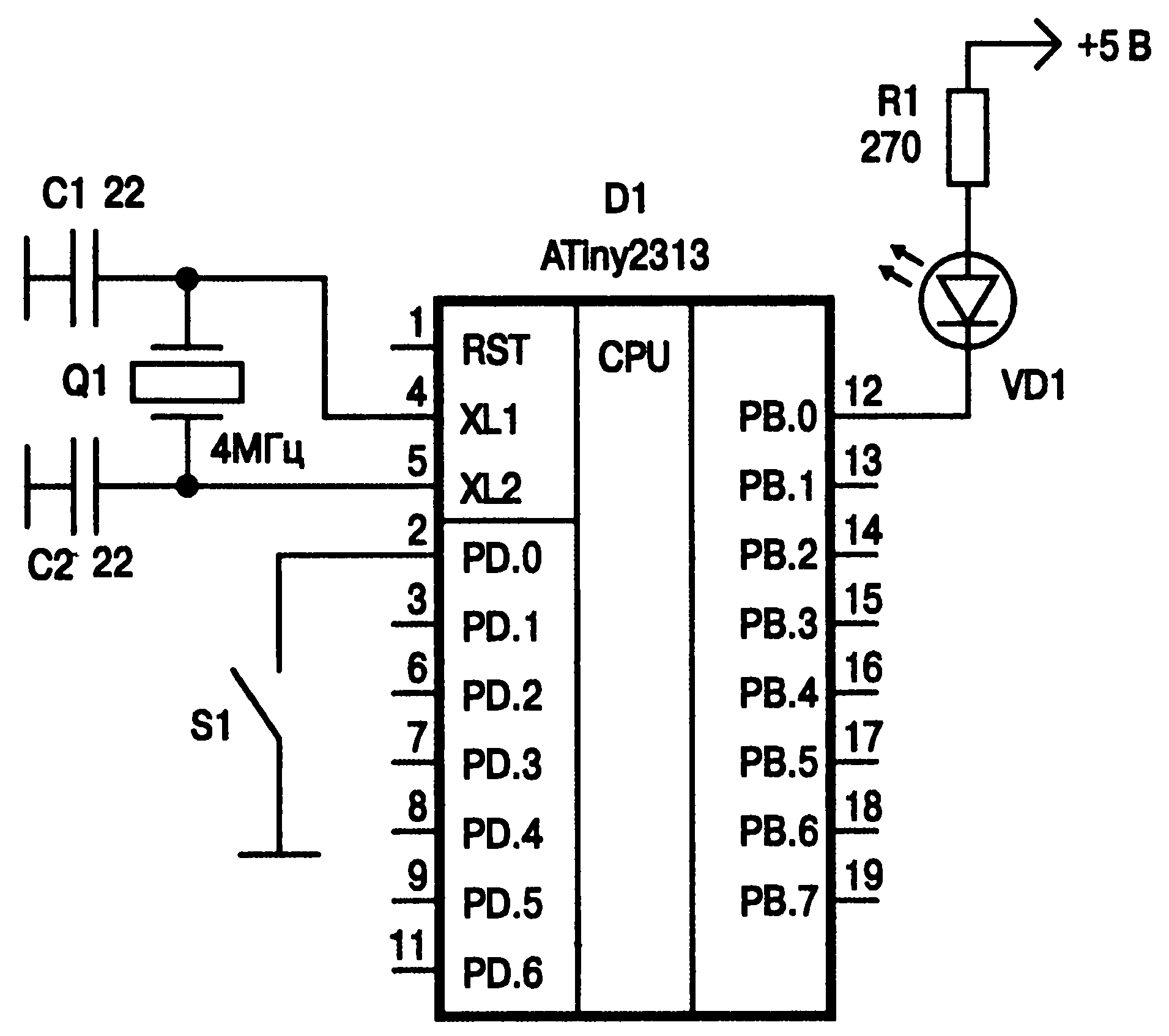


Рисунок 7 – Упрощенная схема включения светодиода

После этого можно приступать к разработке программы. Разработка любой программы начинается с разработки алгоритма.

Алгоритм — это последовательность действий, которую должен произвести микроконтроллер, чтобы достичь требуемого результата. Для простых задач алгоритм можно просто описать словами. Для более сложных задач алгоритм рисуется в графическом виде.

Словесный алгоритм таков: После операций начальной настройки портов микроконтроллер должен войти в непрерывный цикл, в процессе которого он должен опрашивать вход, подключенный к кнопке, и в зависимости от ее состояния управлять светодиодом. Опишем это подробнее.

Операции начальной настройки:

* установить начальное значение для вершины стека микроконтроллера;
* настроить порт В на вывод информации;
* подать на выход РВ.0 сигнал логической единицы (потушить светодиод);
* сконфигурировать порт D на ввод;
* включить внутренние нагрузочные резисторы порта D.

Операции, составляющее тело цикла:

* прочитать состояние младшего разряда порта PD (PD.0);
* если значение этого разряда равно единице, выключить светодиод;
* если значение разряда PD.0 равно нулю, включить светодиод;
* перейти на начало цикла.

Текст возможного варианта программы на ассемблере, реализующий поставленную выше задачу, приведен в листинге 1.

Листинг 1.

; Пример 1

; Программа управления светодиодом

; Команды управления

1 .include "tn2313def.inc" ; Присоединение файла описаний

2 .list ; Включение листинга

3 .def temp = R16 ; Определение главного рабочего регистра

; Начало программного кода

4 .cseg ; Выбор сегмента программного кода

5 .org 0 ; Установка текущего адреса на ноль

; Инициализация стека

6 ldi temp, RAMEND ; Выбор адреса вершины стека

7 out SPL, temp ; Запись его в регистр стека

;Инициализация портов ВВ

8 ldi temp, 0 ; Записываем 0 в регистр temp

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

10 ldi temp, 0xFF ;Записываем число $FF в регистр temp

11 out DDRB, temp ;Записываем temp в DDRB (порт PB на вывод)

12 out PORTB, temp ;Записываем temp в PORTB (потушить светодиод)

13 out PORTD, temp ;Записываем temp в PORTD (включ. внутр. резист.)

; Инициализация компаратора

14 ldi temp, 0x80 ; Выключение компаратора

15 out ACSR, temp

;Основной цикл

16 main: in temp, PIND ; Читаем содержимое порта PD

17 out PORTB, temp ; Пересылаем в порт РВ

18 rjmp main ; К началу цикла

Программа на Ассемблере представляет собой набор команд и комментариев (иногда команды называют инструкциями). Каждая команда занимает одну отдельную строку. Их допускается перемежать пустыми строками. Команда обязательно содержит оператор, который выглядит как имя выполняемой операции.

Некоторые команды состоят только из одного оператора. Другие же команды имеют один или два операнда (параметра). Операнды записываются в той же строке сразу после оператора, через пробел. Если операндов два, их записывают через запятую. Так, в строке 6 программы записана команда загрузки константы в регистр общего назначения. Она состоит из оператора ldi и двух операндов temp и RAMEND.

В случае необходимости перед командой допускается ставить так называемую метку. Она состоит из имени метки, заканчивающимся двоеточием. Метка служит для именования данной строки программы.

Затем это имя используется в различных командах для обращения к помеченной строке.

При выборе имени метки необходимо соблюдать следующие правила:

* имя должно состоять из одного слова, содержащего только латинские буквы и цифры;
* допускается также применять символ подчеркивания;
* первым символом метки обязательно должна быть буква или символ подчеркивания.

Строка 16 программы содержит метку с именем main. Метка не обязательно должна стоять в строке с оператором. Допускается ставить метку в любой строке программы. Кроме команд и меток, программа содержит комментарии.

Комментарий — это специальная запись в тело программы, предназначенная для человека. Компьютер в процессе трансляции программы игнорирует все комментарии. Комментарий может занимать отдельную строку, а может стоять в той же строке, что и команда. Начинается комментарий с символа «точка с запятой». Все, что находится после точки с запятой до конца текущей строки программы, считается комментарием.

Если в уже готовой программе вы поставите точку с запятой в начале строки перед какой-либо командой, то данная строка для транслятора как бы исчезнет. С этого момента транслятор будет считать всю эту строку комментарием. Таким образом, можно временно отключать отдельные строки программы в процессе отладки (то есть при поиске ошибок в программе).

Кроме операторов, в языке Ассемблер применяются псевдооператоры или директивы. Если оператор — это некий эквивалент реальной команды микроконтроллера и в процессе трансляции заменяется соответствующим машинным кодом, который помещается в файл результата трансляции, то директива, хотя по форме и напоминает оператор, но не является командой процессора.

Определение. Директивы — это специальные вспомогательные команды для транслятора, определяющие режимы трансляции и реализующие различные вспомогательные функции.

В данной конкретной версии Ассемблера директивы выделяются особым образом. Имя каждой директивы начинается с точки (см. листинг 1, строки с 1 по 5).

Проведем описания входящих листинг команд.

Директивы

.include

Присоединение к текущему тексту программы другого программного текста. Подобный прием используется практически во всех существующих языках программирования. При составлении программ часто бывает как, что в совершенно разных программах приходится применять абсолютно одинаковые программные фрагменты. Для того, чтобы не переписывать эти фрагменты из программы в программу, их принято оформлять в виде отдельного файла с таким расчетом, чтобы этот файл могли использовать все программы, где этот фрагмент потребуется.

В языке Ассемблер для присоединения фрагмента к программе используется псевдооператор include. В качестве параметра для этой директивы должно быть указано имя присоединяемого файла. Если такой оператор поставить в любом месте программы, то содержащийся в присоединяемом файле фрагмент в процессе трансляции как бы вставляется в то самое место, где находится оператор. Например, в программе на листинге 1.1 в строке 1 в основной текст программы вставляется текст из файла tn2313def.inc.

Файл tn2313def.inc — это файл описаний. Он содержит описание всех регистров и некоторых других параметров микроконтроллера ATiny2313. Это описание понадобится для того, чтобы в программе можно было обращаться к каждому регистру по его имени.

.list

Включение генерации листинга. В данном случае листинг—это специальный файл, в котором отражается весь ход трансляции программы. Такой листинг повторяет весь текст вашей программы, включая все присоединенные фрагменты. Против каждой строки программы, содержащей реальную команду, помещаются соответствующие ей машинные коды. Там же показываются все найденные в процессе трансляции ошибки. По умолчанию листинг не формируется. Если вам нужен листинг, включите данную команду в вашу программу.

.def

Макроопределение. Эта команда позволяет присваивать различным регистрам микроконтроллера любые осмысленные имена, упрощающие чтение и понимание текста программы. В создаваемой программе понадобится один регистр для временного хранения различных величин. Выберем для этой цели регистр r16 и присвоим ему наименование temp от английского слова temporary — временный.

Данная команда выполняется в строке 3 (см. листинг 1). Теперь в любом месте программы вместо имени г16 можно применять имя temp. Присвоение осмысленного имени очень облегчает программирование. Именно таким образом определены имена всех стандартных регистров в файле tn2313def. inc.

.cseg

Псевдооператор выбора программного сегмента памяти. Микроконтроллер для хранения данных имеет три вида памяти: память программ (Flash), оперативную память (SRAM) и энергонезависимую память данных (EEPROM). Программа на Ассемблере должна работать с любым из этих трех видов памяти. Для этого в Ассемблере существует понятие «сегмент памяти». Существуют директивы, объявляющие каждый такой сегмент:

* сегмент кода (памяти программ) cseg;
* сегмент данных (ОЗУ) dseg;
* сегмент EEPROM eseg.

После объявления каждого такого сегмента он становится текущим. Это значит, что все последующие операторы относятся исключительно к объявленному сегменту. Объявленный сегмент будет оставаться текущим до тех пор, пока не будет объявлен какой-либо другой сегмент.

Только в сегменте кода Ассемблер описывает команды, которые затем в виде кодов будут записаны в память программ. В остальных двух сегментах используются директивы распределения памяти и директивы описания данных.

Так как команды в программной памяти должны располагаться по порядку, одна за другой, то их размещение удобно автоматизировать. Программист не указывает, по какому адресу в памяти должна быть расположена та либо иная команда. Программист просто последовательно пишет команды.

А уже транслятор автоматически размещает их в памяти. Для этого используется понятие «указатель текущего адреса». Указатель текущего адреса не имеет отношения к регистру адреса микроконтроллера и вообще физически не существует. Это просто понятие, используемое в языке Ассемблер. Указатель помогает транслятору разместить все команды программы по ячейкам памяти.

По умолчанию считается, что в начале программы значение текущего указателя рано нулю. Поэтому первая же команда программы будет размещена по нулевому адресу. По мере трансляции программы указатель смещается в сторону увеличения адреса. Если команда имеет длину в один байт, то после ее трансляции указатель смещается на одну ячейку. Если команда состоит из двух байтов, на две. Таким образом, размещаются все команды программы.

.org

Принудительное позиционирование указателя текущего адреса. Иногда необходимо разместить какой-либо фрагмент программы в программной памяти не сразу после предыдущего фрагмента, а в конкретном месте программной памяти. Например, начиная с какого-нибудь заранее определенного адреса. Для этого используют директиву org.

Она позволяет принудительно изменить значение указателя текущего адреса. Оператор org имеет всего один параметр — новое значение указателя адреса. К примеру, команда . org 0x10 установит указатель на адрес 0x10. Транслятор автоматически следит, чтобы при перемещении указателя ваши фрагменты программы не налезали друг на друга. В случае несоблюдения этого условия транслятор выдает сообщение об ошибке.

В создаваемой программе команда позиционирования указателя применяется всего один раз. В строке 5 указатель устанавливается на нулевой адрес. В данном случае директива org имеет чисто декларативное значение, так как в начале программы значение указателя и так равно нулю.

Операторы

ldi

Загрузка в РОН числовой константы. В строке 6 программы (Листинг 1) при помощи этой команды в регистр temp (г16) записывается числовая константа, равная максимальному адресу ОЗУ. Эта константа имеет имя RAMEND. Ее значение описано в файле tn2313def.inc. Для микроконтроллера ATiny2313 значение RAMEND равно $7F.

Как можно видеть из листинга 1, оператор ldi имеет два параметра:

* первый параметр — это имя РОН, куда помещается константа;
* второй параметр — значение этой константы.

Обратите внимание, что в команде сначала записывается приемник информации, затем ее источник. Такой же порядок будет в любой другой команде, имеющей два операнда. Это общее правило для языка Ассемблер.

out

Вывод содержимого РОН в регистр ввода—вывода. Команда также имеет два параметра:

* первый параметр — имя РВВ, являющегося приемником информации;
* второй параметр — имя РОН, являющегося источником.

В строке 7 программы содержимое регистра temp выводится в РВВ с именем SPL.

in

Ввод информации из регистра ввода—вывода. Имеет два параметра. Параметры те же, что и в предыдущем случае, но источник и приемник меняются местами. В строке 19 программы содержимое регистра PORTD помещается в регистр temp.

rjmp

Команда безусловного перехода. Команда имеет всего один параметр — адрес перехода. В строке 21 программы оператор безусловного перехода передает управление на строку, помеченную меткой main. То есть на строку 19. Данная строка демонстрирует использование метки.

На самом деле в качестве параметра оператора rjmp должен выступать так называемый относительный адрес перехода. То есть, число байт, на которое нужно сместиться вверх или вниз от текущего адреса. Направление смещения (вверх или вниз) — это знак числа. Он определяется старшим битом. Язык Ассемблера избавляет программиста от необходимости подсчета величины смещения. Достаточно в нужной строке программы поставить метку, а в качестве адреса перехода указать ее имя, и транслятор сам вычислит значение этого параметра.

При использовании команды rjmp существует одно ограничение. Соответствующая команда микроконтроллера кодируется при помощи одного шестнадцатиразрядного слова. Для указания величины смещения она использует всего двенадцать разрядов. Поэтому такая команда может вызвать переход в пределах ±2 Кбайт. Если расположить метку слишком далеко от оператора rjmp, то при трансляции программы это вызовет сообщение об ошибке.

*Описание программы*

Шапка программы представляет собой несколько строк комментариев и помогает отличать программы друг от друга. Кроме названия программы, в шапку можно поместить ее версию, а также дату написания.

Самая первая команда программы — это псевдокоманда include, которая присоединяет к основному тексту программы файл описаний (см. листинг 1 строка 1). В стандартном пакете AtmelStudio имеется целый набор подобных файлов описаний. Для каждого микроконтроллера серии AVR свой отдельный файл. Программисту нужно лишь выбрать нужный файл и включить подобную строку в свою программу. Учтите, что без присоединения файла описаний дальнейшая программа работать не будет.

Для микроконтроллера ATiny2313 файл описаний имеет название tn2313def. inc. В команде include достаточно лишь указать его полное имя (с расширением). Указывать полный путь необязательно.

Назначение команды .list (строка 2), уже не вызывает вопросов. Остановимся на команде макроопределения (строка 3). Уместно задаться вопросом: почему выбран именно r16, а, к примеру, не r0? Это становится понятно, если вспомнить, что регистры, начиная с r0 и заканчивая r15, имеют меньше возможностей. Например, в строке 14 программы регистр temp используется в команде ldi. Однако команда ldi не работает с регистрами r0-r15. Именно по этой причине и выбран r16.

Следующие две команды (строки 4, 5) подробно описаны ранее. Они служат для выбора программного сегмента памяти и установки начального значения указателя.

В строках 6 и 7 производится инициализация стека. В регистр стека SPL записывается адрес его вершины. В качестве адреса выбран самый верхний адрес ОЗУ. Для обозначения этого адреса в данной версии Ассемблера существует специальная константа с именем RAMEND. Значение этой константы определяется в файле описаний (в нашем случае в файле tn2313def. inc). Для микроконтроллера Atiny2313 константа RAMEND равна 0xDF.

Одной строкой записать константу в регистр стека невозможно, так как в системе команд микроконтроллеров AVR отсутствует подобная команда. Отсутствующую команду мы заменяем двумя другими. И тут нам пригодится регистр temp. Он послужит в данном случае передаточным звеном. Сначала константа RAMEND помещается в регистр temp (строка 6), а затем уже содержимое temp помещается в регистр SPL (строка 7).

В строках 8—12 производится настройка портов ввода—вывода.

Ранее было условлено, что порт PD будет работать на ввод, а порт РВ — на вывод. Для выбора нужного направления передачи информации запишем управляющие коды в соответствующие регистры DDRx. Во все разряды регистра DDRD запишем нули (настройка порта PD на ввод), а во все разряды регистра DDRB запишем единицы (настройка порта РВ на вывод). Кроме того, нам нужно включить внутренние нагрузочные резисторы порта PD. Для этого запишем единицы (то есть число 0xFF) во все разряды регистра PORTD. И, наконец, в момент старта программы желательно погасить светодиод. Для этого запишем единицы в разряды порта РВ.

Все описанные выше действия по настройке порта также выполняются с использованием промежуточного регистра temp. Сначала в него помещается ноль (строка 8). Ноль записывается только в регистр DDRD (строка 9). Затем в регистр temp помещается число 0xFF (строка 10). Это число по очереди записывается в регистры DDRB, PORTB, PORTD (строки 11,12,13).

Строки 14 и 15 включены в программу для перестраховки. Дело в том, что встроенный компаратор микроконтроллера после системного сброса остается включен. И хотя прерывания при этом отключены и срабатывание компаратора не может повлиять на работу программы, все-таки желательно отключить компаратор. Это и делается в строках 14 и 15.

Здесь уже знакомым нам способом с использованием регистра temp производится запись константы 0x80 в регистр ACSR. Регистр ACSR предназначен для управления режимами работы компаратора, а константа 0x80, записанная в этот регистр, отключает компаратор.

Настройкой компаратора заканчивается подготовительная часть программы. Подготовительная часть занимает строки 1—15 и выполняется всего один раз после включения питания или после системного сброса.

Строки 16—18 составляет основной цикл программы.

Основной цикл — это часть программы, которая повторяется многократно и выполняет все основные действия.

В данном случае, согласно алгоритму, действия программы состоят в том, чтобы прочитать состояние кнопки и перенести его на светодиод. Есть много способов перенести содержимое младшего разряда порта PD в младший разряд порта РВ. Реализован самый простой вариант – перенос производится одновременно все разряды. Для этого достаточно двух операторов.

Первый из них читает содержимое порта PD и запоминает это содержимое в регистре temp (строка 16). Следующий оператор записывает это число в порт РВ (строка 17). Завершает основной цикл программы оператор безусловного перехода (строка 18). Он передает управление по метке main.

В результате три оператора, составляющие тело цикла, повторяются бесконечно. Благодаря этому бесконечному циклу все изменения порта PD тут же попадают в порт РВ. По этой причине, если кнопка S1 не нажата, логическая единица со входа PD0 за один проход цикла передается на выход РВ0. И светодиод не светится. При нажатии кнопки S1 логический ноль со входа PD0 поступает на выход РВ0, и светодиод загорается.

Эта же самая программа без каких-либо изменений может обслуживать до семи кнопок и такое же количество светодиодов. Дополнительные кнопки подключаются к линиям PD1—PD6, а дополнительные светодиоды (каждый со своим токоограничивающим резистором) к выходам РВ1—РВ7. При этом каждая кнопка будет управлять своим собственным светодиодом. Такое стало возможным потому, что все выводы каждого из двух портов мы настроили одинаково (см. строки 8—13).

**Этап №3. Практическое программирование.**

Далее работа продолжается в окне, название которого совпадает с названием проекта. Это окно соответствует главному модулю проекта.

В окно главного модуля должен вводиться текст программы в соответствии с правилами написания программ на языке ассемблер, как показано на рисунке 8.

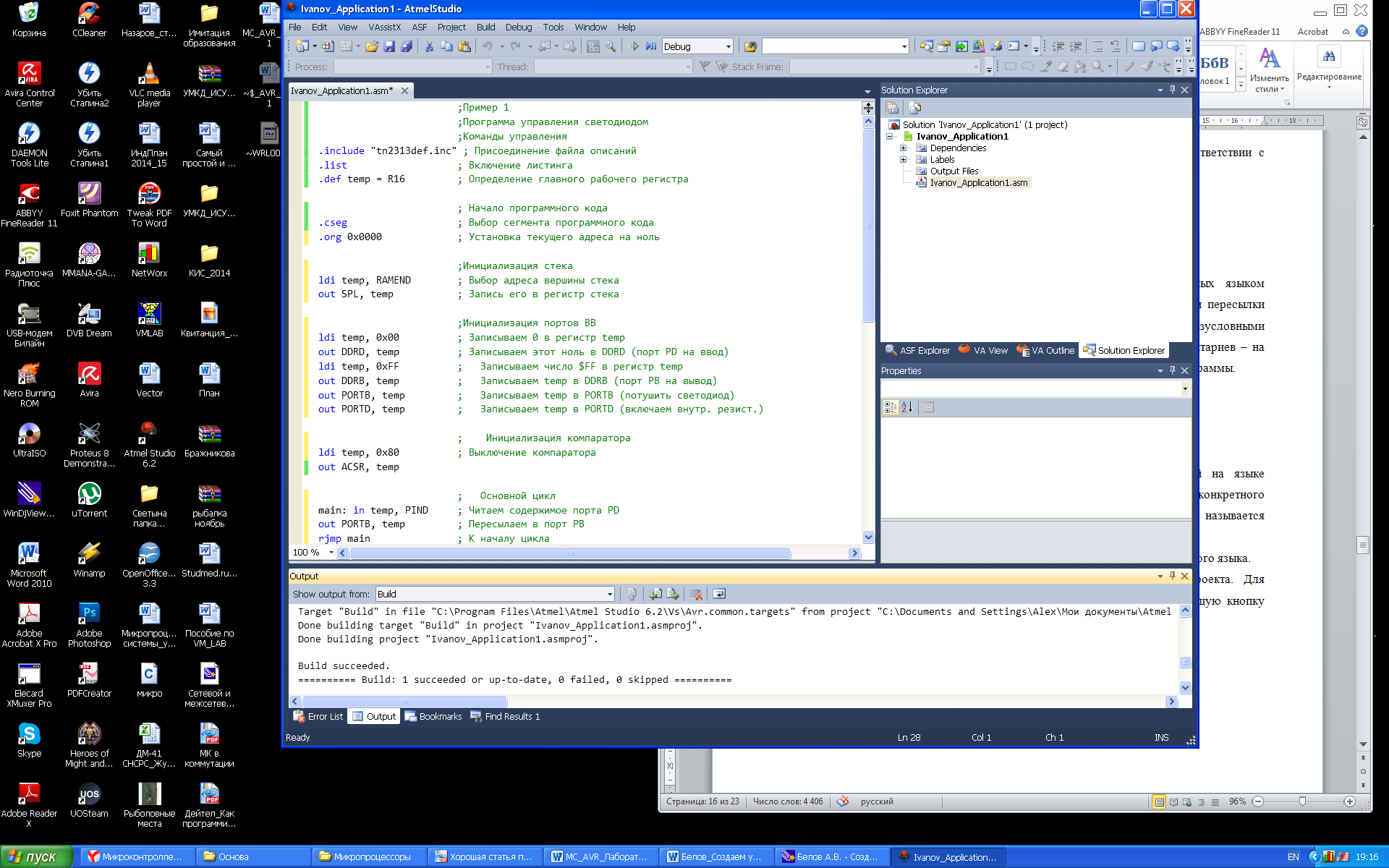


Рисунок 8 – Окно программы

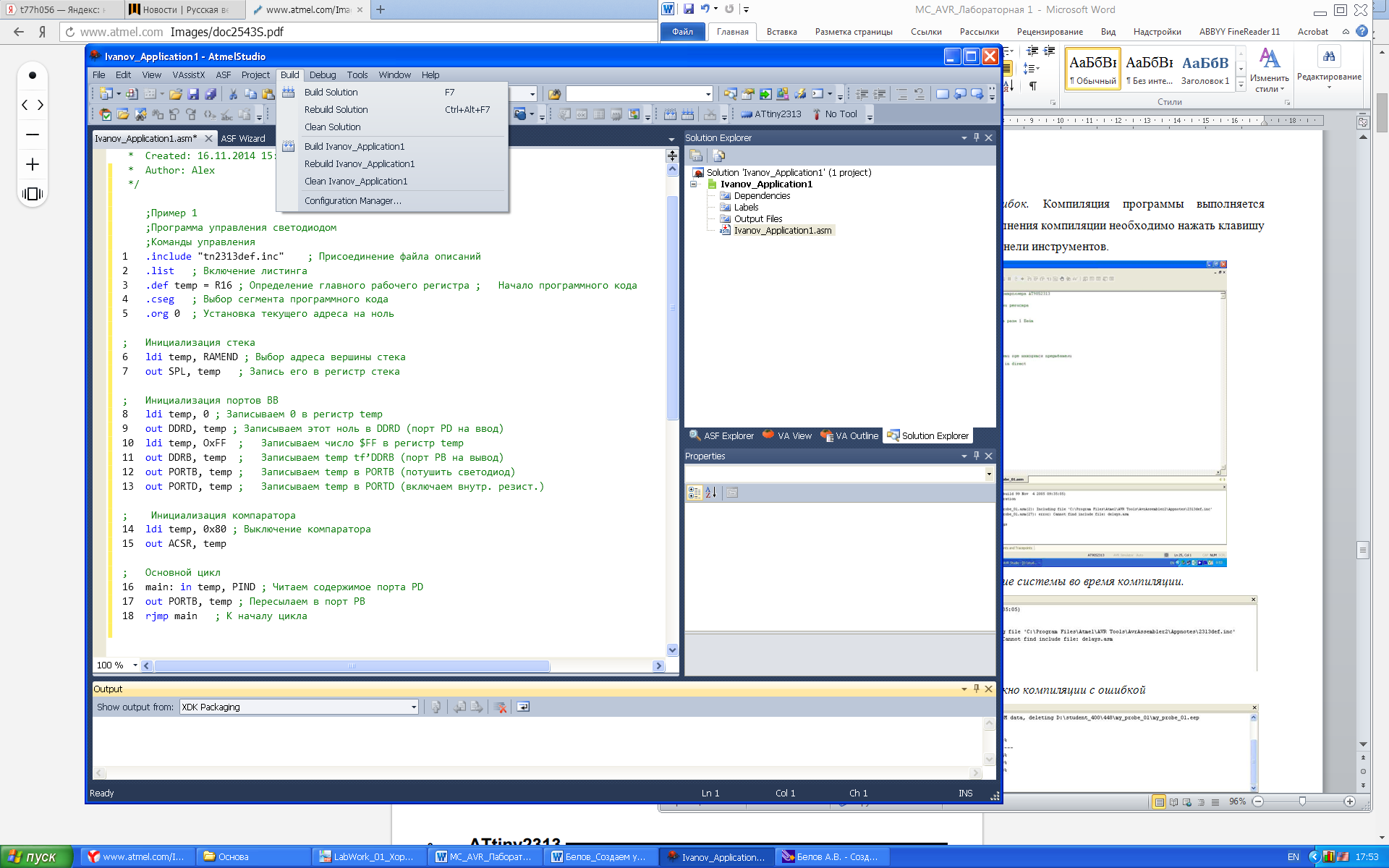
Следует подчеркнуть, что ассемблер один из самых абстрактных языком программирования, поскольку в основном программа управляет процессами пересылки между ячейками памяти, а средства управления ограничиваются условными и безусловными переходами. Поэтому приходится использовать большое количество комментариев – на начальном этапе приходится комментировать практически каждую строку программы.

*Компиляция и исправление ошибок.*

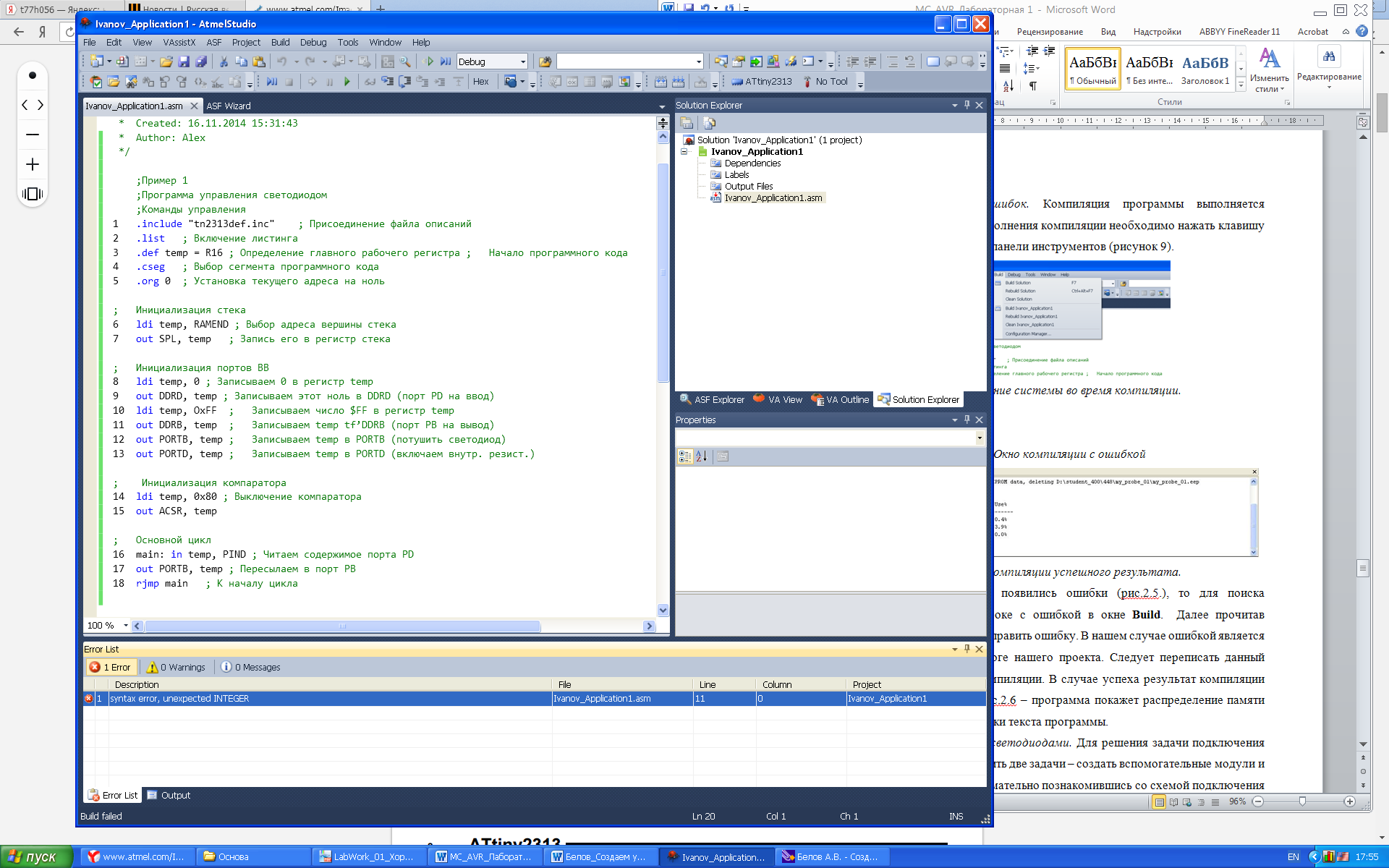
Компиляция – процесс перевода текста программы, написанной на языке программирования, в исполняемый модуль, содержащий машинные команды конкретного процессора. Применительно к микроконтроллерам процесс компиляции называется ассемблирование.

Ассемблирование — трансляция с языка ассемблера в команды машинного языка.

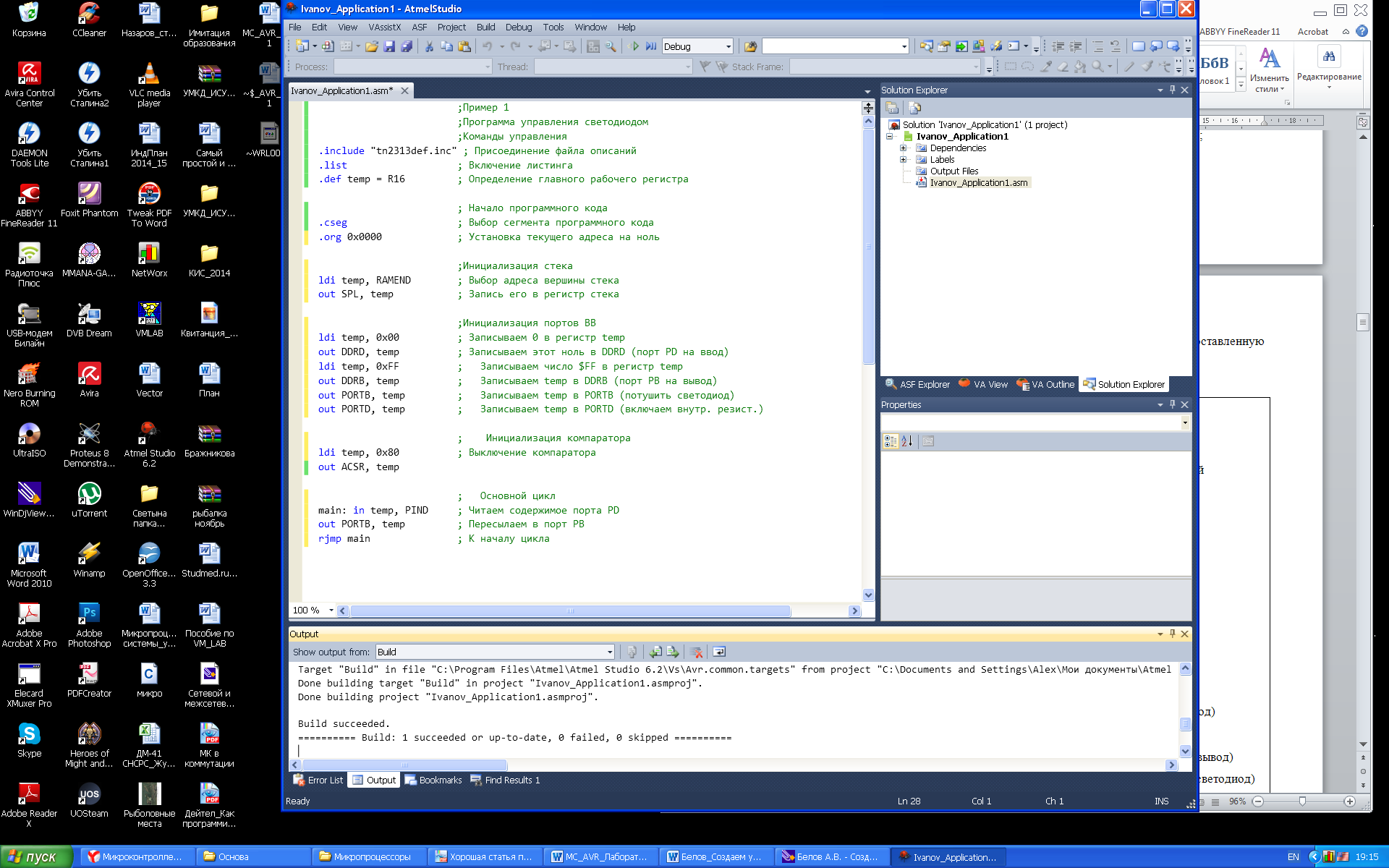
Компиляция программы выполняется автоматически в каталог проекта. Для выполнения компиляции необходимо нажать клавишу F7 или соответствующую кнопку иконки панели инструментов (рисунок 9).



*Рисунок 9- Состояние системы во время компиляции.*



*Рисунок 10 - Окно компиляции с ошибкой*



*Рисунок 11 - Окно компиляции успешного результата*

Если в результате компиляции появились ошибки (рисунок 10), то в столбце Line показан номер строки с ошибкой. Далее прочитав диагностику компилятора необходимо исправить ошибку.

После этого получаем выходной файл в формате .hex, который можно загружать во флэш-память микросхемы и наблюдать работу приложения: - зажигание светодиода.

Создайте новый проект для такого же процессора, но с использованием языка СИ.

Программа на языке СИ, в отличие от Ассемблера, гораздо более абстрагирована от системы команд микроконтроллера. Основные операторы языка СИ вовсе не привязаны к командам микроконтроллера. Для реализации всего одной команды на языке СИ на самом деле используется не одна, а несколько команд микроконтроллера. Иногда даже целая небольшая программа.

В результате облегчается труд программиста, так как он теперь работает с более крупными категориями. Ему не приходится вдаваться в мелкие подробности, и он может сосредоточиться на главном. Язык СИ так же, как и другие языки программирования, состоит из команд. Для записи каждой команды СИ использует свои операторы и псевдооператоры. Но форма написания команд в программе приближена к форме, принятой в математике.

Код данной программы, написанный на языке СИ и основные правила написания такой программы для управления микроконтроллером приведены в пособии **Белов А.В. Разработка устройств на микроконтроллерах AVR: шагаем от «чайника» до профи. — СПб.: Наука и Техника, 2013. с.139-155.**

**Задание на выполнение лабораторной работы:**

1. Скачать и установить на компьютеры программу «AtmelStudio»

2. Изучить работу программы.

3. Используя листинг 1, представленный в данном пособии изучить основные правила написания программ на ассемблере для микроконтроллеров AVR.

4. Используя пособие **Белов А.В. Разработка устройств на микроконтроллерах AVR: шагаем от «чайника» до профи. — СПб.: Наука и Техника, 2013. с.139-155.** изучить основные правила написания программ на языке СИ для МК AVR.

**Индивидуальное задание:**

Используя пособие **Белов А.В. Разработка устройств на микроконтроллерах AVR: шагаем от «чайника» до профи. — СПб.: Наука и Техника, 2013. с.139-155.** студенты с нечетным номером по списку группы создают программу переключающегося светодиода на ассемблере для МК AVR, а студенты с четными номерами - такую же программу, но написанную на СИ.

При этом варианты применяемого микропроцессора, а также подключения кнопки и светодиода будут соответствовать следующим вариантам, указанным в таблице 1:

Таблица 1 – Варианты индивидуального задания

|  |  |  |  |
| --- | --- | --- | --- |
| № студента в списке группы | Порт для подключения кнопки | Порт для подключения светодиода | Используемый процессор |
| 1 | PD1 | PB1 | Tiny2313 |
| 2 | PD1 | PB2 |  |
| 3 | PD1 | PB3 |  |
| 4 | PD1 | PB4 |  |
| 5 | PD2 | PB5 |  |
| 6 | PD2 | PB6 |  |
| 7 | PD2 | PB7 |  |
| 8 | PD2 | PB1 |  |
| 9 | PD3 | PB2 |  |
| 10 | PD3 | PB3 |  |
| 11 | PD3 | PB4 |  |
| 12 | PD3 | PB5 |  |
| 13 | PD4 | PB6 |  |
| 14 | PD4 | PB7 |  |
| 15 | PD4 | PB1 |  |
| 16 | PD4 | PB2 |  |
| 17 | PD5 | PB3 |  |
| 18 | PD5 | PB4 |  |
| 19 | PD5 | PB5 |  |
| 20 | PD5 | PB6 |  |
| 21 | PD6 | PB7 |  |
| 22 | PD6 | PB1 |  |
| 23 | PD6 | PB2 |  |
| 24 | PD6 | PB3 |  |
| 25 | PD6 | PB4 |  |
| 26 | PD1 | PB5 |  |
| 26 | PD3 | PB6 |  |
| 28 | PD5 | PB7 |  |

При защите лабораторной работы студент должен продемонстрировать с помощью отладчика «AtmelStudio» (правила работы с ним освоить самостоятельно) работоспособность своей программы и уметь ответить на вопросы о назначении в программе тех или иных директив или команд.