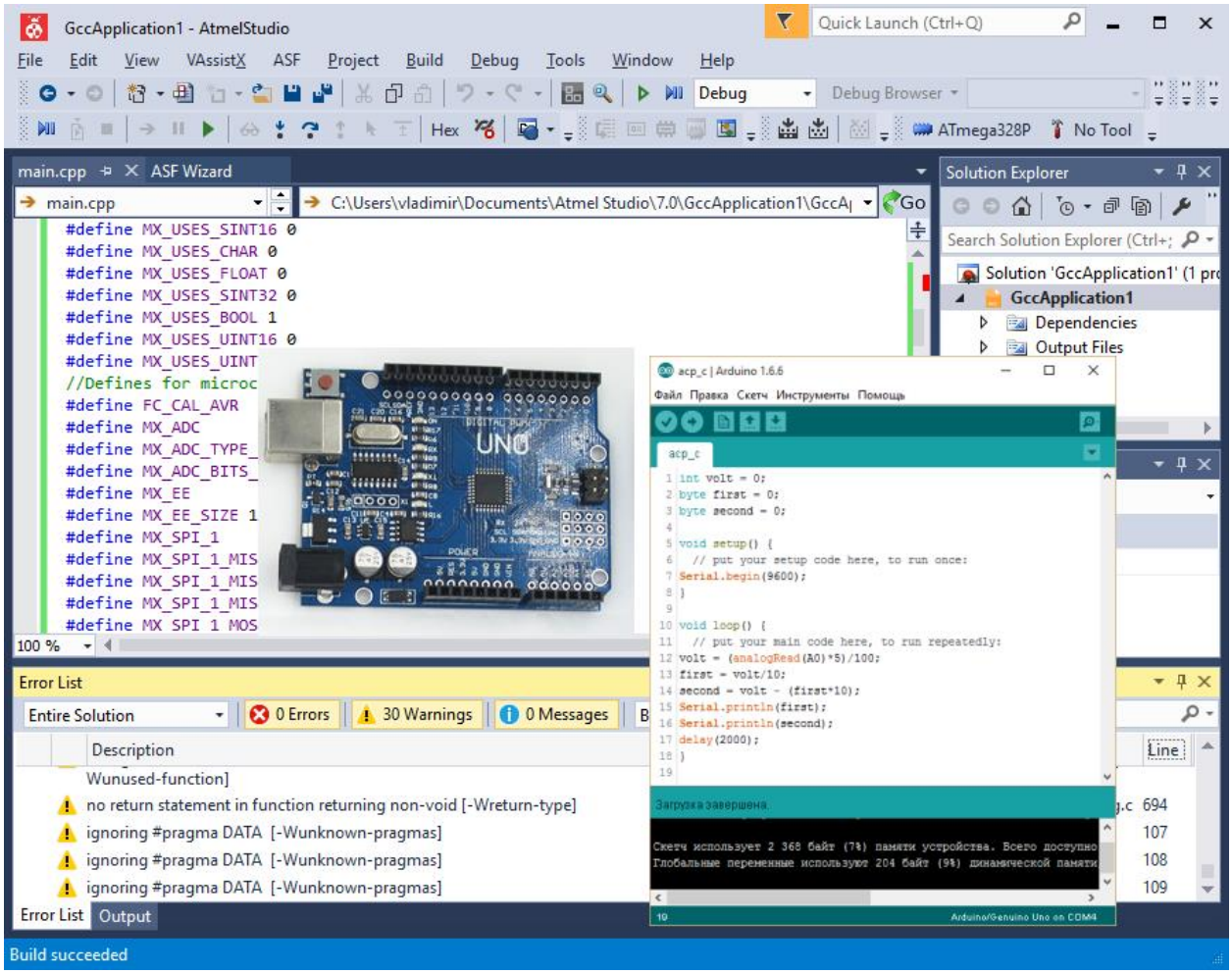
В.Н. Гололобов

Для самых начинающих

История 4. Микроконтроллер



Москва -2016

Оглавление

Если вы обзавелись модулем Arduino UNO, который в предыдущих рассказах использовали в качестве генератора и осциллографа, то в этом рассказе мы используем модуль для создания схемы вольтметра постоянного напряжения. Вам не обязательно приобретать семисегментные индикаторы, если вы не собираетесь реально собирать такой вольтметр. Мы проведём все эксперименты, пользуясь только модулем Arduino, а часть иллюстраций будет выполнена мною в программе Proteus. Но если у вас есть возможность приобрести индикаторы для проведения экспериментов, то можно купить, например, индикаторы BL-S56A-12UR, которые сегодня стоят в магазине «Чип и Дип» примерно по 40 руб.

Изобразим наше устройство, чтобы обращаться к нему позже:

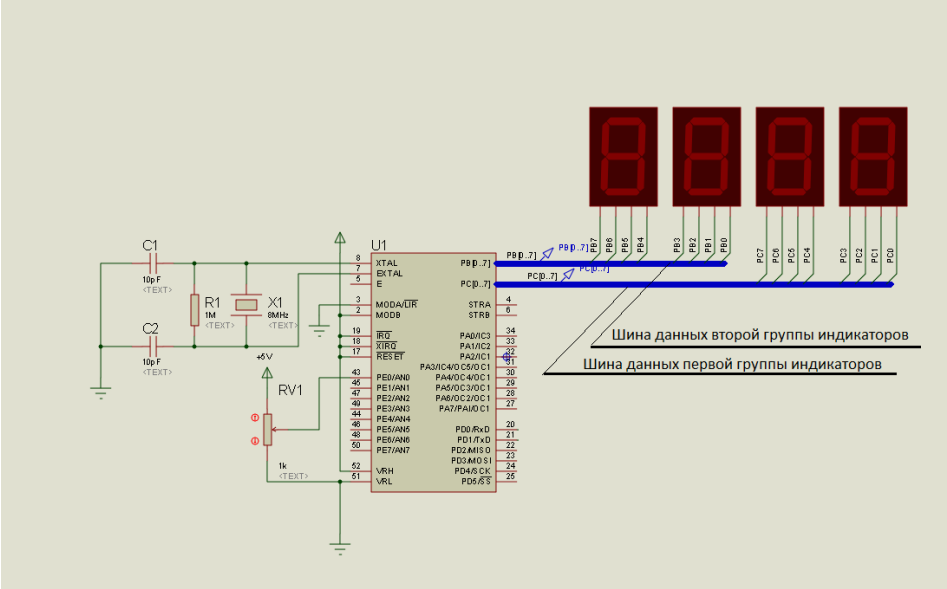


Рисунок нашего проекта для этой истории

Глава 1. Немного о процессоре микроконтроллера

Разобрать микросхему ATmega328, на базе которой сделан модуль Arduino UNO, невозможно. Поэтому приходится всему верить на слово – и тому, что будет написано в этом рассказе, и тому, что написано в большой справке к микроконтроллеру, которую называют datasheet. Можно найти подходящую вам по вашему характеру книгу о микроконтроллерах. Но можно, проделывая простые эксперименты, освоить работу с микроконтроллером, не вдаваясь слишком глубоко в историю и теорию вопроса.

В основе сути работы микроконтроллеров лежат цифровые процессоры, дальние родственники того (или тех), что обслуживает ваш компьютер. Не столь быстрые и не столь мощные они, тем не менее, способны выполнять множество операций, достаточных для реализации почти всех запросов разработчиков.

Но при всей своей красе и мощи микроконтроллер, выйдя с завода-изготовителя, беспомощен как новорождённый младенец. И это вам придётся учить его ходить и что-то делать.

Процессор, заключённый в микроконтроллер, может выполнять разные операции, как арифметические, так и логические. Однако «может», не означает, что будет. Чтобы что-то делать,

процессору необходима последовательность команд, их, и только их, он способен выполнить. Каждая команда – это двоичное число, определённое разработчиками микроконтроллера при его создании. Список этих команд можно найти в справочных данных для конкретной модели. Вот пример команд для одного из микроконтроллеров:

Мнемоника команды		Описание	Циклов	14-разрядный код		Изм. флаги	Прим.
				Бит 13	Бит 0		
Байт ориентированные команды							
ADDWF	**f,d**	Сложение W и f	1	00 0111 dfff ffff		C,DC,Z	1,2
ANDWF	**f,d**	Побитное 'И' W и f	1	00 0101 dfff ffff		Z	1,2
CLRF	**f**	Очистить f	1	00 0001 1fff ffff		Z	2
CLRW	**-**	Очистить W	1	00 0001 0000 0011		Z	
COMF	**f,d**	Инвертировать f	1	00 1001 dfff ffff		Z	1,2
DECF	**f,d**	Вычесть 1 из f	1	00 0011 dfff ffff		Z	1,2
DECFSZ	**f,d**	Вычесть 1 из f и пропустить если 0	1(2)	00 1011 dfff ffff			1,2,3
INCF	**f,d**	Прибавить 1 к f	1	00 1010 dfff ffff		Z	1,2
INCFSZ	**f,d**	Прибавить 1 к f и пропустить если 0	1(2)	00 1111 dfff ffff			1,2,3
IORWF	**f,d**	Побитное 'ИЛИ' W и f	1	00 0100 dfff ffff		Z	1,2
MOVF	**f,d**	Переслать f	1	00 1000 dfff ffff		Z	1,2

Рис. 1.1. Несколько команд микроконтроллера

Мнемоника команды используется в тексте программы на ассемблере, но процессор понимает только двоичное число, присвоенное этой команде. Так для команды очистки регистра-аккумулятора используется команда 00 0001 0000 0011.

Прочитав эту команду, процессор очистит регистр, но больше ничего делать не будет, если не будет других команд. Поэтому команд бывает много.

Если вы не знаете, что такое регистр, то вам нужно как-то представить его, иметь какую-то знакомую вам модель этой части микроконтроллера, тем более, что регистров у микроконтроллера может быть много. Вспомните, что мы говорили о триггерах-защёлках. Рассмотрим такую схему:

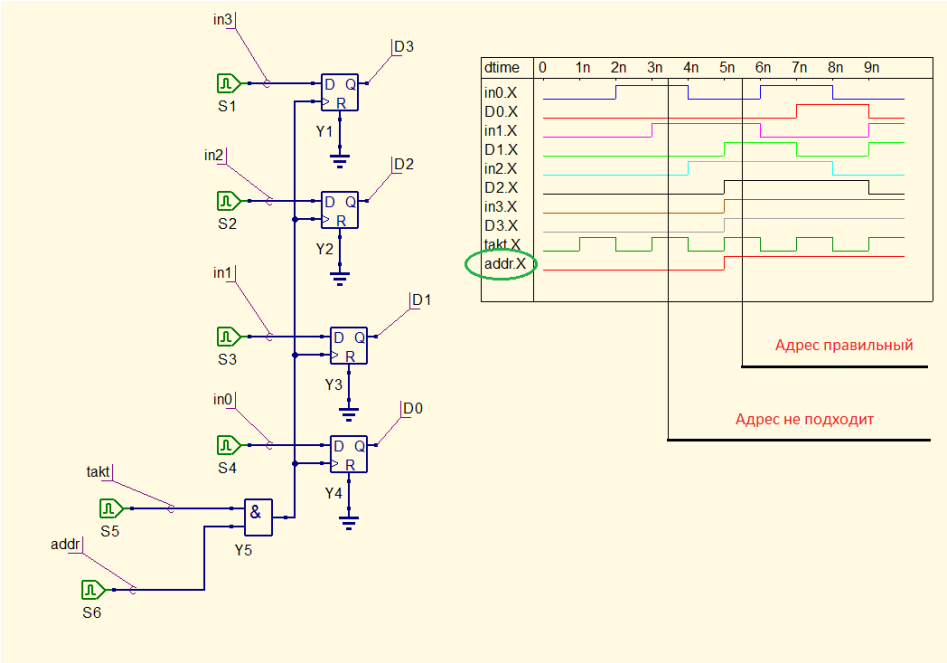


Рис. 1.2. Триггеры-защёлки для хранения данных

Конечно, реальные регистры микроконтроллера устроены иначе, но работают они похожим образом. Итак, элементы S1-S4 нужны в данном случае для формирования значений данных. Эти данные на входе триггеров обозначены как in0-in3. Когда на входы тактирования триггеров приходят импульсы, фронты этих импульсов переписывают данные на выход триггеров, где их можно читать. На выходе триггеров данные обозначены как D0-D3. Импульсы тактирования приходят от генератора прямоугольных импульсов S5. Но есть ещё схема «И», которая разрешает прохождение тактовых импульсов тогда, когда на выходе S6 появляется единица. Элемент S6 обозначен как addr, и о нём мы поговорим чуть позже.

Подав на входы триггеров данные, которые мы хотим сохранить, мы подадим импульс от тактового генератора, чтобы записать эти данные в триггеры. То есть, что-то запишем в наш импровизированный регистр. Но, чтобы данные записались именно в него, мы должны дать этому регистру некий адрес. Записав данные по этому адресу, мы не будем их менять до следующего обращения по этому адресу. Теперь, надеюсь, понятно назначение элемента S6, названного addr. Это адресный селектор. Если адрес правильный, то тактовые импульсы переписывают данные, что вы можете видеть на диаграмме. Если обращение идёт по другому адресу, то тактовые импульсы не проходят к регистру, а записанные ранее данные в нём хранятся. Всё это видно на диаграмме.

Любой адрес – это двоичное число, но и данные тоже представлены двоичным числом. Более того, и команда тоже двоичное число. Как же процессор их различает? Всё дело в том, что первое, что процессор прочитывает – это команда. В команде указывается, в сущности, адрес, где будут храниться данные, и адрес регистра, где после операции с этими данными нужно будет хранить результат. Так в показанной выше команде неявно указан адрес, по которому процессор должен осуществить операцию. За адресами следит специальный узел в процессоре, который выставляет адреса на шину адресов. Но что такое шина адресов (адресная шина)?

Кроме регистров, которые, как правило, имеют фиксированные адреса, есть ещё оперативная память, где хранятся временные данные, которые могут меняться при работе программы, их называют *переменными*. Есть программа, где в устройствах похожих на регистры хранится сама программа, то есть команды процессору. И все эти элементы имеют адреса.

На рисунке изображены четыре ячейки запоминания данных. Это сделано только для того, чтобы не загромождать рисунок. Обычно их бывает не менее восьми. В каждой ячейке могут храниться две цифры – ноль и единица. Каждая ячейка – это разряд двоичного числа. Восемь ячеек, восемь разрядов двоичного числа называют *байтом*.

Байт может хранить целое число без знака от 0 до 255. Но 256 не такое большое число, чтобы удовлетворить даже самые скромные запросы в части арифметики. Поэтому данные записывают в двух смежных «регистрах», что даёт возможность оперировать с числами от 0 до 65535. Если старший бит этого двоичного числа использовать для задания знака, скажем, плюс будет 0, а 1 будет минус, то мы сможем оперировать с числами от -32768 до +32768. Во многих случаях этого достаточно.

Похожая проблема возникает и с адресами. Если для адресов мы будем использовать только один байт, то количество адресов (или адресное пространство) будет очень невелико даже для простых случаев. Поэтому и для адресов используют по два байта. И для данных, и для адресов в современных компьютерах используют большее количество байт, но нам хватит пока и того, о чём мы говорили.

Каждая связь узла адресации (каждый бит адреса) приходит к соответствующему входу адресного селектора, определяющего один адрес, а совокупность этих связей образует *адресную шину*, тогда как разрядность битов адреса даёт нам *адресное пространство*. Кроме шины адресов есть *шина данных*, которая имеет столько разрядов, сколько задумано разработчиками для работы с числами.

Вновь вернёмся к работе процессора внутри микроконтроллера.

При включении питания все узлы микроконтроллера приходят в исходное положение, такое начало называют «сбросом» или сбросом по питанию. При этом процессор начнёт свою работу с обращения к *программной памяти* (где записана программа) по нулевому адресу (иногда это старший адрес). Здесь записана самая первая команда. Если этого не произойдёт, если адрес будет неверным, то вместо команды процессор может прочитать какие-то данные, которые не отвечают ни одной из команд или относятся к неправильной команде. В этом случае всё пойдёт «наперекосяк», скорее всего, процессор «зависнет», он не сможет дальше работать. Если ваш компьютер когда-нибудь зависал, то возможной причиной была эта.

Все команды процессора продуманы так, чтобы он знал, где взять данные, куда их отправить – в регистры (и какие), что с данными сделать. Чаще всего результат будет отправляться в регистр-аккумулятор, откуда результат может отправиться, например, на выход или в оперативную память по адресу какой-то переменной.

Все эти шаги расписаны отдельными командами, список которых довольно большой, и вам предстоит их все записать с помощью мнемоники команд, если вы решили писать программу на ассемблере. Позже вы оттранслируете эту запись в двоичные коды команд с помощью специальной программы, которая называется ассемблер. Используя ассемблер, вы контролируете почти все операции процессора, контролируете каждый его шаг. Но даже для простых действий шагов оказывается довольно много, а запоминать мнемонику команд трудно, поэтому программисты давно придумали, как упростить процесс программирования.

Глава 2. Немного о программе

Любая программа – это указания процессору, что ему следует делать. Но, начиная создавать программу, вы в первую очередь определяетесь не с командами процессору. Вы решаете, что должен делать микроконтроллер, то есть, в нашем случае измерять напряжение на потенциометре и отображать его в десятичном виде на семисегментных индикаторах, как мы решили в самом начале. Вот так будет выглядеть ваша программа изначально.

> *Повторим её: микроконтроллер должен измерять напряжение на потенциометре и отображать это напряжение на семисегментных индикаторах.*

Что дальше?

Если вы хотите только повторить готовую схему, то вам понадобится всё правильно соединить, загрузить готовую программу в микроконтроллер, чтобы понаблюдать за результатом работы. Что вам это даст? Готовый вольтметр. Но у вас уже есть мультиметр, который сделает это не хуже, если не лучше.

Итак. Программирование – это создание программы, которое начинается с продумывания программы. Процесс продолжается написанием текста программы на выбранном языке программирования, за которым следует отладка программы, а завершается всё трансляцией программы в форму, которую можно отправить в микроконтроллер, где она осядет в памяти микроконтроллера, предназначенной для хранения программы. Процесс отправки программы в микроконтроллер называют загрузкой программы в микроконтроллер, для чего служат программаторы. Модуль Arduino, и это удобно, не требует программатора, он уже есть в модуле.

Продумывание программы мы начнём в следующей главе, поскольку мы не выяснили, например, что такое семисегментный индикатор?

А сейчас несколько слов о языках программирования.

Я уже упоминал о том, что текст программы можно написать на ассемблере. Вы можете этим заняться, если увидите в этом смысл. Текст программы можно написать на языке высокого уровня. Сегодня все языки высокого уровня, которые раньше предполагалось использовать для разных целей: Basic для начинающих, Pascal для обучения в ВУЗе и т.д., - сегодня все языки программирования претерпели изменения, превращающие их в удобный инструмент для создания программ. Пусть вас не смущает, что вам нравится Basic – он не хуже других языков программирования.

В последнее время появились графические языки программирования. Не обращайте внимание, что их кто-то ругает. Если вам нравится графический язык программирования, пользуйтесь им.

Но мы договорились, что будет использовать модуль Arduino, поэтому будем использовать то, что есть для этого модуля в части среды разработки. Это будет либо программа Arduino, либо ещё что-то, что позволяет работать с Arduino.

Любой язык высокого уровня, чаще других используется язык Си, программу транслирует, то есть, переводит. Вначале программу, написанную на языке Си, транслируют на ассемблер, потом программу на ассемблере транслируют в форму либо машинных кодов, как для компьютера, либо в форму для загрузки в микроконтроллер (чаще всего hex-файл). Раньше все эти процессы, включающие ещё и компоновку программы, выполнялись по отдельности, сегодня вы нажимаете иконку трансляции в меню среды разработки и получаете готовый файл. С модулем Arduino вы даже получаете микроконтроллер с уже загруженной и работающей программой.

Поскольку вам предстоит научить микроконтроллер «ходить», давайте научимся сами, как сделать первый маленький шаг. На плате модуля Arduino есть светодиод, который мы в качестве первого шага зажжём.

Для этого вам следует установить программу Arduino, о чём мы говорили ранее. У меня одна из последних на сегодня версий программы устанавливается, как любая программа Windows, в раздел «Program Files» на диске. В Linux она будет в другом разделе, но появится в списке других программ.

Чтобы запустить среду разработки программ Arduino, выберите её из списка, щёлкните левой клавишей мышки по названию, и вы получите такой «блокнот программиста»:

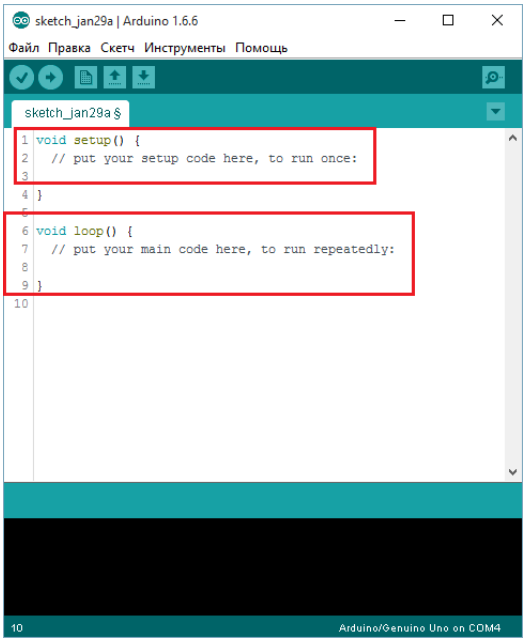


Рис. 2.1. Запуск программы Arduino

Язык программирования, используемый в компиляторе (трансляторе, переводчике) Arduino похож на язык Си, поэтому многое выглядит так, как написанное на языке Си.

Ваши программы называются здесь скетчами, что не так важно, вы сохраните программу под тем именем, который вам нужен. Когда вы начинаете новую программу, то получаете шаблон, заготовку под программу, где есть две необходимые части, они выделены на рисунке. Это две части или блоки программы, первый из которых предназначен для настройки микроконтроллера. Надпись в этом блоке гласит: добавьте здесь ваш код, который будет выполнен один раз. Это так. В этом блоке вам предстоит объяснить контроллеру многое, но не сейчас.

Второй блок предназначен для самой программы, которая, как гласит надпись, будет повторяться многократно (бесконечно).

Что же мы должны объяснить контроллеру в первом блоке программы. Пока немногое. Процессор микроконтроллера укрыт «за забором» портов ввода-вывода. Порт можно представить себе так же, как мы представили модель регистра. Реальный порт может быть устроен иначе, но модель будет работать похожим образом. Каждый вывод (или почти каждый, это следует уточнить в datasheet) может работать в качестве цифрового ввода и вывода. Что это означает?

На вывод микроконтроллера можно подать сигнал высокого или низкого уровня, если мы его «запросим» как цифровой ввод. И каждый вывод микроконтроллера можно установить в высокий или низкий уровень, если мы его назначим выходом. Для порта в том виде модели, каким мы его себе представили, есть один нюанс – мы записываем в нашей модели все значения в порт. Но, если мы хотим изменить только один вывод порта, что можно сделать? Можно прочитать ранее сделанную запись в порт, скажем в регистр, изменить нужное значение в регистре, а затем переписать регистр в порт. Такой механизм изменения значения одного вывода порта используется часто и называется методом «чтения-модификации-записи». Не забывайте об этом, как сделал я однажды и был неправ.

Мы знаем, что светодиод через резистор можно подключить к источнику питания, и тогда светодиод будет светиться. В качестве источника питания в данном случае мы используем выход микроконтроллера. Любой микроконтроллер может через свои выводы пропускать довольно значительный постоянный ток, конечно по меркам электронных слаботочных устройств. Во всяком случае, этого тока достаточно, чтобы запитать индикаторный светодиод. А нам достаточно, заказав вывод 13 (к этому выводу у модуля подключён светодиод) в качестве выходного в первом блоке программы, установить его в единицу (или высокий уровень, HIGH).

Запись настройки вывода на выход на языке Arduino выглядит так: pinMode(13, OUTPUT);

Как вы видите, она гласит: режим_вывода(здесь два параметра, номер вывода и выход, то есть, сам режим работы вывода). На этом пока всё. Обратимся к программе.

Чтобы установить вывод 13 в состояние высокого уровня, мы должны написать: digitalWrite(13, HIGH); И это ясно: цифровая_запись(номер вывода, состояние). В итоге мы получим:

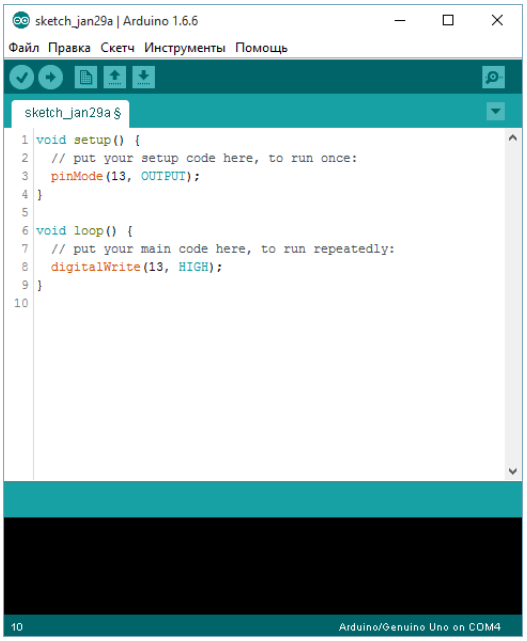


Рис. 2.2. Первая программа

Перед тем, как продолжить работу, программу нужно сохранить.

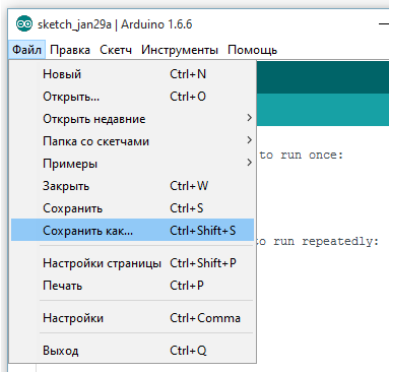


Рис. 2.3. Сохранение программы

По умолчанию при установке программы Arduino создаётся место для хранения программ, но можно создать свою папку, где хранить свои программы. В выборе имени для программы… если вы намерены и дальше что-то программировать, используя разные среды программирования, постарайтесь записывать имена латиницей, используя недлинные и понятные вам названия. Иной раз до достижения цели приходится делать множество предварительных «эскизов», которые вам понадобятся, если что-то пойдёт не так, если придётся вернуться к началу. Придумайте для себя систему названий, которая вам понятна и удобна. Я, признаться, в спешке порой придумываю неудачные названия, а позже сам не могу вспомнить это название. Учитесь на чужих ошибках, хотя не бойтесь и своих, но постарайтесь их избегать.

После того, как вы сохранили программу под каким-то именем, это имя появится в заголовке. Теперь, если вы ранее не настроили программу под ваш модуль, выберите модель вашего модуля и виртуальный порт, к которому он подключен.

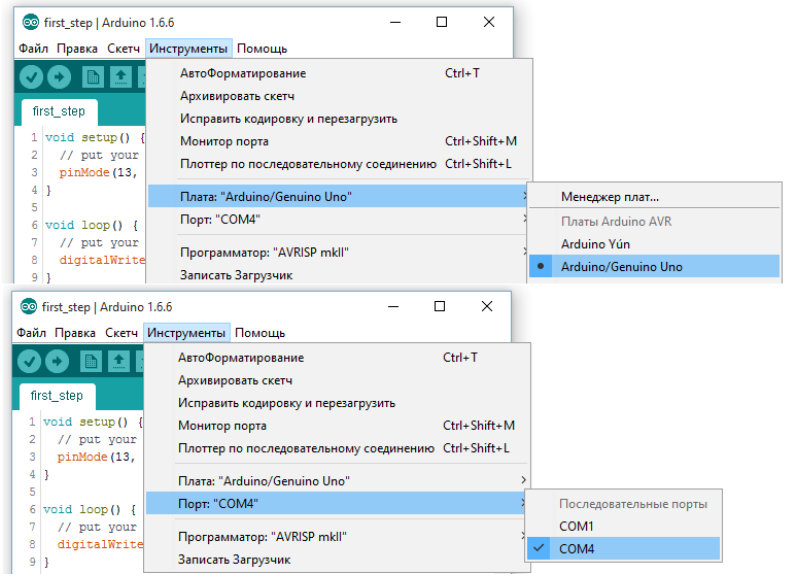


Рис. 2.4. Настройка программы

После настройки программы вы можете проверить, правильно ли написана программа:

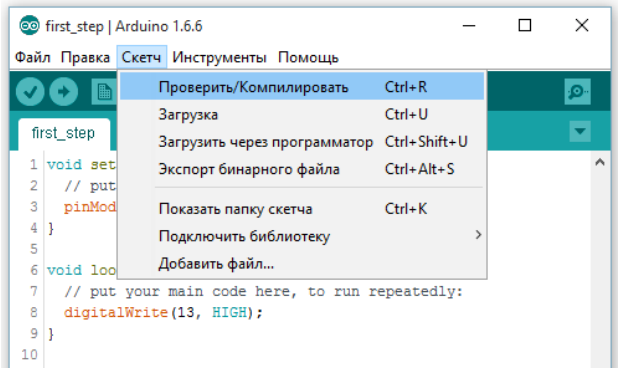


Рис. 2.5. Предварительная компиляция программы

А может сразу загрузить её в модуль Arduino, используя иконку со стрелкой. Если вы обошлись без проверки, но в программе что-то не так, то в нижней части окна появятся сообщения об ошибках.

Далеко не всегда в программе столько ошибок, сколько вы их увидите в окне сообщений. Иногда одна ошибка влечёт за собой ряд других.

Поэтому при появлении ошибок используйте прокрутку окна, чтобы найти самое первое сообщение об ошибке. Кроме сообщений об ошибках появляются предупреждения, как правило, они не мешают работе программы, но это не означает, что на них можно не обращать внимания.

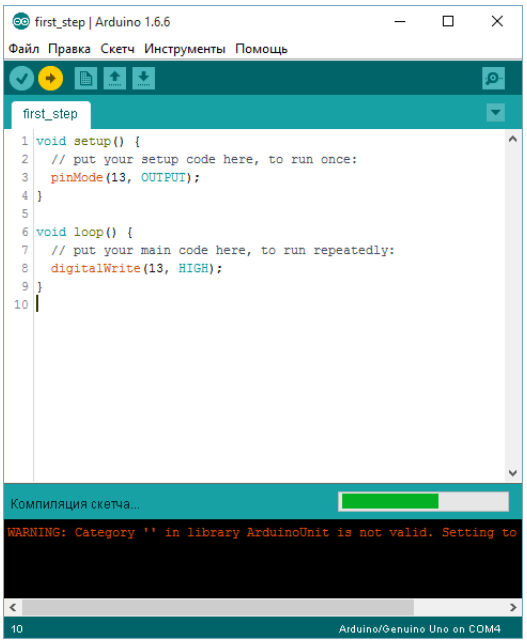


Рис. 2.6. Загрузка программы в модуль Arduino

Теперь, если у вас такой же модуль Arduino как у меня, вы увидите два горящих светодиода – один индицирует наличие питающего напряжения, а второй зажгли мы.

Измените текст основной программы, пусть вас не смущает, что одну строку я называю текстом программы, следующим образом: digitalWrite(13, LOW);

Загрузите новую программу в модуль, вы увидите, что горит только один светодиод. Мы сделали ещё один маленький шаг. И обратите внимание: *любая инструкция на языке Си завершается точкой с запятой!* Эти инструкции объединяют несколько команд процессору из набора для данной модели, чем они и интересны: инструкции языка высокого уровня понятнее, чем мнемоника ассемблера, они представляют блок процессорных команд, образуя макрокоманду.

Давайте сделаем ещё один маленький шаг, объединив эти две программы, но между зажиганием и гашением светодиода добавим паузы. Здесь они записываются так: delay(1000);

Текст программы «потолстел» и имеет вид:

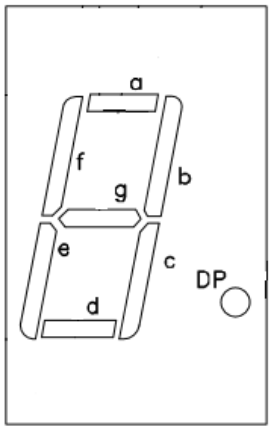
```
void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Это первая программа, с которой начинают своё знакомство с программированием очень многие, не беспокойтесь, если кто-то будет говорить, что это «детский сад». Можно подумать, что они родились не просто в рубашке, а в смокинге и с сигарой во рту. Все мы ходили в коротких штанишках!

Глава 3. Внешние элементы и внутренние модули

Трудно придумать устройство, где кроме микроконтроллера не было бы ничего. Даже генератор, который можно выполнить на контроллере, следует снабдить делителем напряжения. Более того, не использовать кнопки для изменения частоты импульсов – это было бы странно.

Сейчас нас интересует такой внешний компонент схемы, как семисегментный индикатор. Он содержит семь светодиодов специальной конструкции, чтобы при зажигании они выглядели светящимися полосками. Аноды или катоды соединяются вместе, а на оставшиеся выводы через резисторы подают напряжение.



Расположение и обозначение сегментов

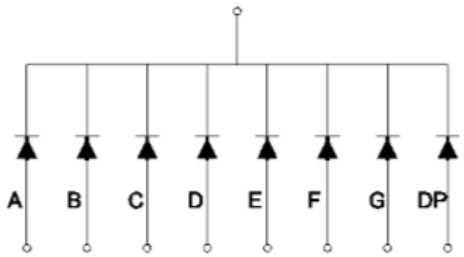


Схема включения светодиодов сегментов

Рис. 3.1. Семисегментный индикатор

Светодиод, обозначенный как DP – это десятичная точка, которую можно использовать при необходимости. Комбинируя включение отдельных сегментов можно высветить все цифры от 0 до 9. Так, зажигая сегменты b и c, мы получим единицу, с помощью сегментов a, b и c мы получим семёрку…

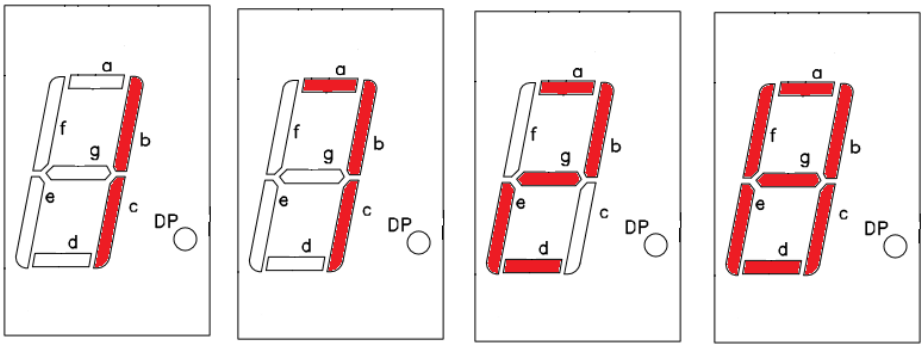
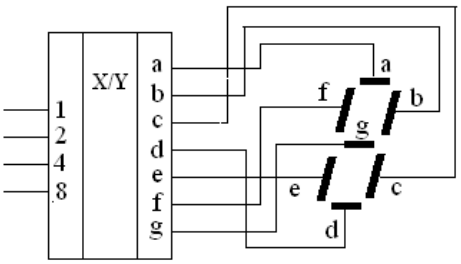


Рис. 3.2. Несколько цифр на семисегментном индикаторе

Мы можем записать в порт любую цифру от 0 до 9 в двоичном виде, для этого достаточно четырёх битов. Но мы не можем к этим четырём выводам порта подключить семь выводов индикатора. Мы можем подключить все восемь выводов индикатора, с учётом точки DP, к восьми выводам порта, но тогда записанная в порт двоичная цифра не будет нужной нам для отображения на индикаторе цифрой от 0 до 9.

Есть два решения проблемы: мы можем как-то позаботиться о преобразовании цифр в число, записываемое в порт, это раз; мы можем использовать внешний компонент.

Внешний компонент, который нам нужен, называется дешифратором семисегментного индикатора. Это микросхема, четыре входа которой подключаются к четырём выводам, дающим двоичные цифры, а семь выходов подключаются к сегментам индикатора. Удобной будет микросхема КР514ИД1, которая имеет встроенные токоограничительные резисторы (что обязательно следует проверить или добавить резисторы, если нет уверенности в справедливости утверждения о наличии резисторов), она работает с индикаторами, имеющими общий катод. Для индикатора с общим анодом потребуется микросхема КР514ИД2 и резисторы между выходами и сегментами индикатора. Подключение дешифратора к индикатору выглядит так:



На рисунке изображена схема подключения дешифратора, но следует обратиться к справке по микросхеме дешифратора, чтобы правильно соединить его с индикатором. Входы соединяются так, что 1 – это младший бит, 8 – старший.

Рис. 3.3. Схематическое соединение дешифратора с индикатором

Применить два индикатора с дешифраторами – не столь плохая идея. Она упростит программирование. Но, если вы хотите подключить блок из четырёх индикаторов, вам придётся проделать больше работы при программировании.

Теперь посмотрим, что можно сказать про АЦП, встроенный в микроконтроллер модуль.

Не задаваясь целью в точности рассмотреть схему, не думаю, что её так легко будет найти, мы можем обратиться к модели, которая пояснит нам принцип работы этого встроенного модуля.

Мы имеем представление о счётчике, считающем приходящие на его вход импульсы. Мы имеем представление о компараторе, который сравнивает напряжение на одном его входе с напряжением на другом входе. Представим, что у нас есть некое цифровое устройство, которое при каждом импульсе на входе увеличивает напряжение на выходе на некоторую фиксированную величину. Как могло бы выглядеть такое устройство, собранное из знакомых нам элементов?

Мы знаем, что четыре D-триггера, включённые определённым образом, превращаются в счётчик. Добавим тактовый генератор и немного странный делитель из четырёх резисторов. Пусть вас не смущает, что микросхема двоичного счётчика SN74193 имеет больше входов и выходов, чем мы использовали ранее. Не обращайте пока на них внимание. Нас интересует вход для подключения тактового генератора VG1, обозначенный как UP, и четыре выхода счётчика, обозначенные как QA, QB, QC и QD. Про остальные входы и выходы, которые мы не использовали, вы можете прочитать

в описании этого счётчика. Есть и другие счётчики, попроще, но я не нашёл их в программе, которую использовал для демонстрации. Может быть, плохо искал, так тоже бывает.

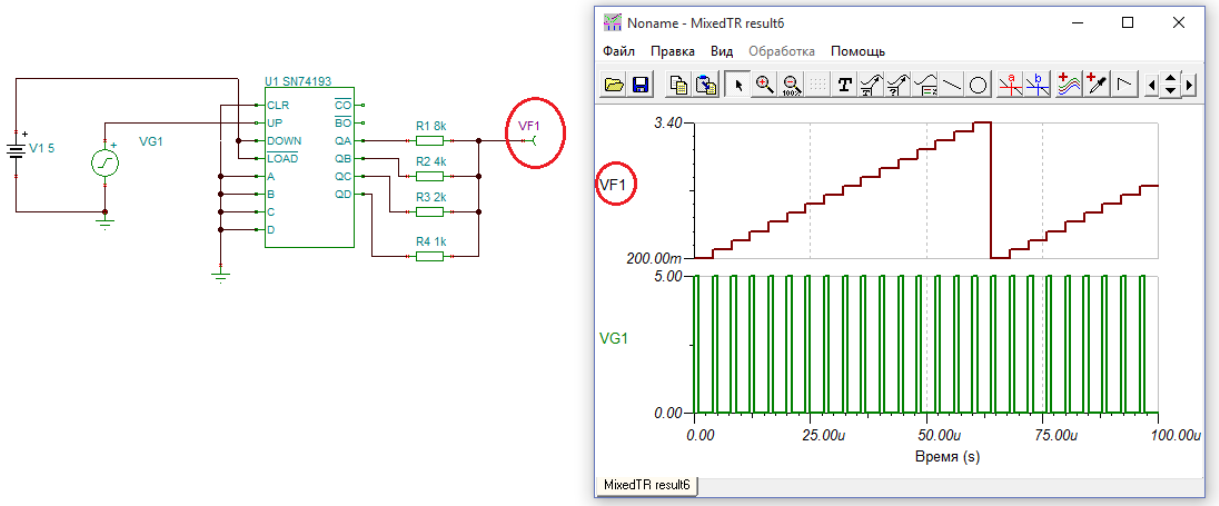


Рис. 3.4. Получение ступенчатого напряжения с помощью счётчика

Нижняя диаграмма показывает тактовые импульсы от генератора, а верхняя, что с приходом каждого из импульсов напряжение увеличивается на небольшую фиксированную величину. Сигнал такого вида называют ступенчатым напряжением. И, если отбросить непривычный вид микросхемы счётчика, то мы обошлись уже знакомыми деталями.

Дополним нашу схему компаратором, сделанным из знакомого нам операционного усилителя.

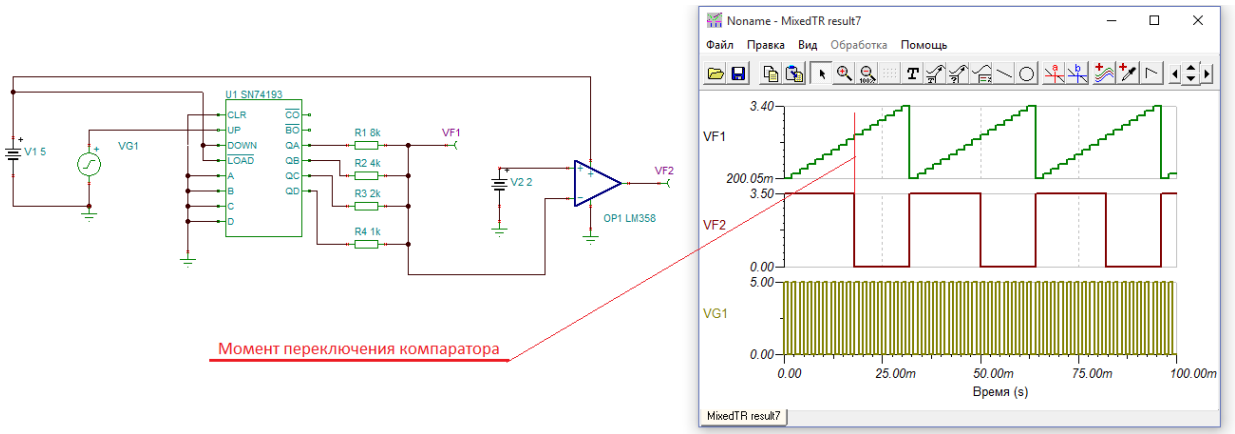


Рис. 3.5. Схема с компаратором

На один их входов компаратора подаётся аналоговый сигнал, в данном случае напряжение от батарейки V2 = 2В, на другой вход компаратора подаётся ступенчатое напряжение. Когда ступенчатое напряжение превышает 2 В, компаратор переключается.

Теперь представим, что инвертированным выходным напряжением от компаратора мы производим запись в наш регистр, построенный из D-триггеров, входы которых подключены к выходам счётчика. В регистр запишется, сколько тактовых импульсов было подано на счётчик до

момента совпадения напряжений на входах компаратора. Или иначе, сколько было ступенек нарастающего напряжения.

Мы превратили аналоговое напряжение 2 В в двоичное число. Зная, сколько всего ступенек, мы можем посчитать величину каждой из ступенек напряжения. Умножив двоичное число, записанное в регистр, на напряжение одной ступеньки, мы получим (с некоторой погрешностью) значение равное 2 В.

Реальный аналого-цифровой преобразователь может работать не совсем так, но нам нужна модель, чтобы понять, как можно аналоговый сигнал (напряжение) превратить в двоичное число.

Кроме встроенного в микроконтроллер модуля АЦП есть ещё один встроенный модуль, которым мы воспользуемся далее. Этот модуль выводит в определённом формате двоичное число в виде последовательности импульсов. Существуют разные варианты подобного обмена данными. Мы рассмотрим один из них в формате, который используется COM-портом, но с импульсами другого вида.

Единице соответствует наличие импульса, нулю отвечает отсутствие импульса. Данные передаются байтами, предваряются стартовым битом и завершаются стоповым битом. Иногда таких битов бывает два, и есть возможность добавить бит чётности. Поскольку при обмене данными не используется передача тактовых импульсов, импульсов синхронизации, скорость обмена фиксирована и заранее обговаривается. Если у вашего компьютера есть COM-порт, то в его настройках можно посмотреть, какие скорости передачи данных используются.

Для модели мы возьмём те же знакомые нам компоненты: счётчик и двухвходовые схемы «И». Посмотрим, получится ли у нас моделировать работу UART – модуля последовательной передачи данных. Счётчик будет, правда, не двоичный, а десятичный. То есть, каждая цифра на выходе будет принимать высокое значение с приходом очередного тактового импульса, тогда как остальные будут обнуляться.

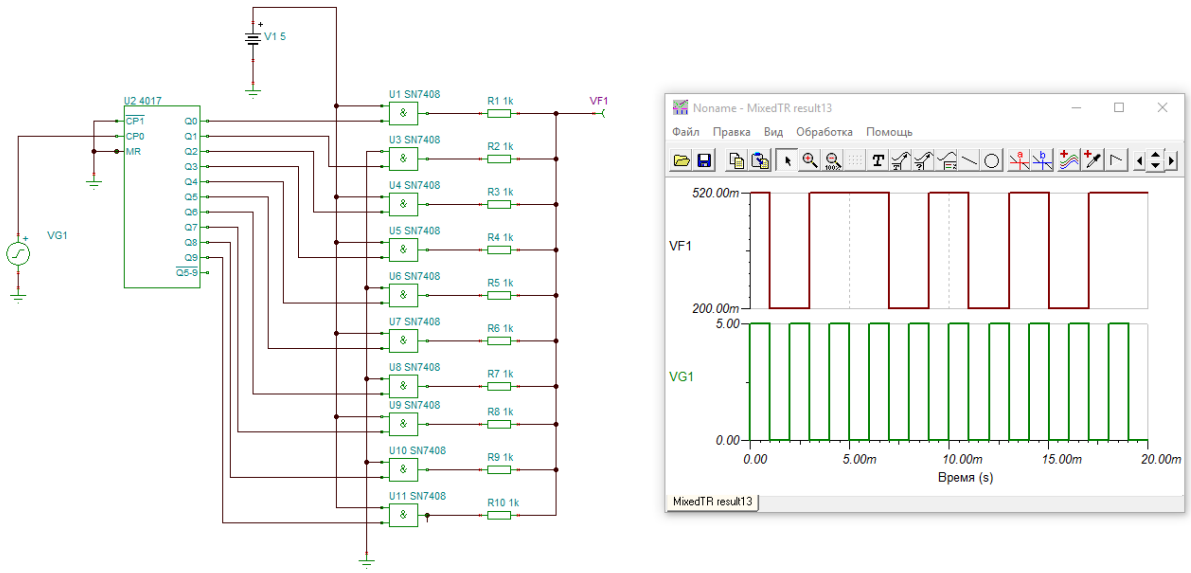


Рис. 3.6. Формирование последовательной передачи байта данных

На схеме десятичный счётчик с приходом импульсов от генератора VG1 (сигналы на нижней диаграмме) устанавливает в высокий уровень один из выходов Q0-Q9. Этот выход подаёт единицу на один из элементов «И». Второй вход такого элемента подключён либо к питанию (единица), либо к земле (ноль). Эти подключения и формируют передаваемый байт элементами U3-U10. Микросхемы U1 и U11 формируют стартовый и стоповый биты. Старший бит данных (U10) равен нулю. Младший бит данных (U3) равен тоже нулю, а весь байт в двоичном виде выглядит так:

0101 0110

На верхней диаграмме можно прочитать это число, учитывая тот факт, что первым передаётся младший бит. Совсем не обязательно запоминать эту схему, предложенную в качестве модели, но модель даёт некоторое представление о том, как это может быть сделано. С учётом того, что те входы элементов «И», которые сейчас «жёстко» привязаны к земле и напряжению 5В, будут соединены с регистром, куда записывается число, полученное нами в модели АЦП.

Глава 4. Возвращаясь к программированию

Языки программирования высокого уровня позволяют при написании текста программы использовать такие понятные команды процессору как read, write или print, не задумываясь над теми маленькими шажками, которые будет делать процессор.

Язык Arduino подобно языку Си использует такой полезный механизм как библиотеки функций. Что такое функция?

Любая операция с числами, возьмём сложение, должна выполнить математическое действие: 1+1=2. В общем виде эту операцию можно записать так: a+b=c. Языки высокого уровня позволяют записать подобный общий вид как функцию. Её ещё можно называть подпрограммой, то есть, выделенной частью программы, которой даётся имя. По этому имени функцию можно вызывать в программе многократно. Иногда функции не требуются параметры, в противном случае они, параметры, записываются в скобках. Так параметрами для функции сложения будут те самые a и b. Сама функция может выглядеть так: sum(a, b). Обращаясь в программе к этой функции, мы запишем: sum(1,1); - в качестве параметров в скобках мы использовали нужные нам числа.

В том случае, когда нас не интересует результат работы функции, она ничего не возвращает, но в данном случае мы хотели бы знать сумму. На языке Си принято записывать возвращаемое значение командой *return* c;

Написав в программе один раз эту функцию, мы можем многократно обращаться к её услугам, что называется вызовом функции. Как вы уже догадываетесь, функций существует великое множество. Чтобы не переписывать эти полезные блоки программы каждый раз, когда вы пишите новую программу, а написать нужную вам функцию бывает тоже непросто, программисты придумали механизм библиотек функций. Включив в начале программы нужную библиотеку, вы можете на протяжении всей программы вызвать нужные вам функции, вошедшие в эту библиотеку функций. Однако вернёмся к микроконтроллеру.

Встроенный в микроконтроллер аналого-цифровой преобразователь может работать с положительным напряжением от 0 до 5 вольт. То есть, минус соединяется с землёй, а плюс с выводом, который может считывать аналоговый сигнал. В отличие от цифровых сигналов,

аналоговые сигналы можно подавать не на все выводы портов. У модуля Arduino эти выводы собраны в отдельную линейку и обозначены как A0, A1 и т.д.

Команда чтения проста и понятна: analogRead(A0);

Место A0 может занимать любой из аналоговых входов. Считывание значения, цифрового или аналогового, как правило, происходит в некую ячейку памяти, которой придаётся имя, и которая теперь называется *переменной*. Что подразумевает, значение здесь может меняться на протяжении работы программы. Если к этой переменной обращение идёт в любом месте программы, то переменную называют *глобальной*, её объявляют в самом начале программы. Если переменная нужна только в одной подпрограмме, то её объявляют в этой подпрограмме, и переменная будет *локальной*. Дело в том, что локальные переменные занимают место в памяти только на время работы подпрограммы, освобождая его для других переменных по завершении работы подпрограммы (или функции). Переменные могут быть разных типов (разных форматов). Чтобы программа знала, сколько места в памяти следует отвести для этой переменной, переменную объявляют, записывая её тип. Переменная типа byte (char, uint8) занимает байт в оперативной памяти. Переменная типа int (от integer, целое) занимает два байта в памяти.

Чем меньше ступеньки при работе АЦП, тем выше разрешение, а для этого, вспомните нашу модель, используют 10 бит для получения результата. Поэтому для записи результата нам нужна переменная типа int, которую мы назовём volt.

Запустив программу Arduino, напишем нашу программу чтения напряжения:

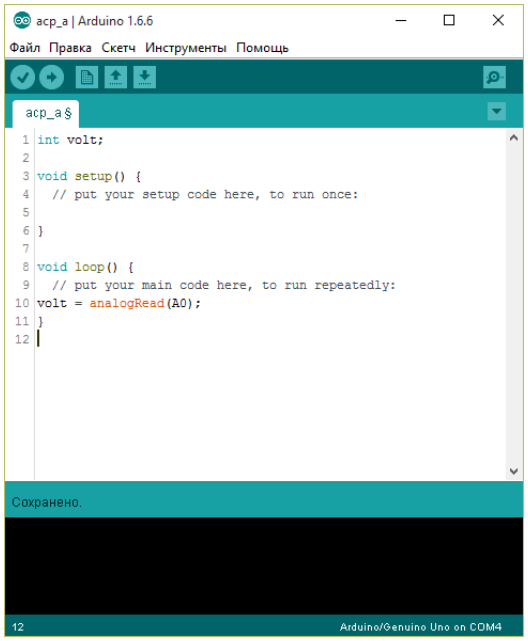


Рис. 4.1. Программа чтения напряжения

Спасибо тем, кто о нас позаботился, мы легко написали программу. Однако, и мы об этом говорили, всё, что делает процессор, укрыто от нас портами ввода-вывода. Мы можем подключить батарейку 1.5 В к выводам A0 и GND. Но мы даже не поймём, прочитали мы напряжение или нет. Позже мы постараемся вывести прочитанное значение на индикаторы. Но сейчас мы поступим иначе. Мы выведем полученное значение переменной через встроенный

модуль последовательного обмена данными. Для этого добавим настройку модуля в первый блок, добавим команду вывода во второй блок. Запустим монитор порта (раздел основного меню «Инструменты->Монитор порта»).

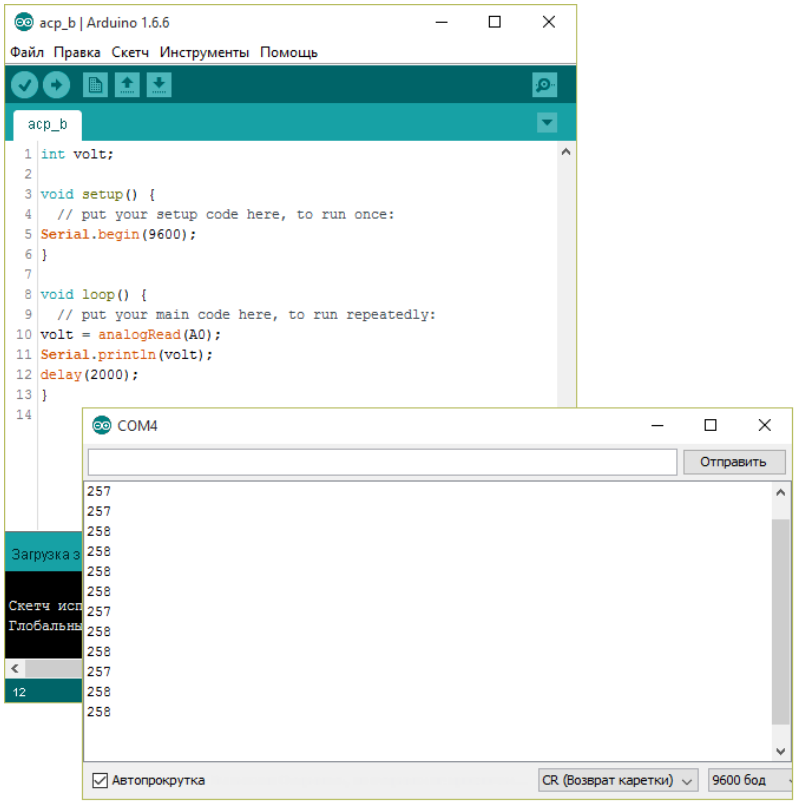


Рис. 4.2. Чтение напряжения с помощью модуля последовательного вывода данных

Монитор порта подключён к виртуальному порту, который был создан модулем Arduino, а добавленная нами команда Serial.println(volt) выводит значение переменной *volt* на экран монитора этого виртуального порта. Сама команда, как вы видите, достаточно понятна: Serial – последовательный; print – печатать, выводить, а ln, добавленное в конце, означает перевод каретки (как у пишущей машинки, откуда это всё и пошло). Пауза в конце нужна, чтобы показания не менялись слишком часто. Кстати, вход чувствителен к наводкам, проводя эксперимент, не пытайтесь держать концы проводов пальцами, впрочем, попробуйте…

Мы увидели десятичное число 258. Измеряли мы напряжение порядка 1.5 В. Но вспомним, что АЦП работает, увеличивая напряжение для сравнения ступеньками. Десять бит позволяют получить число 1023, которое будет соответствовать напряжению 5 В. Одна «ступенька» измерения получится делением полного напряжения на число, определяемое десятью битами, а умножив это значение на то, что мы видим, мы получим: (5/1023) * 258 = 1.26(В). Так и есть. Батарейка плохая, но это же значение даёт и мультиметр.

Если вам помимо мультиметра понадобится вольтметр для каких-то экспериментов, вы можете слегка подправить программу, внеся вычисления, и получить такой вид результата измерений:

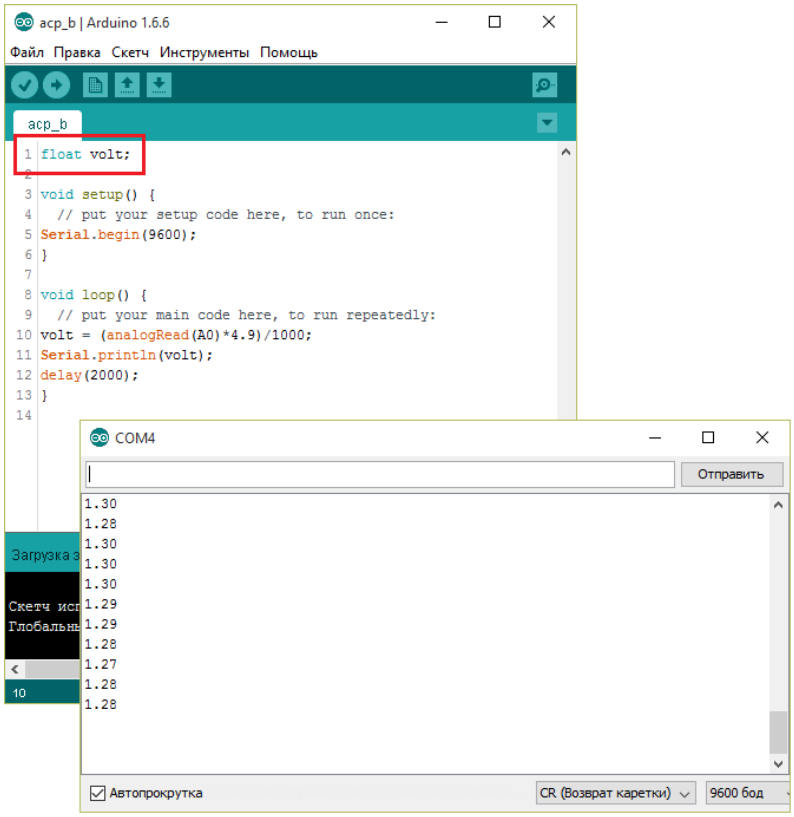


Рис. 4.3. Получение значения напряжения

Прежде, чем перейти к дальнейшим экспериментам, следует иметь в виду, что модуль Arduino имеет свои особенности. Хотя микроконтроллер, используемый в этом модуле, общего назначения, но…

> Цифровые вводы-выводы 0 и 1 модуля Arduino используются при последовательном обмене данными, и могут быть связаны с программатором Arduino. Поэтому пока лучше не трогать эти выводы модуля Arduino,.

С учётом этой особенности модуля Arduino остановимся на выводе двух значащих цифр считываемого напряжения на два индикатора через дешифраторы.

Начнём мы с того, что, как и раньше, прочитанное число (258 на рисунке 4.2) мы преобразуем в милливольты, умножив его на 5. Почему на 5? Потому, что мы будем работать с целыми числами. Такой подход даёт ошибку. Так максимальное число 1023, умноженное на 5, даст не пять вольт, а больше. Но ошибка не столь велика, чтобы отказаться от этого простого решения.

Итак, мы умножим прочитанное в переменную *volt* значение на 5: 258*5 = 1290. Это значение напряжения в милливольтах. Но нам нужны только две значащие цифры. Поэтому мы разделим полученное число на 100. Итогом станет число 12. Это число состоит из двух цифр, единицы и двойки. Именно эти цифры мы и должны вывести на индикаторы.

Первое, что мы должны сделать теперь в программе – это разделить число на две цифры, которые мы и будем выводить на индикаторы. Чтобы сохранить полученные цифры, мы добавим две переменные в программу типа byte, пусть первая называется *first*, а вторая *second*. Их мы добавим перед первым блоком программы, записав это так:

```
byte first = 0;
byte second = 0;
```

Обратите внимание, что знак равенства в данном случае имеет в языке программирования иной смысл, чем в арифметике. Это не равенство, а *операция присваивания*! В других языках программирования присваивание может иметь другое написание, например, в Паскале это записывается как «:=». И ещё, мы присвоили значение ноль этим переменным. Каждая переменная должна иметь начальное значение. Часто компиляторы присваивают им значение по умолчанию равное нулю, если вы не сделали этого. Но лучше взять за правило самостоятельно присваивать начальные значения переменным.

Чтобы получить первую цифру, мы разделим прочитанное в переменную volt и преобразованное нами ранее значение на десять: 12/10 = 1. Теперь мы можем присвоить это значение переменной *first*: first = volt/10.

Чтобы получить вторую цифру, мы вначале умножим первую цифру на десять, а затем вычтем из переменной *volt* это значение: second = volt – first*10. В программе эти операции мы добавим в основной блок и, как и ранее, проверим, что у нас получается, выводя две цифры на монитор последовательного порта.

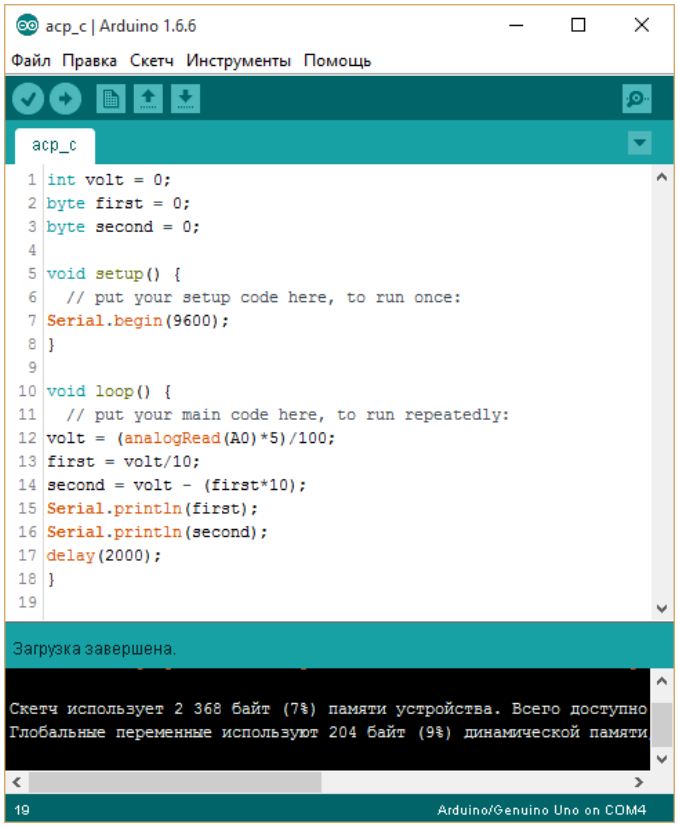


Рис. 4.4. Программа с разделением числа на две значащие цифры

Нам осталось вывести две цифры на два индикатора. До этого мы пользовались тем, что назначали один из выводов на выход и управляли его состоянием. Сейчас нам хотелось бы вывести всё значение сразу. И приходится, увы, учитывать особенности подключения микроконтроллера. На сайте Arduino можно найти схему подключения контроллера к выводам модуля.

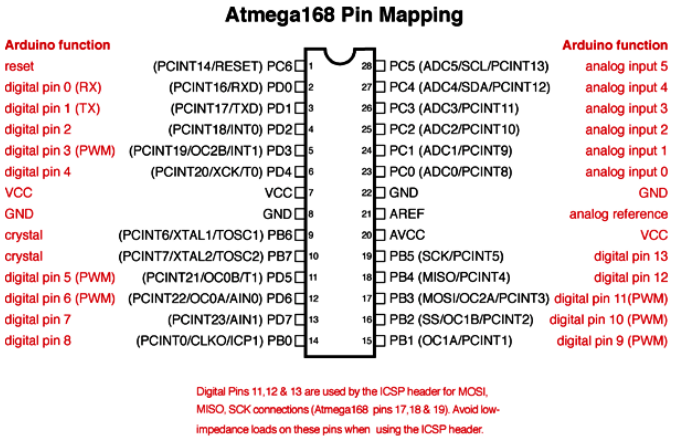


Рис. 4.5. Карта подключения микроконтроллера к выводам модуля

Пусть вас не смущает, что модель не ATmega328, как у модуля Arduino UNO, схема одинакова для двух моделей контроллера. Без возможности использовать порт целиком, мы будем использовать старшие биты порта D, которые подключены к выводам pin7-pin4, для одной цифры, а для другой цифры младшие биты порта B, выводы pin8-pin11. Чтобы назначить эти выводы на выход, следует в блоке настроек записать: DDRD = DDRD | B11110000; DDRB = DDRB | B00001111;

Поскольку мы используем только старшие биты порта D, нам нужно сдвинуть четыре бита переменной first влево. Для этого есть операция сдвига влево, что мы запишем так:

```
first = first<<4;
```

После этого мы можем присваивать значения цифр портам вывода. Программа приобретает следующий вид:

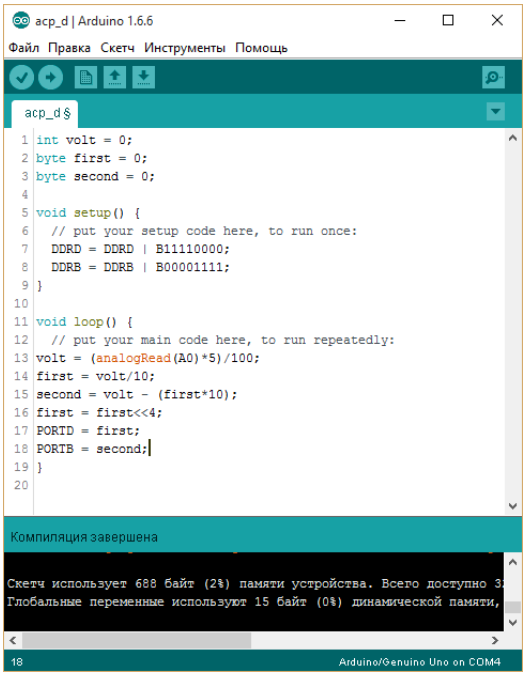


Рис. 4.6. Вид программы с выводом чисел в порты

Я ещё раз запишу эту программу, если вы захотите скопировать её:

```
int volt = 0;
byte first = 0;
byte second = 0;

void setup() {
  // put your setup code here, to run once:
  DDRD = DDRD | B11110000;
  DDRB = DDRB | B00001111;
}

void loop() {
  // put your main code here, to run repeatedly:
volt = (analogRead(A0)*5)/100;
first = volt/10;
second = volt - (first*10);
first = first<<4;
PORTD = first;
PORTB = second;
}
```

Я предпочитаю проверить эту программу с помощью моделирования в другой программе:

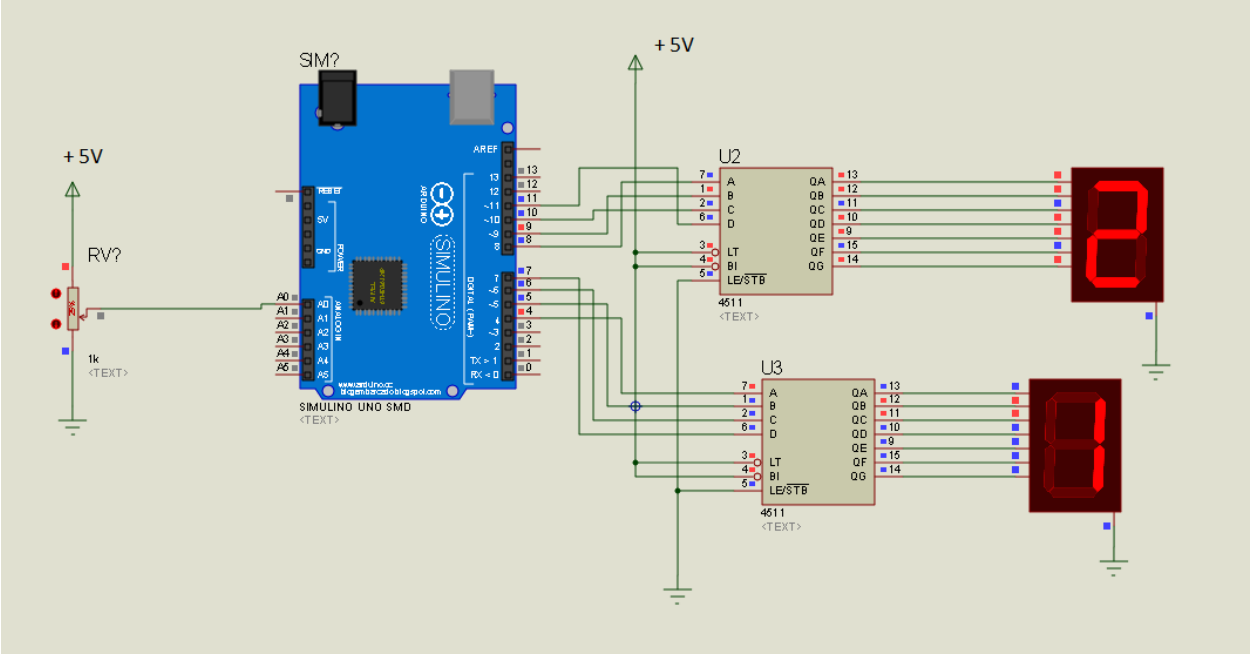


Рис. 4.7. Моделирование программы

Если вам захочется выполнить проверку с реальными компонентами, то может возникнуть вопрос о выборе резисторов, которые следует подключить к каждому из сегментов индикаторов. Можно ориентироваться на значение сопротивления порядка 470 Ом. Как определить это? В справочных данных на индикатор указывают номинальный или максимальный ток для каждого из сегментов. Если указан максимальный ток, возьмите среднее значение, положим, 10 мА. Напряжение на выводе дешифратора для каждого из сегментов 5 В. Вам нужно измерить падение напряжения на сегменте, взяв достаточно большое сопротивление, например, 1 кОм:

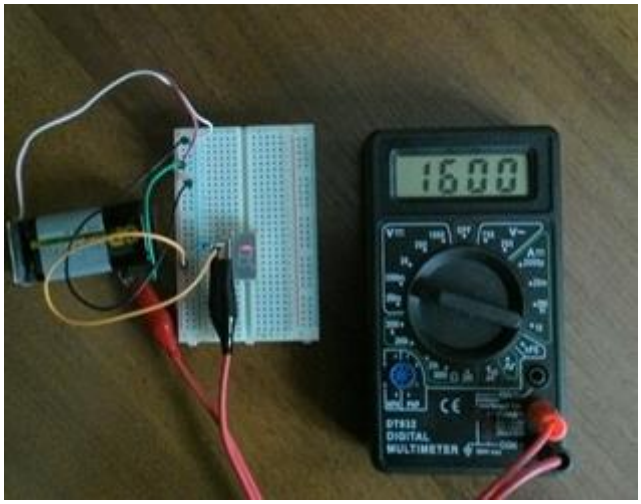


Рис. 4.8. Измерение падения напряжения на сегменте индикатора

Вычитая полученное значение (1.6 В) из 5 вольт (3.4 В), вы получаете возможность разделить это напряжение на оптимальный ток (0.01 А), чтобы получить сопротивление: 3.4/0.01 = 340 Ом.

Глава 5. Среда программирования

Проект Arduino содержит всё необходимое для изучения работы с микроконтроллером: язык программирования, среда разработки, линейка модулей с разными моделями микроконтроллеров, примеры и многочисленные библиотеки функций. Кроме этого есть дополнительные платы, существенно расширяющие возможности создания разнообразных устройств.

Вы можете сделать первые шаги, повторяя простые примеры и используя недорогие внешние компоненты. По мере овладения языком программирования, по мере того, как вы познакомитесь с разнообразными дополнительными компонентами, вы сможете расширить круг интересов, например, сменив среду разработки. Это касается и реализации устройств. Проведя первоначальные опыты с модулем Arduino, вы можете изготовить устройство, используя непосредственно микроконтроллер.

Основная линейка модулей Arduino выполнена на микроконтроллерах Atmel. Но производитель предлагает и свою среду разработки для выпускаемых контроллеров. Вот пример того, как выглядит Atmel Studio 7:

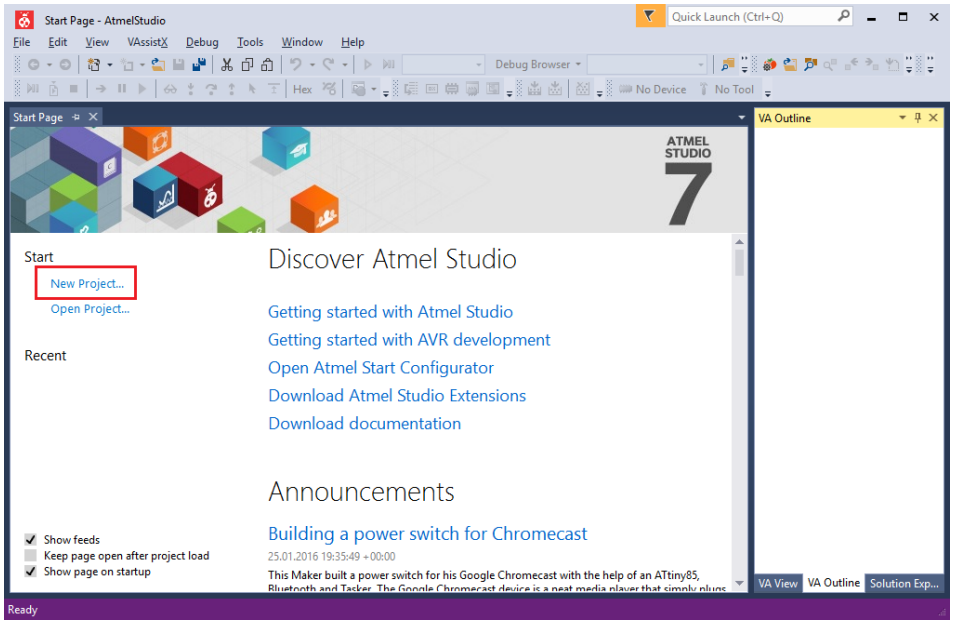


Рис. 5.1. Запуск программы Atmel Studio 7

Создавая новый проект, вы должны быть готовы к тому, что вам придётся изучить язык C/C++ и освоить компилятор, используемый программой Atmel Studio.

При создании нового проекта вы должны указать его имя и место, где будет расположена папка с проектом, а также указать модель микроконтроллера. После чего вы получите шаблон для написания собственной программы.

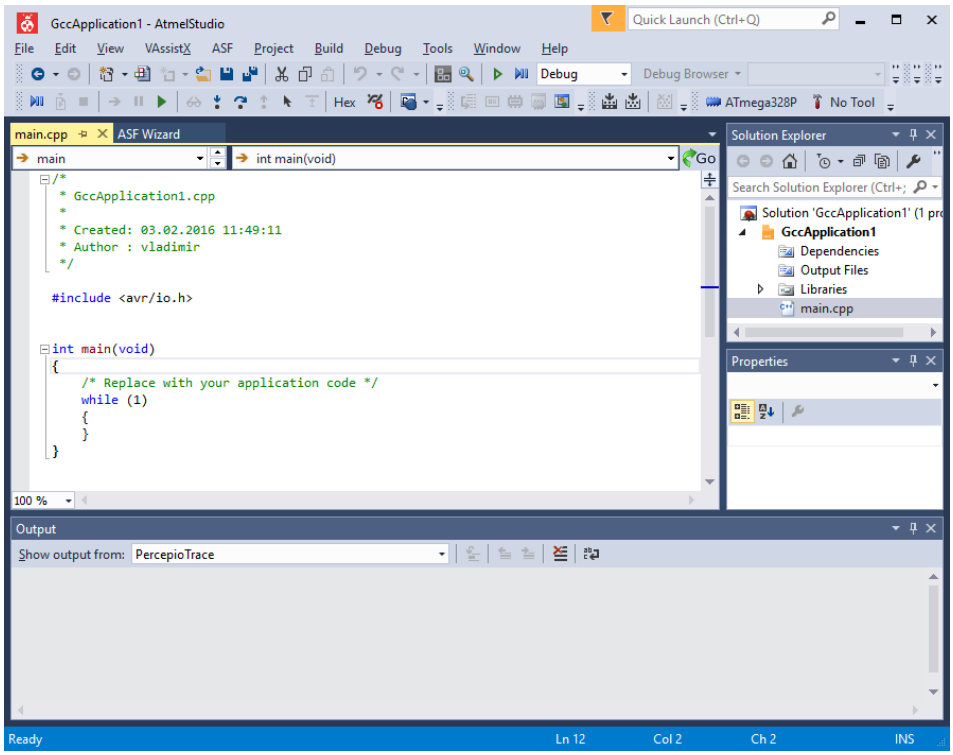


Рис. 5.2. Заготовка под программу в Atmel Studio

Облегчить эту работу, облегчить работу по освоению языка Си поможет, если у вас есть такая возможность, другая среда разработки Flowcode for AVR.

При запуске программы вам следует выбрать вашу модель контроллера, после чего вы получаете шаблон программы, использующий графический язык программирования.

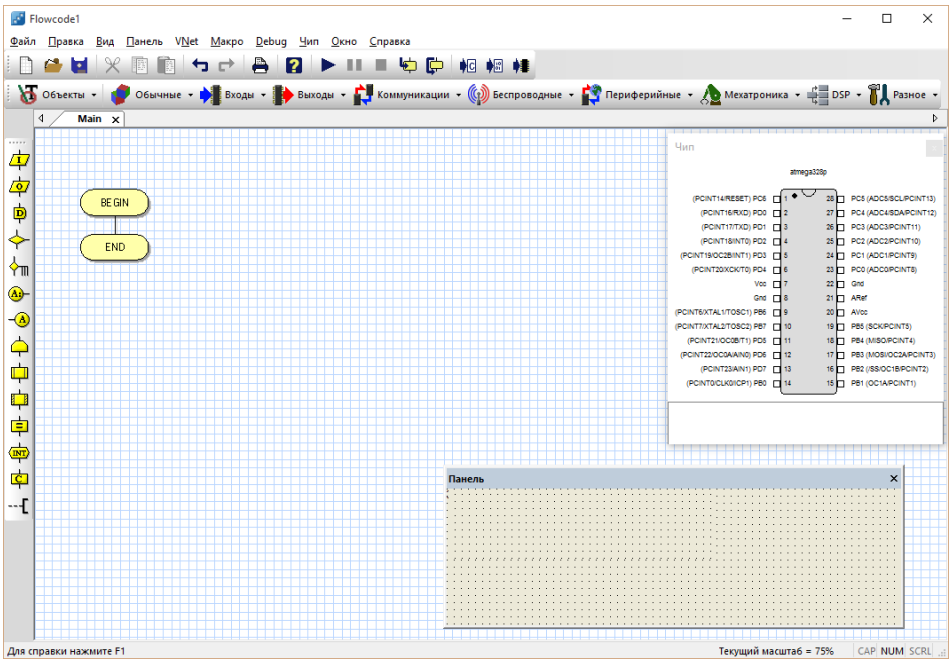


Рис. 5.3. Заготовка программы в Flowcode

Повторим первые маленькие шаги, сделанные ранее в программе Arduino. Для этого с левой инструментальной панели перетащим графический эквивалент программного вывода, который разместим между BEGIN и END (программные скобки). Выяснив, что вывод 13 модуля Arduino – это вывод PB5 микроконтроллера, настроим выход на этот вывод и «зажжём» светодиод.

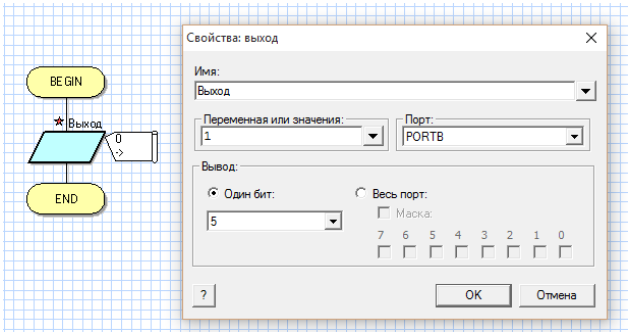


Рис. 5.4. Включение светодиода на выводе 13 модуля Arduino

Сохранив проект, можно откомпилировать его в программу на языке Си. Конечно, можно откомпилировать программу и в конечный hex-файл, можно даже, настроив программатор, загрузить программу в модуль Arduino. Но я хотел рассказать о другом. Мы создали новый проект в программе Atmel Studio. Мы получили текст программы на языке Си в программе Flowcode. Вставим этот текст программы в редактор Atmel Studio и запустим компиляцию (Build).

Мы получим много предупреждений, это так, но программа будет откомпилирована, то есть, вы получите hex-файл для загрузки в микроконтроллер.

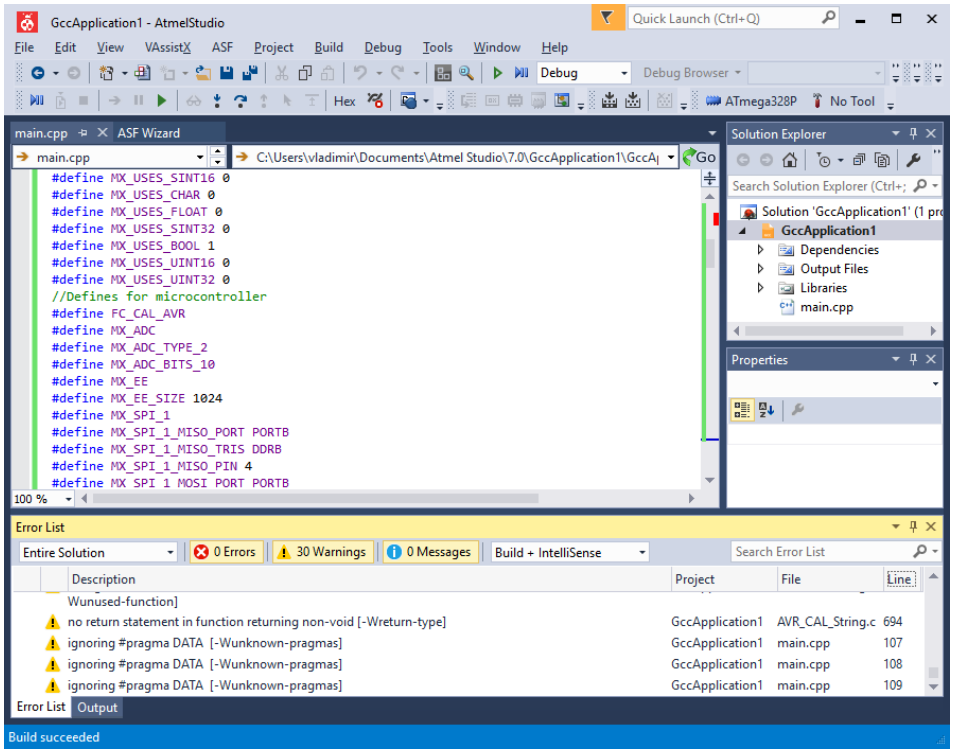


Рис. 5.5. Замена шаблона текстом программы

Для загрузки программы в микросхему вам понадобится программатор. Его можно купить, его можно сделать самостоятельно, вы сможете даже загрузить программу в модуль Arduino, но…

Если вас привлекает подобная возможность, то это означает, что вы перестали быть «самым начинающим». Что ж, в добрый путь!