Подключение микроконтроллера. Ликбез.

Казалось бы простая тема, а однако в комментах меня завалили вопросами как подключить микроконтроллер. Как подключить к нему светодиод, кнопку, питание. Что делать с **AGND** или **AREF**. Зачем нужен **AVCC** и все в таком духе. Итак, раз есть вопросы, значит тема не понятна и надо дать по возможности исчерпывающий ответ. Все описываю для контроллеров AVR, но для каких нибудь PIC все очень и очень похоже. Т.к. принципы тут едины.

Чтобы понимать ряд терминов активно упоминающихся в этой статье, надо сначала прочитать статью про порты ввода-вывода.

Питание

Для работы микроконтроллеру нужна энергия — электричество. Для этого на него естественно нужно завести питалово. Напряжение питание у МК **Atmel AVR** разнится от **1.8** до **5** вольт, в зависимости от серии и модели. Все **AVR** могут работать от 5 вольт (если есть чисто низковольтные серии, то просьба уточнить в комментах, т.к. я таких не встречал). Так что будем считать что напряжение питания контроллера у нас всегда 5 вольт или около того. Плюс напряжения питания обычно обозначается как **Vcc**. Нулевой вывод (а также Земля, Корпус, да как только его не называют) обозначают **GND**. Если взять за пример комповый блок питания. То черный провод это GND (кстати, земляной провод традиционно окрашивают в черный цвет), а красный это +5, будет нашим **Vcc**. Если ты собираешься запитать микроконтроллер от батареек, то минус батареек примем за **GND**, а плюс за **Vcc** (главное чтобы напряжение питания с батарей было в заданных пределах для данного МК, позырь в даташите. Параметр обычно написан на первой странице в общем описании фич:

• Operating Voltages
–1.8 - 5.5V (ATtiny2313V)
–2.7 - 5.5V (ATtiny2313)
• Speed Grades
–ATtiny2313V: 0 - 4 MHz @ 1.8 - 5.5V, 0 - 10 MHz @ 2.7 - 5.5V
–ATtiny2313: 0 - 10 MHz @ 2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V

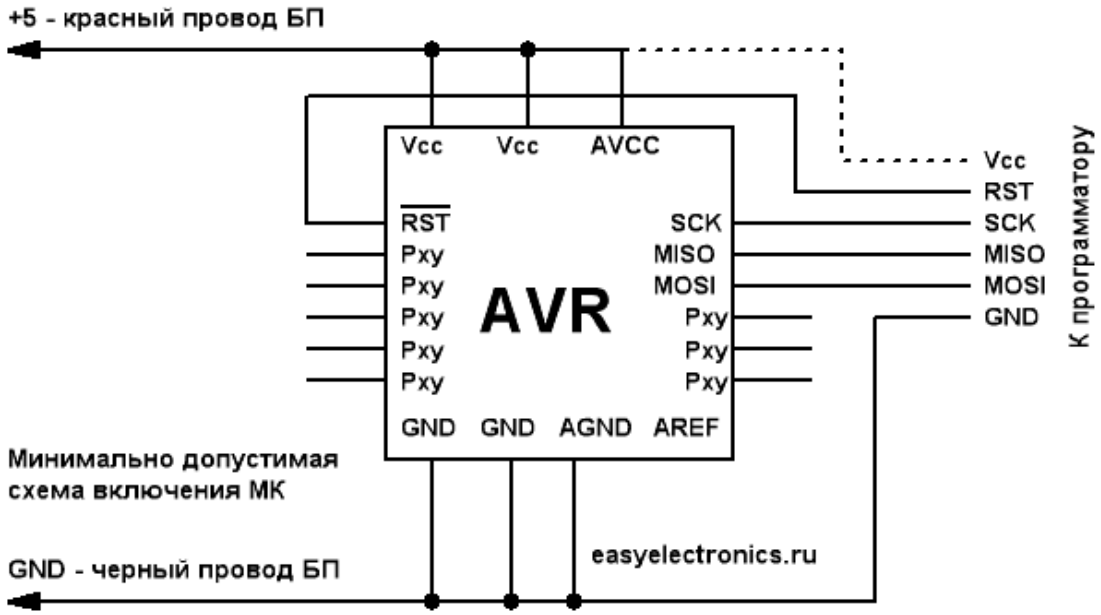
Обрати внимание, что есть особые низковольтные серии (например 2313V низковльтная) у которых нижня граница напряжения питания сильно меньше. Также стоит обратить внимание на следующий пункт, про частоты. Тут показана зависимость максимальной частоты от напряжения питания. Видно, что на низком напряжении предельные частоты ниже. А низковольтные серии раза в два медленней своих высоковольтных коллег. Впрочем, разгону все процессоры покорны ;)))))

Для работы контроллерам серии **AVR** достаточно только питания. На все входы **Vcc** надо подать наши 5 (или сколько там у тебя) вольт, а все входы **GND** надо посадить на землю. У микроконтроллера может быть много входов **Vcc** и много входов **GND** (особенно если он в квадратном **TQFP** корпусе. У которого питалово со всех сторон торчит). Много выводов сделано не для удобства монтажа, а с целью равномерной запитки кристалла со всех сторон, чтобы внутренние цепи питания не перегружались. А то представь, что подключил ты питалово только с одной стороны, а с другой стороны чипа навесил на каждую линию порта по светодиоду, да разом их зажег. Внутренняя тонкопленочная шина питания, офигев от такой токовой нагрузки, испарилась и проц взял ВНЕЗАПНО и без видимых, казалось бы, причин отбросил копыта. Так что**ПОДКЛЮЧАТЬ НАДО ВСЕ ВЫВОДЫ Vcc и GND**. Соединить их соответственно и запитать.

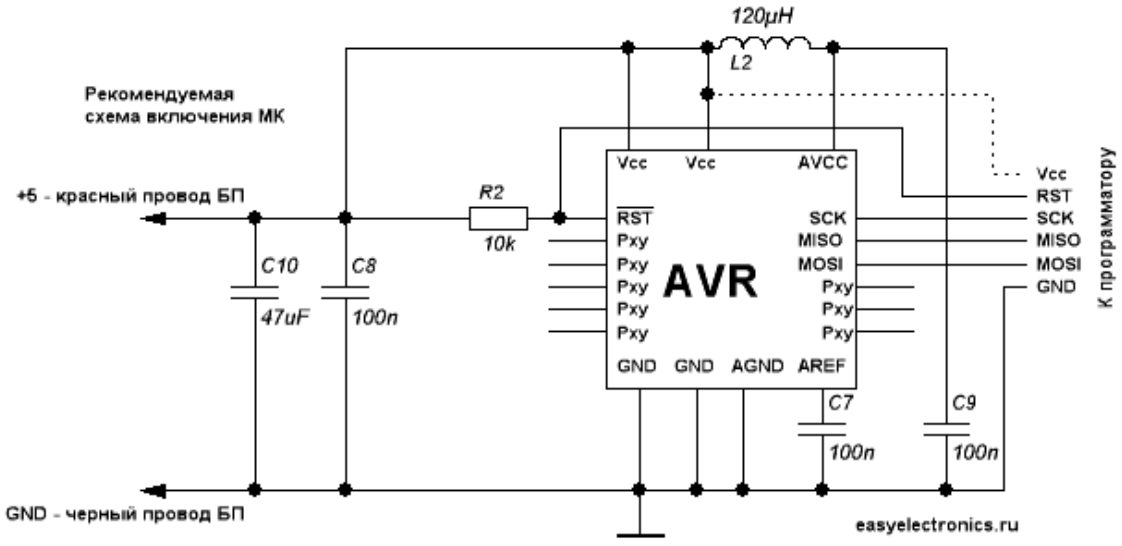
Отдельные вопросы вызвают **AGND** и **AVCC** — это аналоговая земля и питание для Аналого-Цифрового Преобразователя. АЦП это очень точный измеритель напряжения, поэтому его желательно запитать через дополнительные фильтры, чтобы помехи, которые не редки в обычной питающей цепи, не влияли на качество измерения. С этой целью в точных схемах проводят разделение земли на цифровую и аналоговую (они соединены должны быть только в одной точке), а на **AVCC** подается напряжение через фильтрующий дроссель. Если ты не планируешь использовать АЦП или не собираешься делать точные измерения, то вполне допустимо на **AVCC** подать те же 5 вольт, что и на **Vcc**, а **AGND** посадить на ту же землю что и все.**Но подключать их надо обязательно!!!** ЕМНИП от AVCC питается также порт А.

Warning!!!
В чипе Mega8 похоже есть ошибка на уровне топологии чипа — Vcc и AVcc связаны между собой внутри кристалла. Между ними сопротивление около (!!!) 5Ом Для сравнения, в ATmega16 и ATmega168 между Vcc и AVcc сопротивление в десятки МЕГА ом! В даташите на этот счет никаких указаний нет до сих пор, но в одном из топиков за 2004 год на AVRFreaksсказано, что люди бодались с цифровым шумом АЦП, потом написали в поддержку Atmel мол WTF??? А те, дескать, да в чипе есть бага и Vcc и AVcc соединены внутри кристалла. В свете этой инфы, думаю что ставить дроссель на AVcc для Mega8 практически бесполезно. Но AVcc запитывать надо в любом случае — кто знает насколько мощная эта внутренняя связь?

Простейшая схема подключения Микроконтроллера AVR приведена ниже:



Это необходимый минимум чтобы контроллер запустился. Провод **Vcc** до программатора показан пунктиром поскольку он не обязателен. Если ты собираешься питать МК от внешнего источника, то он не нужен. Но я все же рекомендую для начала питать всю систему (МК+программатор) от одного источника — больше вероятность успешной прошивки :) Для учебной цели, диодиком помигать, сойдет и так.
Но настолько все упрощать я не рекомендую. Лучше сразу добавить парочку навесных внешних элементов. Правильней будет. Чтобы было вот так:



Как видишь, добавился дроссель в цепь питания **AVCC**, а также конденсаторы. Хорошим тоном является ставить керамический конденсатор на сотню нанофарад между **Vcc** и **GND** у каждой микросхемы (а если у

микрухи много вход питания и земель, то между каждым питанием и каждой землей) как можно ближе к выводам питания — он сгладит краткие импульсные помехи в шине питания вызыванные работой цифровых схем. Конденсатор на 47мКФ в цепи питания сгладит более глубокие броски напряжения. Кондесатор между**AVcc** и **GND** дополнительно успокоит питание на **АЦП**.
Вход **AREF** это вход опорного напряжения **АЦП**. Туда вообще можно подать напряжение относительно которого будет считать **АЦП**, но обычно используется либо внутренний источник опорного напряжения на 2.56 вольта, либо напряжение на **AVCC**, поэтому на **AREF** рекомендуется вешать конденсатор, что немного улучшит качество опорного напряжения **АЦП** (а от качества опоры зависит адекватность показаний на выходе **АЦП**).

Схема сброса

Резистор на **RESET**. Вообще в **AVR** есть своя внутренняя схема сброса, а сигнал **RESET** изнутри уже подтянут резистором в 100кОм к **Vcc**. НО! Подтяжка это настолько дохлая, что микроконтроллер ловит сброс от каждого чиха. Например, от касания пальцем ножки **RST**, а то и просто от задевания пальцем за плату. Поэтому крайне рекомендуется **RST** подтянуть до питания резистором в 10к. Меньше не стоит, т.к. тогда есть вероятность, что внутрисхемный программатор не сможет эту подтяжку пересилить и прошить МК внутри схемы не удасться. 10к в самый раз.
Есть еще вот такая схема сброса:

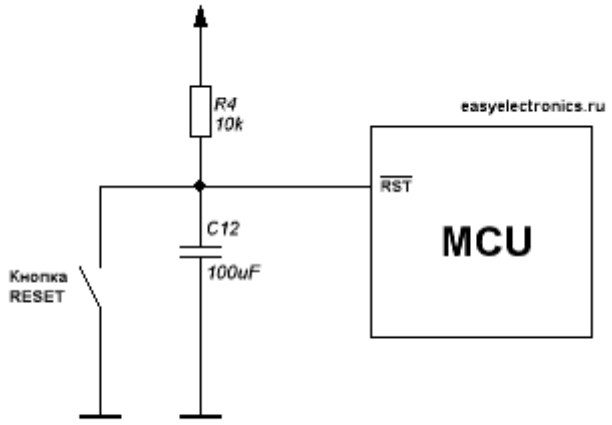
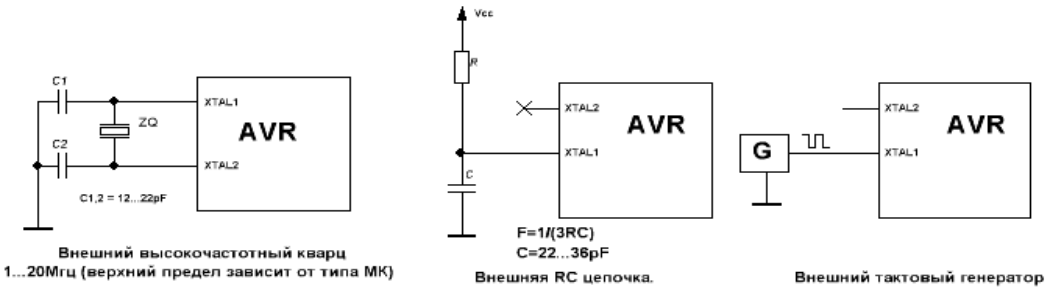


Схема надежного сброса МК

Она замечательна чем — при включении схемы конденсатор разряжен и напряжение на **RST** близко к нулю — микроконтроллер не стартует, т.к. ему непрерывный сброс. Но со временем, через резистор, конденсатор зарядится и напряжение на **RST** достигнет лог1 — МК запустится. Ну, а кнопка позволяет принудительно сделать сброс если надо.
Задержка будет примерно T=R*C для данного примера — около секунды. Зачем эта задержка? Да хотя бы для того, чтобы МК не стартовал раньше чем все девайсы платы запитаются и выйдут на установившийся режим. В старых МК (**AT89C51**, например) без такой цепочки, обеспечивающей начальный сброс, МК мог вообще не стартануть.
В принципе, в **AVR** задержку старта, если нужно, можно сделать программно — потупить с пол секунды прежде чем приступать к активным действиям. Так что кондер можно выкинуть нафиг. А кнопку… как хочешь. Нужен тебе внешний **RESET**? Тогда оставь. Я обычно оставляю.

Источник тактового сигнала

Тактовый генератор это сердце микроконтроллера. По каждому импульсу происходит какая нибудь операция внутри контроллера — гоняют данные по регистрам и шинам, переключаются выводы портов, щелкают таймеры. Чем быстрей тактовая частота тем шустрей МК выполняет свои действия и больше жрет энергии (на переключения логических вентилей нужна энергия, чем чаще они переключаются тем больше энергии надо).
Импульсы задаются тактовым генератором встроенным в микроконтроллер. Впрочем может быть и внешний генератор, все очень гибко конфигурируется! Скорость с которой тикает внутренний генератор зависит от настроек микроконтроллера и обвязки.



Внешний высокочастотный кварц 1...20Мгц (верхний предел зависит от типа МК)

Внешняя RC цепочка.

Внешний тактовый генератор

Генератор может быть:

- Внутренним с внутренней задающей RC цепочкой.
  В таком случае никакой обвязки не требуется вообще! А выводы XTAL1 и XTAL2 можно не подключать вовсе, либо использовать их как обычные порты ввода вывода (если МК это позволяет). Обычно можно выбрать одно из 4х значений внутренней частоты. **Этот режим установлен по дефолту**.
- Внутренним с внешней задающей RC цепочкой.
  Тут потребуется подключить снаружи микроконтроллера конденсатор и резистор. Позволяет менять на ходу тактовую частоту, просто подстраивая значение резистора.
- Внутренним с внешним задающим кварцем.
  Снаружи ставится кварцевый резонатор и пара конденсаторов. Если кварц взят низкочастотный (до 1МГц) то конденсаторы не ставят.
- Внешним.
  С какого либо другого устройства идет прямоугольный сигнал на вход МК, который и задает такты. Полезен этот режим, например, если надо чтобы у нас несколько микроконтроллеров работали в жестком синхронизме от одного генератора.

У разных схем есть разные достоинства:

В случае **внутренней RC цепи** мы экономим место на плате, нам не нужно дополнительных деталек, но мы не можем развить максимальную частоту и частота немного зависит от температуры, может плавать.

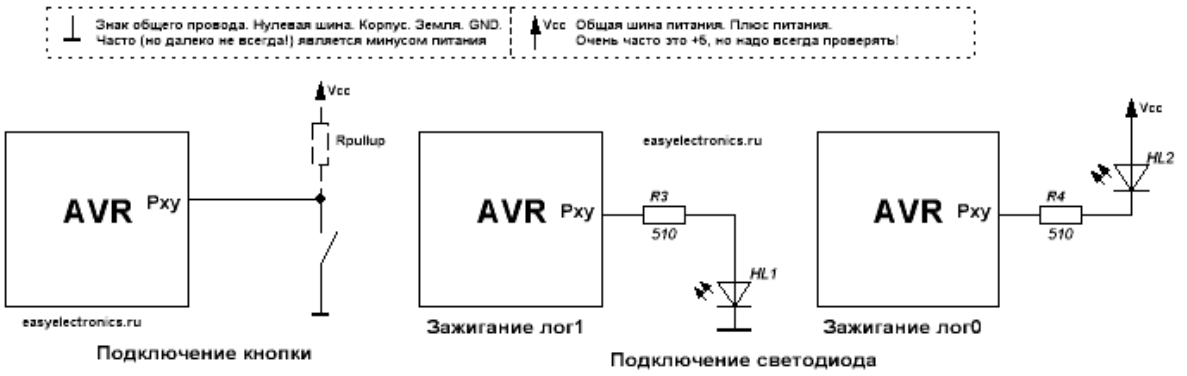
У внешнего кварца отличные показатели точности, но он стоит лишних 15 рублей и требует дополнительных деталей и, что самое обидное, часто съедает пару ног I/O. Также на внешнем же кварце можно добиться максимальной производительности от МК. Частота МК определяется частотой на которую заточен выбранный кварц. **Внешная RC цепь** позволяет тикать генератору МК быстрей чем от внутренней, стоит дешевле кварца, но имеет те же проблемы со стабильностью частоты, что и внутренняя RC цепь.

Способы тактования МК описаны в даташите в разделе **System Clock and Clock Options** и всецело определяются конфигурацией **Fuse Bit's**. Пока же я настоятельно рекомендую **НЕ ТРОГАТЬ FUSE** пока ты не будешь твердо знать что ты делаешь и зачем. Т.к. выставив что нибудь не то, можно очень быстро превратить МК в кусок бесполезного кремния, вернуть к жизни который будет уже очень непросто (но возможно!)

Подключение к микроконтроллеру светодиода и кнопки

Сам по себе, без взаимодействия с внешним миром, микроконтроллер не интересен — кому интересно что он там внутри себя тикает? А вот если можно как то это отобразить или на это повлиять…

Итак, кнопка и светодиод подключаются следующим образом:



Для кнопки надо выбраную ножку I/O подключить через кнопку на землю. Сам же вывод надо сконфигурировать как **вход с подтяжкой** (DDRxy=0 PORTxy=1). Тогда, когда кнопка не нажата, через подтягивающий резистор, на входе будет высокий уровень напряжения, а из бит **PINxy** будет при чтении

отдавать 1. Если кнопку нажать, то вход будет положен на землю, а напряжение на нем упадет до нуля, а значит из **PINxy** будет читаться 0. По нулям в битах регистра **PINx** мы узнаем что кнопки нажаты. Пунктиром показан дополнительный подтягивающий резистор. Несмотря на то, что внутри AVR на порт можно подключить подтяжку, она слабоватая — 100кОм. А значит ее легко придавить к земле помехой или наводкой, что вызовет ложное срабатывание. А еще эти внутренние подтягивающие резисторы очень любят гореть от наводок. У меня уже с десяток микроконтроллеров с убитыми PullUp резисторами. Все работает, но только нет подтяжки — сгорела. Вешаешь снаружи резистор и работает как ни в чем ни бывало. Поэтому, для ответственных схем я настоятельно рекомендую добавить внешнюю подтяжку на 10кОм — даже если внутреннюю накроет, внешняя послужит. В процессе обучения на это можно забить.

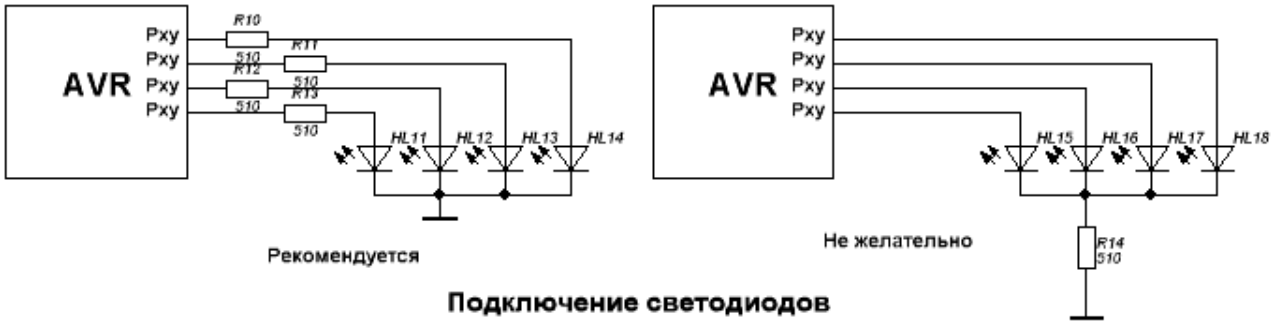
Светодиод подключается на порт двумя способами. По схеме **Порт-земля** или **Порт-Питание**. В первом случае для зажигания диода надо выдать в порт лог1 — высокий уровень (примерно равен Vcc). Во втором случае для зажжения диода требуется выдать в порт лог0 — низкий уровень (около нуля). Для **AVR** разницы вроде бы нет, а вот многие старые серии микроконтроллеров вниз тянули куда лучше чем вверх, так что схема Порт-Питание распространена чаще. Я применяю и ту и другую схему исходя из удобства разводки печатной платы. Ну, а на программном уровне разницы особой нет.

Вывод порта для работы со светодиодом надо сконфигурировать на **выход** (DDRxy=1) и тогда в зависимости от значения в PORTxy на ножке будет либо высокий либо низкий уровень напряжения.

Светодиод **надо подключать через резистор**. Дело в том, что прямое сопротивление светодиода очень мало. И если не ограничивать ток через него, то он просто напросто может сгореть нафиг. Либо, что вероятней, пожечь вывод микроконтроллера, который, к слову, может тянуть что то около 20-30мА. А для нормального свечения обычному светодиоду (всякие термоядерные ультраяркие прожектора мы не рассматриваем сейчас, эти монстры могут и ампер сожрать) надо около 3…15мА.

Так что, на вскидку, считаем:

- Напряжение на выходе ноги МК около 5 вольт, падение напряжени на светодиоде обычно около 2.5 вольт (выше нельзя, иначе диод сожрет тока больше чем надо и подавится, испустив красивый дым)
- Таким образом, напряжение которое должен взять на себя ограничительный резистор будет 5-2.5 = 2.5В.
- Ток нам нужен 5мА — нефига светодиод зря кормить, нам индикация нужна, а не освещение :)
- R=U/I= 2.5/5E-3 = 500Ом. Ближайший по ряду это 510 Ом. Вот его и возьмем. В принципе, можно ставить от 220 Ом до 680 Ом что под руку попадется — гореть будет нормально.



Подключение светодиодов

Если надо подключить много светодиодов, то на каждый мы вешаем по собственному резистору. Конечно, можно пожадничать и поставить на всех один резистор. Но тут будет западло — резистор то один, а диодов много! Соответственно чем больше диодов мы запалим тем меньше тока получит каждый — ток от одного резистора разделится между четырьмя. А поставить резистор поменьше нельзя — т.к. при зажигании одного диода он получит порцию тока на четверых и склеит ласты (либо пожгет порт).

Немного схемотехнических извратов или пара слов о экономии выводов

То что не удается запаять приходится программировать. (С) народная мудрость.

Очень часто бывает так, что вроде бы и памяти контроллера под задачу хватает с лихвой, и быстродействия через край, а ножек не хватает. Вот и приходится ставить избыточный и более дорогой микроконтроллер только потому, что у него банально больше выводов. Покажу парочку примеров как можно за счет усложнения программного кода сэкономить на железе.

Во главу угла такой экономии обычно ставится принцип динамического разделения назначения выводов во времени. То есть, например, вывод может работать на какую-либо шину, а когда шина не активна, то через

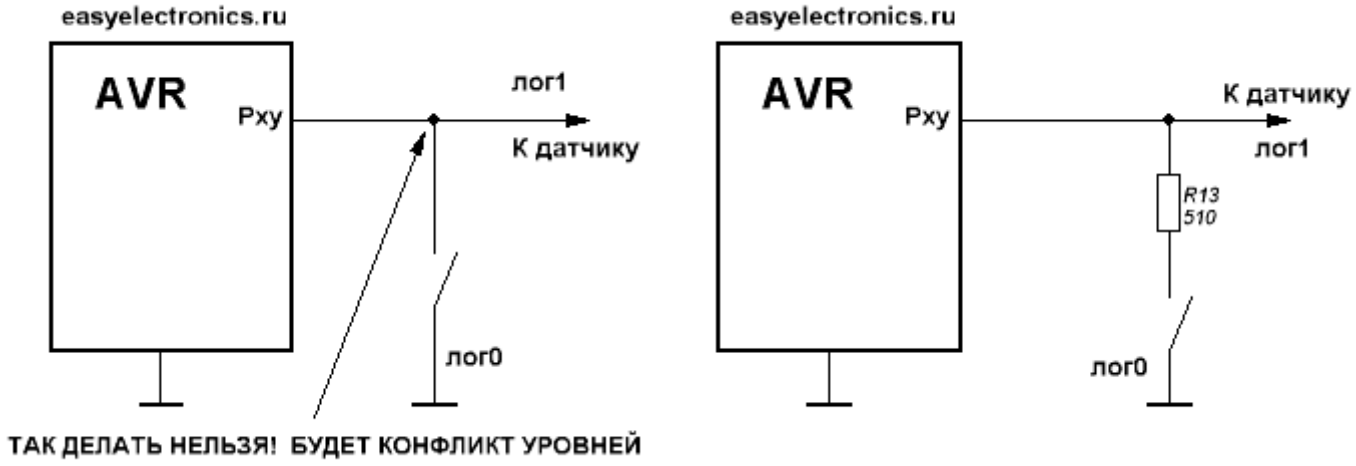
этот же вывод можно проверить состояние кнопки, или что нибудь передать по другой шине. Быстро (десятки или даже тысячи раз в секунду) переключаясь между двумя разными назначениями можно добиться эффекта “одновременной работы”.

Главное, тут следовать двум правилам:

- Два разных применения не должны мешать друг другу т.е. разделение во времени должно быть построено таким образом, чтобы смежная функция не искажала результат работы проверяемой функции.
- Ни в коем случае нельзя допускать конфликта уровней напряжений.

Приведу пример:

- У есть у нас вывод на который повешан выход с некого датчика и кнопка. Выход с датчика может быть 0, 1 в активном режиме и Hi-Z когда на датчик не приходит сигнал Enable.
- Кнопка же дает на линию жесткий 0, путем короткого замыкания.



Как это должно работать:

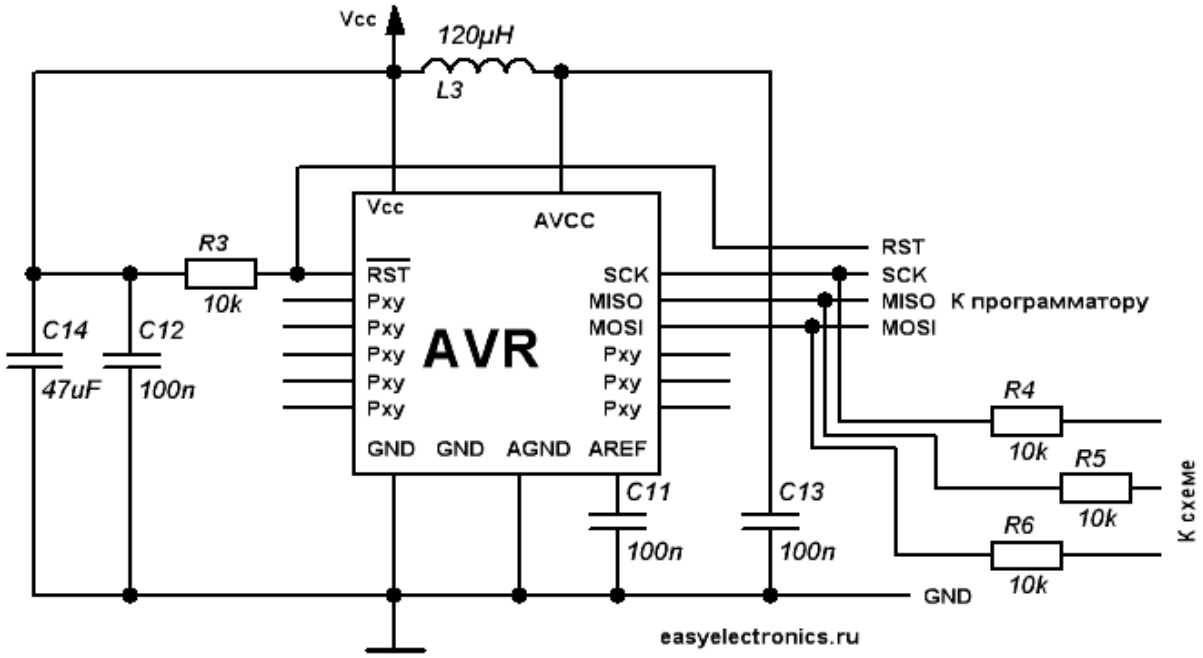
Скажем, основную часть времени у нас ввод микроконтроллера настроен на вход Hi-Z и мы снимаем показания с датчика на который подан еще и сигнал Enable. Когда нам надо опросить кнопку, то мы отбираем у датчика Enable и его выходы становятся в режим Hi-Z и нам не мешают. Вывод микроконтроллера мы переводим в режим Pull-Up и проверяем нет ли на входе нуля — сигнал нажатой кнопки. Проверили? Переводим вход МК в Hi-Z вход и подаем Enable на датчик снова. И так много раз в секунду.

Тут у нас возникает два противоречия:

- **Логическое противоречие**
  0 на линии может быть в двух случаях от датчика или от кнопки. Но в этом случае, пользуясь здравым смыслом и требуемым функционалом, мы логическое противоречие можем не брать во внимание. Просто будем знать, что нажатие кнопки искажает показания датчика, а значит когда датчик работает — мы кнопку жать не будем. А чтобы показания датчика не принять за нажатие кнопки мы, в тот момент когда ждем данные с датчика, просто не опрашиваем кнопку. От тупых действий, конечно, это не защитит. Но для упрощения примера защиту от дурака я сейчас во внимания не беру.
- **Электрическое противоречие**
  Если датчик выставит 1, а мы нажмем кнопку, то очевидно, что GND с Vcc в одном проводе не уживутся и кто нибудь умрет. В данном случае умрет выход датчика, как более слабый — куда там хилому транзистору тягаться с медной кнопкой.
  Организационными методами такое противоречие не решить — на глаз нельзя определить напряжение на линии и решить можно жать кнопку или нет. Да и в каком месте сейчас программа можно тоже только догадываться. Поэтому решать будем схемотехнически.
  Добавим резистор в цепь кнопки, резистор небольшой, рассчитывается исходя из максимального тока самого слабого вывода линии.
  Если у нас, например, вывод датчика может дать не более 10мА, то резистор нужен такой, чтобы ток через него от Vcc до GND не превышал этой величины. При питании 5 вольт это будет 510Ом. Теперь, даже если на линии со стороны датчика будет лог1, высокий уровень, то нажатие на кнопку не вызовет даже искажения логического уровня т.к. резистор рассчитан с учетом максимальной нагрузки порта

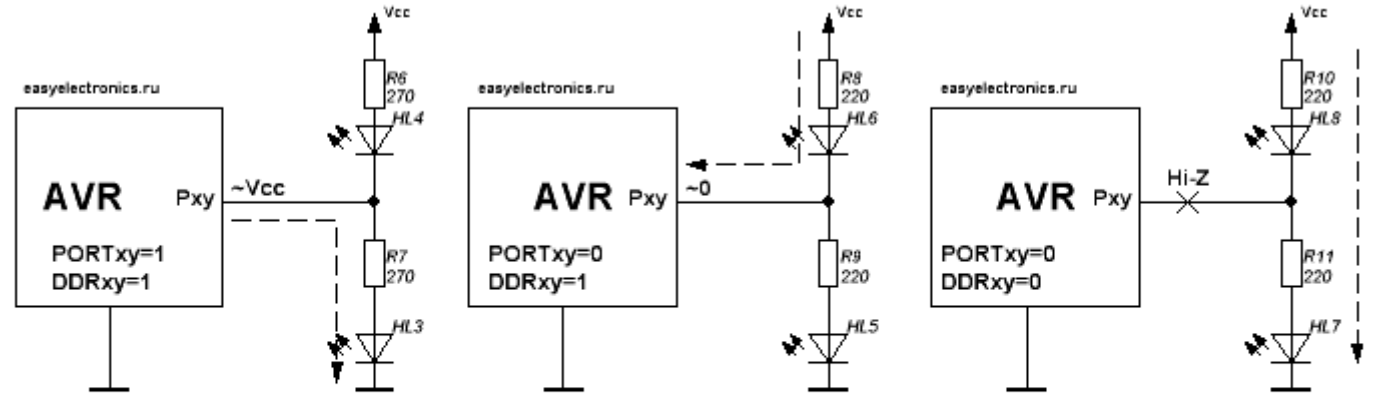
Пример получился немного сумбурный, но суть думаю понятна. Я хочу чтобы ты увидел и понял не только как делается, но и зачем это делается :)

Ну и несколько примеров нескольких функций на одной ноге:
Во-первых, **ISP разьем**. Я уже давным давно забыл что такое тыкать микроконтроллер вначале в колодку программатора, потом в плату, потом обратно и так по многу раз, пока прогу не отладишь. У меня на плате торчат 6 выводов ISP разьема и при отладке программатор вечно воткнут в плату, а программу я перешиваю порой по нескольку раз в 10 минут. Прошил — проверил. Не работает? Подправил, перепрошил еще раз… И так до тех пор пока не заработает. Ресурс у МК на перепрошивку исчисляется тысячами раз. Но ISP разьем сжирает выводы. Целых 3 штуки — **MOSI, MISO, SCK.**
В принципе, на эти выводы можно еще повесить и кнопки. В таком случае никто никому мешать не будет, главное во время прошивки не жать на эти кнопки. Также можно повесить и светодиоды (правда в этом случае простейший программатор Громова может дать сбой, а вот USBasp молодцом!) тогда при прошивке они будут очень жизнерадостно мерцать :)))
На линии под ISP можно повесить и что нибудь другое, главное, **чтобы при прошивке это ЧТОТО не начало ВНЕЗАПНО чудить**. Например, управление стокилограммовым манипулятором висит на линии ISP и во время прошивки на него пошла куча бредовых данных — так он может свихнуться и кому нибудь бошку разнести. Думать надо, в общем. А вот с каким нибудь LCD вроде HD44780, который работает по шинному интерфейсу прокатит такая схема:



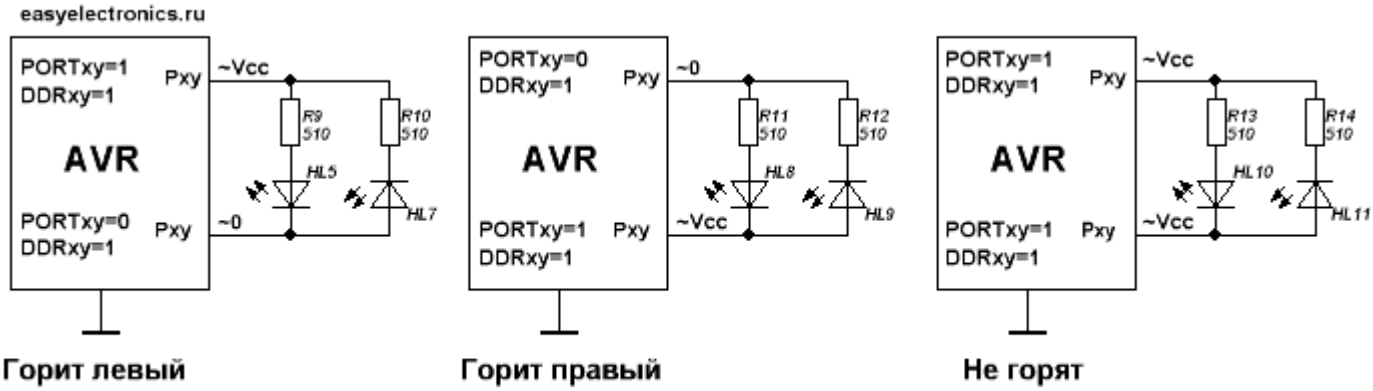
Согласование линий ISP и схемы

Резисторам в 10к отделяем линии программатора от основной схемы. В таком случае, даже если там будут какие либо другие логические уровни, то программатор их легко пересилит и спокойно прошьет микросхему. А при нормальной работе шины эти 10к резисторы особо влиять не будут.
Ножки можно зажать, например, на светодиодах:



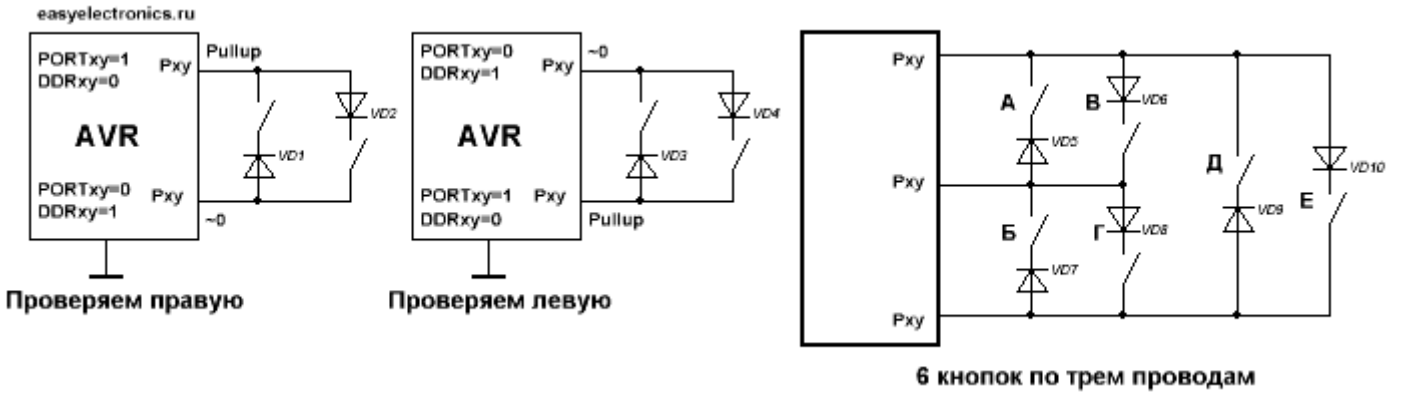
Два светодиода на один порт

Переключаем выход с 0 на 1 и зажигаем то верхний то нижний диод. Если надо зажечь оба, то мы просто переводим вывод микроконтроллера в режим **Hi-Z** и словно нет его, а диоды будут гореть сквозным током. Либо быстро быстро переключать диоды между собой, в этом случае на глаз они будут оба гореть. Недостаток схемы очевиден — диоды нельзя погасить. Но если по задумке хотя бы один должен гореть, то почему бы и нет? **UPD:** Тут подумал, а ведь можно подобрать светодиоды и резисторы так, чтобы их суммарное падение напряжения было на уровне напряжения питания, а суммарные резисторы в таком случае загонят ток в такой мизер, что когда нога в Hi-Z то диоды вообще гореть не будут. По крайней мере на глаз это будет не заметно совсем. Разве что в кромешной тьме.
Следующий вариант он не дает экономию ножек, зато позволяет упростить разводку печатной платы, не таща к двум диодам еще и шину питания или земли:

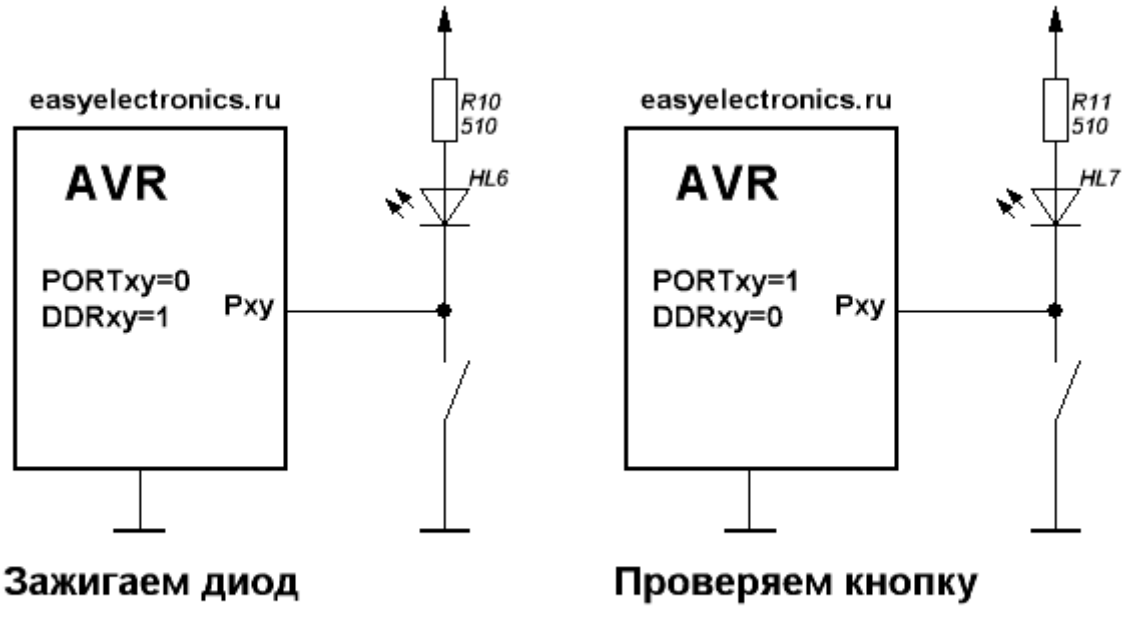


Подключение светодиодов по схеме ПОРТ-ПОРТ

Тут все просто — превращая один из выводов то в 0 то в 1 гоняем ток то в одну сторону то в другую. В результате горит то один то другой диод. Для погашения обоих — переводим ноги в какое то единое положение 11 или 00. Два диода сразу зажечь не получится, но можно сделать динамическую индикацию — если их быстро быстро переключать, то глаз не заметит подставы, для него они будут оба горящими. А добавив третью линию можно по трем ногам прогнать до 6 светодиодов на том же принципе.



А применив сходную тактику к кнопкам можно либо упростить разводку, либо по трем ножкам развести 6 кнопок.
Тут тоже все просто — одна нога дает подтяг, вторая косит под землю. Нажатие кнопки дает просадку напряжения на подтягивающей ножке. Это чует программа, поочередно опрашивающая каждую кнопку. Потом роли ножек меняются и опрашивается следующая кнопка.
В шестикнопочном режиме ситуация схожая — одна ножка дает подтяг, другая землю, а третья прикидывается ~~ветошью~~ Hi-Z и не отсвечивает. Но тут есть один побочный эффект. Например, опрашиваем мы кнопку “B”. Для этого у нас верхняя линия встает на **вход с подтяжкой** (PORTxy=1, DDRxy=0), средня дает **низкий уровень на выходе** (PORTxy=0, DDRxy=1), нижняя не участвует в процессе ибо стоит в **Hi-Z**(PORTxy=0, DDRxy=0). Если мы нажмем кнопку “B” то верхняя линия в этот момент просядет и программа поймет что нажата кнопка “B”, но если мы не будем жать “B”, а нажмем одновременно “E” и “Б” то верхняя линия также просядет, а программа подумает что нажата “B”, хотя она там и рядом не валялась. Минусы такой схемы — возможна неправильная обработка нажатий. Так что если девайсом будут пользоваться быдло-операторы, жмущие на все подряд без разбора, то от такой схемы лучше отказаться.
Ну и, напоследок, схема показывающая как можно обьединить кнопку и светодиод:



Обьединение кнопки и светодиода

Работает тоже исключительно в динамике. То есть все время мы отображаем состояние светодиода - то есть выдаем в порт либо 0 (диод горит) либо Hi-Z (диод не горит). А когда надо опросить кнопку, то мы временно (на считанные микросекунды) переводим вывод в режим вход с подтягом (DDRxy=0 PORTxy=1) и слушаем кнопку. Режим когда на выводе сильный высокий уровень (DDRxy=1 PORTxy=1) включать ни в коем случае нельзя, т.к. при нажатии на кнопку можно пожечь порт.

Минусы — при нажатии на кнопку зажигается светодиод как ни крути. Впрочем, это может быть не багой, а фичей :)
Вот такие пироги. А теперь представьте себе прогу в которой реализованы все эти динамические фичи + куча своего алгоритма. Выходит либо бесконечная череда опросов, либо легион всяких флагов. В таких случаях простейшая диспетчеризация или кооперативная RTOS это то что доктор прописал — каждый опрос гонишь по циклу своей задачи и не паришься. Зато юзаешь везде какую-нибудь ATTiny2313 и ехидно глядишь на тех кто в ту же задачу пихает Mega8 или что пожирней :)