Кнопка

Схема подключения

Кнопка обычно может находиться в одном из двух состояний: нажата и отжата. При этом в зависимости от подключения на ней могу быть разные потенциалы. Например, если кнопка подключена, так как изображено на рис 1, на отжатой кнопке в точке Input будет высокий потенциал, а при нажатой - низкий. Это связано с тем, что при отжатой кнопке точка Input подключена через резистор к питанию. Если кнопка нажата, то всё напряжение падает на резисторе, а точка Input оказывается закорочена на землю.

Если же кнопка подключена, как показано на рис 2, то, наоборот, при отжатом состоянии на кнопке будет ноль, а при нажатом, напряжение питания (логическая единица).

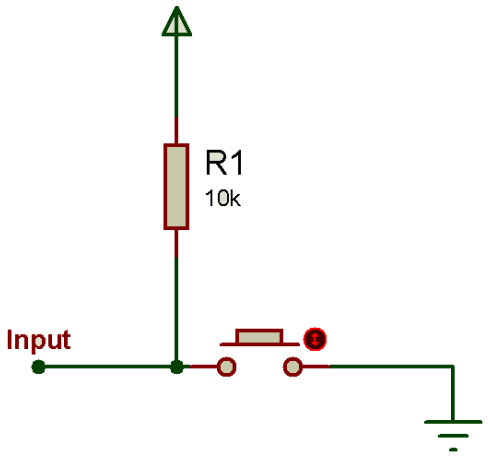

Рис 1

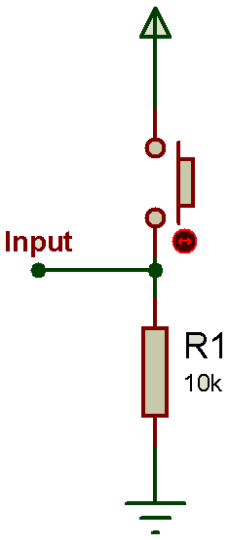

Рис 2

Дребезг контактов

При моделировании кнопка считается идеальной и её модель выглядит, как показано на рис 3.

Рис 3

Фронты при нажатии считаются идеальными. В реальности обычно при изменении состояния кнопки появляется эффект дребезга контактов. Этот эффект заключается в том, что пластины соприкоснувшись друг с другом (при нажатии) не остаются в таком состоянии, а какое-то время болтаются и сигнал на кнопке принимает вид, как показано на рис. 4.

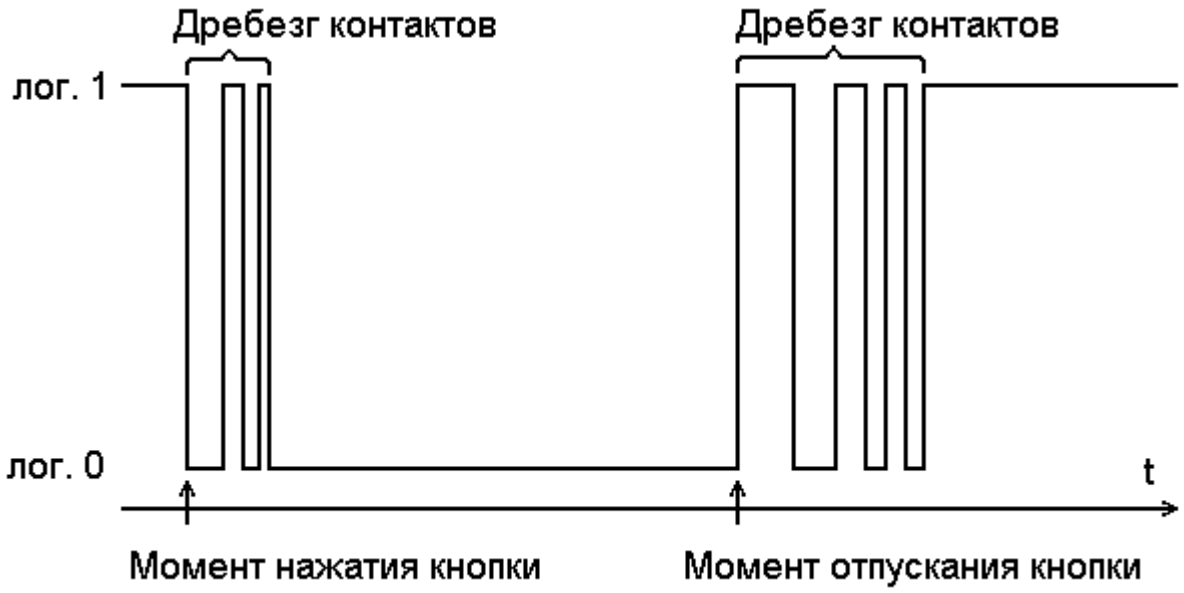

Рис 4

Длительность этого переходного процесса обычно составляет от сотых долей до единиц миллисекунд в зависимости от кнопки. Сам по себе дребезг представляет собой случайный процесс.

Дребезг не является проблемой, если необходимо просто отличать состояния нажатой и отжатой кнопок. Например, при решении задачи включения светодиода при зажатой кнопке.

Но он сильно усложняет задачу обнаружения момента нажатия на кнопку. Если просто отлавливать момент изменения уровня сигнала, то, как видно из рисунка, мы получим за одно нажатие хаотическое многократное нажатие и отжатие кнопки.

Алгоритм борьбы с дребезгом

Существует ряд алгоритмов, позволяющих бороться с эффектом дребезга. Самый простой из них – после первоначального обнаружения изменения уровня не отслеживать изменения уровня на выводе, к которому подключена кнопка в течение времени за которое гарантировано окончится эффект дребезга.

Этот алгоритм прост, но имеет тот недостаток, что может среагировать на шумовое изменение уровня сигнала от кнопки. Т.е. на мгновенное короткое изменение состояния сигнала. Это может произойти, например, из-за наводок с других дорожек.

Этот алгоритм обычно применяют при использовании внешних прерываний.

Другой алгоритм подразумевает накапливание информации об изменении состояния кнопки. Т.е. решение об изменении состояния принимается, если N раз подряд было получено ожидаемое значение. Например, если ожидается нажатие на кнопку, при котором на вывод микроконтроллера будет подан логический ноль, мы увеличиваем значение переменной на единицу, когда на выводе, в которому подключена кнопка, появляется ноль. Если на выводе единица, то счёт сбрасывается. Этот алгоритм позволяет отфильтровать дребезг и принять решение только при наличии на входе МК стабильного сигнала.

Алгоритмы борьбы с дребезгом

Общий обзор алгоритмов борьбы с дребезгом

Наиболее популярны 2 алгоритма борьбы с дребезгом. Первый – это реакция на первый же появившийся импульс и ожидание после этого времени, через которое дребезг гарантированно пройдёт.

Вторым является алгоритм с набором статистики. Этот алгоритм заключается в том, что если на входе устройство превалирует 0, то делается вывод, что на входе 0. И наоборот.

Разумеется, борьба с дребезгом нужна только, если требуется поймать момент нажатия на кнопку или отжатия оной.

Рассмотрим оба варианта отдельно.

Алгоритм борьбы с дребезгом с помощью задержки

В этом варианте проверяется, нажата ли кнопка. Если кнопка нажата, то совершается действие и проверка кнопки отключается на заданное время. Отключение можно производить с помощью функции delay или с помощью флага.

Алгоритм программы для борьбы с дребезгом с помощью функции delay может выглядеть примерно так, как представлено на рис 5.

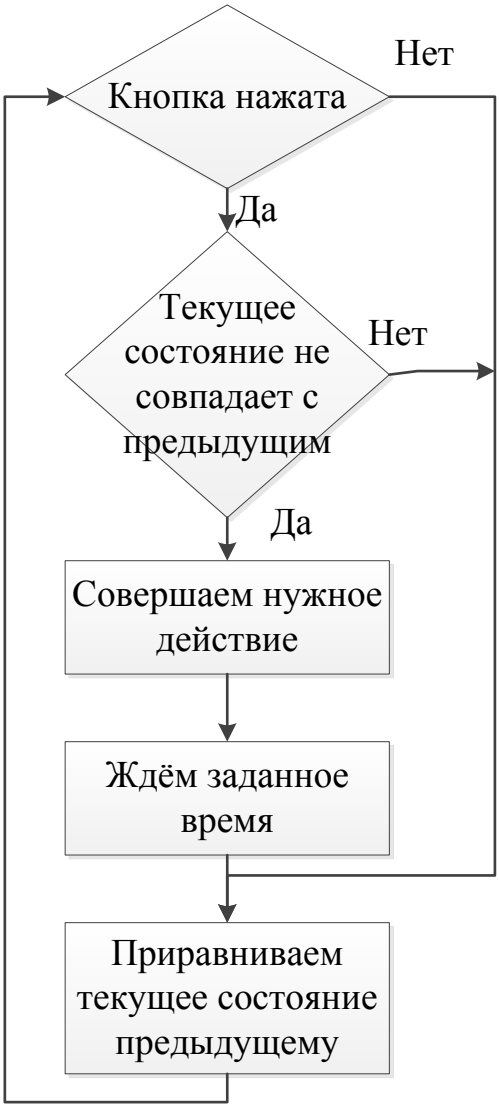

Рис 5

На рис 5 проверяем, нажата ли кнопка. Если кнопка нажата, то проверяем, совпадает ли текущее состояние кнопки с предыдущим. Т.к. кнопка только что была нажата, то её состояние не совпадает и условие выполняется. Далее совершаем нужные нам действия, которые должны быть совершены по нажатию на кнопку и ждём некоторое время, пока дребезг гарантированно не пропадёт. В конце присваиваем предыдущему состоянию следующее.

Если при следующем выполнении цикла, кнопка ещё будет нажата, текущее состояние уже будет равно предыдущему и условие не выполнится. Т.о. отлавливается только нажатие на кнопку.

Если же ожидать по флагу, то алгоритм немного поменяется на тот, что показан на рис 6.

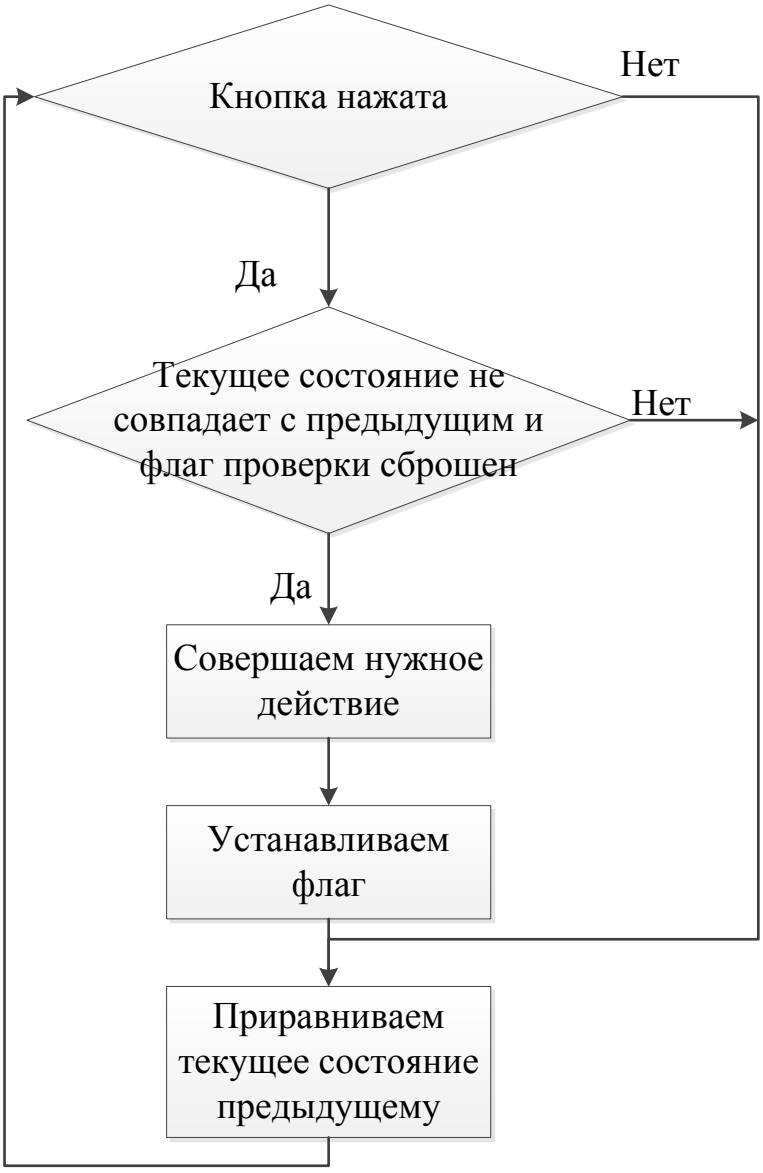

Рис 6

Отличием алгоритма на рис 6 от алгоритма на 5 является то, что при нажатии кнопки устанавливается флаг, запрещающий проверку кнопки. Сам флаг через заданное время сбрасывается в системном таймере.

Гистерезисный алгоритм борьбы с дребезгом

Алгоритмов, реализующих данный метод много. Рассмотрим один из них. Суть его заключается в получении интегральной составляющей входного сигнала. Т.е. какой сигнал чаще встречается (ноль или единица), в пользу того сигнала и делается вывод. Алгоритм данного анализа приведён на рис 7.

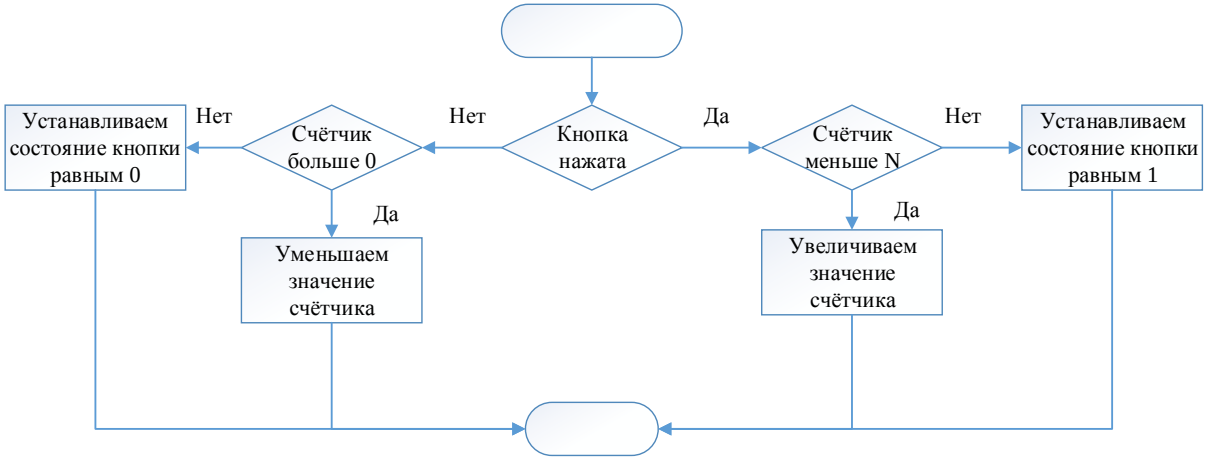

Рис 7

Алгоритм на рис 7 вызывается периодически в заданное время и проверяет, нажата ли кнопка. Если она нажата, то переменная Count увеличивается на 1, если отжата, то уменьшается. В результате, значения N или нуля она достигнет только тогда, когда ноль или единица явно преобладают в системе.

Такой способ обработки сигнала напоминает петлю гистерезиса, в которой обратный ход характеристики отличается от прямого.

Изменением значения N можно менять чувствительность кнопки, т.к. при больших значениях N могут пропускаться нажатия, а при очень маленьких – проскакивать дребезг.

Выход данного алгоритма уже можно рассматривать как кнопку без дребезга. Т.е. использовать алгоритмы, рассмотренные выше, но без функции задержки, т.к кнопка – идеальная.