**II. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

**1. ПОДКЛЮЧЕНИЕ КНОПОК К МИКРОКОНТРОЛЛЕРУ**

Общая схема подключения кнопок к порту Е микроконтроллера ATmega128 представлена на рис. 1.

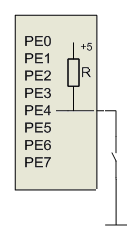


Рис. 1. Подключение кнопки к порту PE.4

В исходном состоянии контакты кнопочного переключателя разомкнуты. На вход микроконтроллера через внутренний подтягивающий резистор R подается напряжение от источника питания +5 В. Микроконтроллер воспринимает это напряжение как логическую единицу. При нажатии кнопки контакты замыкаются и соединяют вывод микроконтроллера с общим проводом. В результате напряжение на входе падает до 0 В. Этот уровень входного сигнала воспринимается микроконтроллером как логический «0».

Резистор R служит токоограничивающим элементом, предотвращающим короткое замыкание между шиной питания и общим проводом.

**2. ЯВЛЕНИЕ МЕХАНИЧЕСКОГО ДРЕБЕЗГА КОНТАКТОВ**

При замыкании и размыкании переключателей в цепи возникают импульсные помехи, вызванные дребезгом контактов (рис. 2). Эти помехи обычно называют «шумом» или «звоном». Такое явление часто возникает в системах на базе микроконтроллеров, где для ввода данных используется клавиатура, и «звон» может восприниматься как многократное нажатие клавиши.

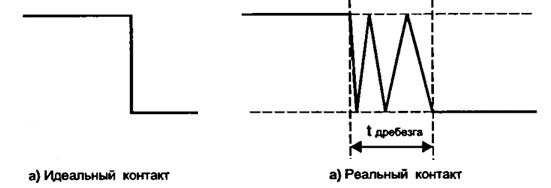


Рис. 2. Явление дребезга контактов

Чтобы устранить данный эффект, используются специальные схемы или программные методы для подавления «звона». Один из простых схемотехнических способов устранения «звона» состоит в подключении RC-цепи. В этой схеме время, требуемое для заряда разряда конденсатора до порогового напряжения, маскирует «звон» контактов при переключении. Можно также установить триггер Шмитта между схемой ключа и микроконтроллером, чтобы усилить эффект подавления «звона».

Недостатки этих методов – дополнительные затраты на компоненты, которые должны быть установлены на плате, и дополнительное время, требуемое для заряда/разряда RC-цепи. Все это может затруднить использование данной схемы, так как для некоторых ключей с большим уровнем шумов дополнительная задержка может составить десятые доли секунды.

Значительно лучший способ избавиться от «звона» – сделать это программно (рис. 3). Если уровень напряжения на выходе ключа не изменяется в течении 20 – 50 мс, то можно считать, что «звон» окончился, и больше изменения состояний не ожидается. В программе для подавления эффекта дребезга используется задержка.

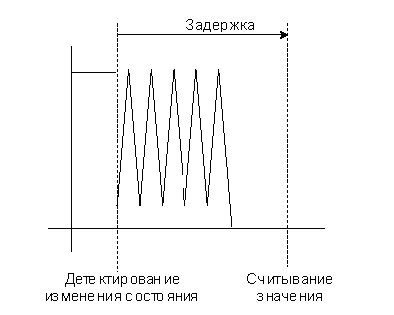


Рис. 3. Пример подавления эффекта дребезга контактов

**3. ОБРАБОТКА НАЖАТИЯ КНОПКИ**

Существует два варианта обработки нажатия кнопки:

1) по опросу;

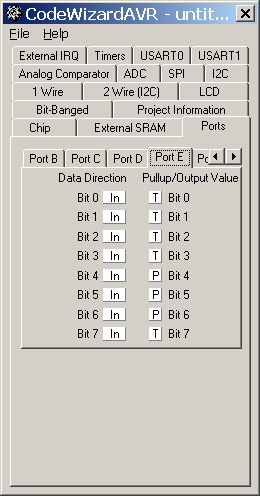
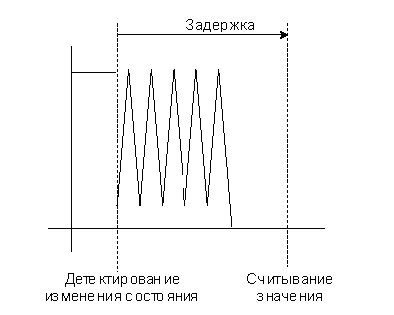
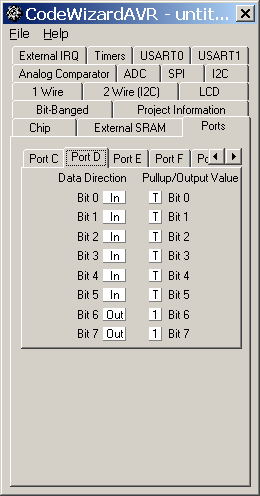
2) по прерыванию.

Первый вариант заключается в постоянной проверке состояния входа того разряда порта, к которому подключена кнопка (в бесконечном цикле). Если на входе логическая «1», то кнопка не нажата, если «0» – нажата. Этот способ обработки нажатия неудобен тем, что процессор будет всё время занят опросом кнопки.

Второй вариант больше подходит для обработки нажатия кнопки. Он заключается в том, что по нажатию кнопки генерируется прерывание, в обработчике которого выполняются необходимые действия, в том числе и защита от дребезга. Таким образом, отпадает необходимость постоянного сканирования состояния входа.

Рассмотрим подробнее реализацию первого варианта в CodeVisionAVR. На отладочной плате AS-megaM кнопки подключены к выводам PE.4 – PE.7, которые необходимо настроить на ввод с использованием подтягивающего резистора (рис. 4а).

Для индикации нажатия кнопки будем использовать светодиоды, подключенные к выводам PD.6 и PD.7 порта D микроконтроллера.

а) для кнопок б) для светодиодов

Рис. 4. Настройка портов ввода/вывода

**4. ПРИМЕР ПРОГРАММЫ ОПРОСА КНОПКИ**

while (1)

{

// опрос кнопки в бесконечном цикле

if (PINE.4 = = 0)

{ delay\_ms(100); // задержка для исключения дребезга

if (PINE.4 = = 0)

{PORTD.6 = ~PORTD.6; // изменение сост. диода на PD6 }}

// постоянное мигание диода на PD7

PORTD.7 = 0; // Включение светодиода

delay\_ms(200); // Пауза 0,2 с

PORTD.7 = 1; // Выключение светодиода

delay\_ms(200); // Пауза 0,2 с

};

Теперь рассмотрим реализацию второго варианта в CodeVisionAVR. Для обработки нажатия кнопки будем использовать внешнее прерывание.

**5. ПРЕРЫВАНИЯ В МИКРОКОНТРОЛЛЕРАХ AVR**

Прерывание – это запуск специальной подпрограммы (называемой «обработчиком прерывания» или «программой обслуживания прерывания»), который вызывается поступлением специального сигнала от одного из внешних или внутренних аппаратных блоков МК. На время выполнения этой подпрограммы реализация текущей программы приостанавливается и в стеке сохраняется ***регистры контекста***.

МК может не реагировать на прерывания, пока не завершится выполнение текущей задачи – это реализуется путём запрещения (маскирования) обслуживания запроса прерывания. После решения задачи возможен один из двух вариантов: сброс маски и разрешение обслуживания прерывания, что приведет к вызову обработчика прерывания, или анализ значения битов, указывающих на поступление запросов прерывания и непосредственное выполнение программы обслуживания без вызова обработчика прерывания. Такой метод обработки прерывания используется, когда требуется обеспечить заданное время выполнения основной программы, так как любое пребывание может нарушить реализацию необходимого алгоритма.

Обработчик всегда обеспечивает следующую последовательность действий:

1. Сохраняет содержимое регистров контекста;

2. Сбрасывает контроллер прерываний и оборудование, вызвавшее запрос;

3. Обрабатывает данные;

4. Восстанавливает содержимое регистров контекста;

5. Возвращается к выполнению прерванной программы.

***Регистры контекста – это регистры, определяющие текущее состояние выполнения основной программы.***

Обычно к ним относят счетчики команд (СК) и регистры состояния процессора (РСП). Другие регистры МК, такие как индексные регистры, регистры общего назначения (РОН) могут быть использованы в процессе обработки прерывания, поэтому при необходимости их содержимое также сохраняется в стеке. Все остальные регистры являются специфическими для каждого типа МК и их использование зависит от конкретной задачи.

После сброса в исходное состояние контроллер прерываний готов воспринимать следующий запрос, а оборудование, вызывающее прерывание, готово посылать запрос, когда возникают соответствующие события. Если поступит новый запрос прерывания, то сброшенный разряд регистра маскирования прерываний процессора предотвратит обработку прерывания, но регистр состояния прерываний зафиксирует поступивший запрос, который будет ожидать своего обслуживания. После завершения обслуживания текущего прерывания маска разрешения прерывания будет восстановлена, и вновь поступивший запрос поступит на обработку.

***Вектор прерывания – это адрес, который загружается в СК при переходе к обработчику прерывания*.** Существует несколько типов векторов. Адрес, который загружается в СК при запуске МК (или по сигналу RESET) называется «вектор сброса». С этого адреса начинается программа, которая выполняется в микроконтроллере.

В регистре **SREG** есть бит 7(он называется «I»).

Если этот бит равен «1» (установлен), то ГЛОБАЛЬНО разрешены прерывания – значит, могут возникать разрешенные индивидуально прерывания.

Если этот бит равен «0» (бит сброшен, очищен), то ГЛОБАЛЬНО запрещены прерывания – значит любые события не вызывают прерывания программы.

Флаг прерывания – это просто бит, который становится «1» (устанавливается) при возникновении события, которое может вызвать прерывание независимо от того разрешены ли прерывания ГЛОБАЛЬНО (установленным битом 7 в SREG) или индивидуально. То есть флаг всегда устанавливается при возникновении события, которое может вызывать прерывание – даже если это прерывание не обрабатывалось программой, и не было разрешено.

Чтобы сбросить флаг прерывания нужно записать в него «0».

Программно установить бит флага прерывания НЕВОЗМОЖНО!

Если программа находится в функции обработчика прерывания (и в этой функции программно не был установлен бит SREG.7 (так обычно и происходит по умолчанию), то другие события вызывающие прерывания не могут уже вызвать прерывание программы и свою обработку, они лишь «устанавливают» свои флаги, образуя, таким образом, очередь флагов.

Поэтому, делайте функции-обработчики прерывания как можно короче! Не засиживайтесь в них.

Нужно тщательно продумывать алгоритм программы, чтобы успевать обрабатывать все прерывания – то есть не пропускать нужные события и обрабатывать их вовремя.

В конце функции обработчика прерывания компилятор ставит инструкцию RETI, после выполнения которой бит 7 в регистре SREG устанавливается в «1» – то есть прерывания опять ГЛОБАЛЬНО разрешаются и программа продолжает выполняться с того места, где она была в момент возникновения прерывания – то есть выполняет инструкцию, следующую за той, на которой прервалась.

Однако,если в момент глобального разрешения прерываний обнаружится установленный флаг разрешённого прерывания, то будет вызвана функция-обработчик этого прерывания.

Такая ситуация может возникнуть, если во время выполнения обработчика прерывания возникло другое прерывание – то есть установился его флаг и оно разрешено (включено) программой.

Если возникнет несколько разрешенных прерываний одновременно или несколько прерываний пока бит 7 был нулевым, то после установки бита 7 в SREG первым будет выполняться то прерывание, которое выше в списке векторов прерываний МК (см. описание конкретного МК). Соответственно, по мере отработки очереди накопившихся и разрешённых прерываний их флаги будут очищаться.

А вот флаги неразрешенных прерываний не очистятся, пока программа этого не сделает записью в них числа «0». Вы можете сами очистить флаги прерываний записью в них «0».

***Вы можете запрещать и разрешать как все прерывания сразу, так и каждое по отдельности!***

Изменить бит 7 в регистре SREG можно следующими командами в компиляторе CodeVisionAVR:

**#asm("sei")** / бит I устанавливается в «1», теперь разрешённые прерывания будут обрабатываться. Если есть установленный флаг разрешённого прерывания, то произойдет вызов его функции-обработчика и флаг очистится.

**#asm("cli")** / бит I устанавливается в «0» и запрещаются все прерывания **ГЛОБАЛЬНО**.

Внешние прерывания в микроконтроллере ATmega128 поступают на порт E и могут соответствовать наступлению одного из следующих событий:

1. Появление низкого уровня на входе;

2. Появление восходящего фронта на входе;

3. Появление ниспадающего фронта на входе;

4. Появление любого изменения состояния на входе.

Требуемый режим генерации внешнего прерывания может быть установлен в CodeWizardAVR (рис. 5).

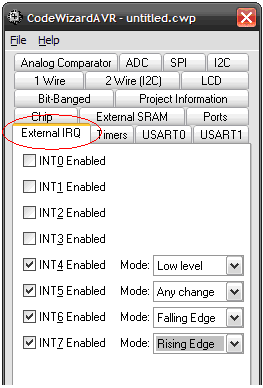


Рис. 5. Настройка внешних прерываний в CodeWizardAVR

А в обработчике прерываний реализуются необходимые для обработки соответствующего события действия (рис. 6).

Для копирования необходимой части кода можно воспользоваться окном просмотра сгенерированного кода программы. Для этого выберите в главном меню пункт «**File**», далее «**Program Preview**».

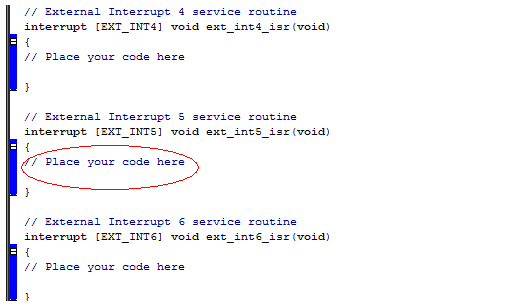


Рис. 6. Окно Program Preview

**6. ПРИМЕР ПРОГРАММЫ ОБРАБОТКИ**

**ВНЕШНЕГО ПРЕРЫВАНИЯ**

#include <mega128.h>

#include <delay.h>

// по нажатию кнопки вызывается прерывание (используйте прерывание по ниспадающему уровню)

// Declare your global variables here

interrupt [EXT\_INT4] void ext\_int4\_isr(void)

{

EIMSK &= 0xEF; //«&=» поразрядное (побитное) логическое И

для маскирования прерывания INT4

delay\_ms(50);

if (PINE.4 = = 0)

PORTD.6 = ~PORTD.6;

EIMSK |= 0x10; // «|=» поразрядное (побитное) логическое ИЛИ для

разрешения прерывания INT4

}

#asm("sei") // глобальное разрешение прерываний

while (1)

{

// для постоянного мигания светодиода PD6

PORTD.6 = 0;

delay\_ms(500);

PORTD.6 = 1;

delay\_ms(500);

}

}

**III. ПРАКТИЧЕСКАЯ ЧАСТЬ**

**Задание**: Напишите программу обработки двух кнопок. При нажатии первой кнопки число должно увеличиваться на 1, при нажатии второй – уменьшаться на 1. Диапазон изменения числа – от 0 до 99. Вывод результата необходимо реализовать с использованием динамической индикации.

Напишите два варианта программы:

1. Кнопки обрабатываются по опросу;

2. Кнопки обрабатываются по прерыванию.