Лабораторная работа №3

Порты ввода вывода и работа с прерываниями

Теоретическая часть

При разработке программы для микроконтроллера мы можем отслеживать состояние входа двумя способами:

- Поллинг. Можно с заданным интервалом опрашивать состояние входа, сравнивать текущее значение с предыдущим, и при изменении совершать заданные действия.
- Прерывания. Определенным образом настроив микроконтроллер, можно заставить его выполнять нужную функцию мгновенно по приходу переднего или заднего фронта.

Очевидно, что второй способ выгоднее сразу по нескольким параметрам. Преимущества использования прерываний:

- Минимальные задержки, прерывание вызывается сразу же при наступлении события
- Отсутствие пустой нагрузки на ЦП, так как не нужно постоянно опрашивать вход
- Простота реализации

Прерывания в микроконтроллерах представляют собой механизм, который позволяет микроконтроллеру реагировать на внешние события. Этот механизм работает таким образом, что при наступлении некоторого события в процессоре возникает сигнал, заставляющий процессор прервать выполнение текущей программы, т.е. говорят, что возникло прерывание. После того как выполнение текущей программы прервано, процессор должен перейти к выполнению программной процедуры, связанной с этим событием (прерыванием) – процедуры обработки прерывания. Однако, прежде чем перейти непосредственно к процедуре обработки прерывания, процессор должен выполнить ряд предварительных действий. Прежде всего, для того чтобы в будущем он смог корректно продолжить прерванную программу, необходимо сохранить состояние процессора (счётчик команд, слово состояния процессора, внутренние регистры и т.д.) на момент, предшествующий прерыванию. Т.е. другими словами, требуется сохранить состояния всех тех ресурсов, которые так или иначе могут быть изменены в процессе обработки прерывания. Далее, если в системе имеется несколько возможных источников прерываний (а обычно так и бывает), процессор должен определить источник запроса прерываний. И, наконец, затем перейти к самой процедуре прерываний, конкретной для данного прерывания. По завершению обработки прерывания процессор должен восстановить состояние ресурсов, соответствующее прерванной программе, после чего она может быть продолжена. Следует отметить, что для сохранения всех требуемых ресурсов, поиска источника прерывания и перехода к процедуре обработки прерывания процессор должен затратить вполне определенное время. Это время называется скрытым временем прерывания. Чем меньше скрытое время прерывания, тем выше скорость реакции системы на внешние события и тем выше производительность системы. Во многом это определяется системой прерывания процессора и она является одной из основных особенностей архитектуры контроллера.

Под системой прерываний мы будем понимать совокупность аппаратных и программных средств, реализующих механизм прерываний в микроконтроллере. Хотя существуют большое множество различных вариантов построения систем прерываний, тем не менее, можно выделить несколько основных способов организации систем прерываний. Они отличаются между собой объемом аппаратных средств, необходимых для реализации такой системы и соответственно имеют различное быстродействие. Рассмотрим эти варианты.

Одноуровневые прерывания

Данная система прерываний реализована таким образом, что при возникновении прерывания процессор аппаратно переходит к подпрограмме обработки прерываний, расположенной по некоторому

фиксированному адресу. Чтобы упростить аппаратную часть системы прерываний, этот адрес обычно располагается либо в начале, либо в конце адресного пространства программной памяти. Поскольку для обработки ВСЕХ прерываний используется только ОДНА точка входа, то такая система прерываний получила название одноуровневой. В такой системе выявление источника прерываний путем опроса состояния флажков признаков прерываний в начале программы обработки прерываний. При обнаружении установленного флажка происходит переход к соответствующему участку процедуры. Чем больше возможных источников прерываний, тем больше времени необходимо для обнаружения источника прерывания. Такой метод обнаружения источника прерывания называется программным опросом или поллингом (polling). Его недостатком является довольно большое время, затрачиваемое на поиск источника прерывания и, как следствие, замедленная реакция системы на внешние события. Его достоинство – простота реализации системы прерываний.

Векторные прерывания

Чтобы значительно уменьшить время реакции на внешние события, используются многоуровневые или, что то же самое, векторные прерывания. В векторных прерываниях КАЖДОМУ источнику прерывания соответствует СВОЙ, вполне определенный, адрес процедуры обработки прерывания, который принято называть вектором прерывания.

Вообще, в качестве вектора прерывания могут быть использованы любые данные (адрес подпрограммы, адрес перехода, значение смещения относительно начала таблицы прерываний, специальные инструкции и т.д.), которые позволяют непосредственно перейти к процедуре обработки прерывания, не затрачивая времени на поиск источника прерывания. Какие данные используются в качестве вектора прерывания, и каким именно образом они используются, зависит от способа реализации системы прерываний в соответствующем процессоре.

Прерывания в микроконтроллерах STM32

Контроллер вложенных векторизированных прерываний STM32 (Nested vectored interrupt controller (NVIC)), в зависимости от модели микроконтроллера, способен обслуживать до 68 источников прерываний IRQ от периферийных модулей микроконтроллера и осуществлять:

- разрешение и запрет вызова прерываний;
- назначение и динамическое изменение приоритета прерываний (16 уровней от 0 (максимального) до 15);
- автоматическое сохранение и восстановление контекста данных при обработке прерываний;
- при одновременном вызове, механизм отложенных прерываний позволяет отложить обработку менее приоритетного прерывания, без возврата в фон и восстановления контекста данных.

Работа с прерываниями в библиотеке Mbed

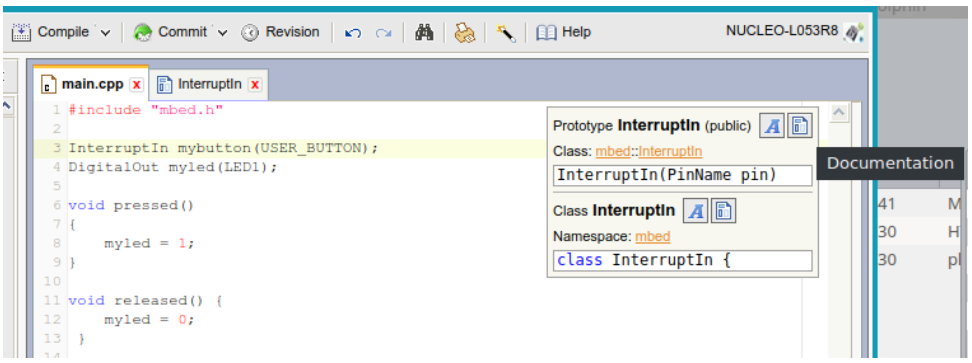
Библиотека Mbed в значительной степени упрощает работу с прерываниями. Для каждого из типов прерываний существует специализированный набор классов. В данной работе будем рассматривать так называемые внешние прерывания (External Interrupts), с помощью которых можно определять смену логического уровня на одном из входов контроллера. Для этого необходимо использовать класс **`InterruptIn`**:

```
InterruptIn mybutton(USER_BUTTON);
```

Как можно заметить, объявление внешнего прерывания аналогично объявлению цифрового входа. В качестве параметра конструктора передается номер входа, с которым будет связано данное прерывание.

Методы и свойства класса InterruptIn

Для того, чтобы ознакомиться с доступными свойствами и методами классов библиотеки необходимо установить курсор на название класса и нажать кнопку Документация, как показано на скриншоте:



int	read ()	
		Считывает текущее состояние входа в виде 0 или 1 (int)
void	rise (Callback< void()> func)	
		Подключает функцию-обработчик, которая будет вызвана по восходящему фронту (переход из 0 в 1).
void	fall (Callback< void()> func)	
		Подключает функцию-обработчик, которая будет вызвана по нисходящему фронту (переход из 1 в 0).
void	enable_irq ()	
		Включить обработку прерываний
void	disable_irq ()	
		Выключить обработку прерываний

В первую очередь, нам интересны функции rise и fall, которые позволяют привязать наши собственные функции, к смене состояний входа. Говоря проще, мы может вызывать определенные функции по нажатию на кнопку и по отжатию. При этом, не будет необходимости в постоянном опросе кнопки и хранения предыдущего состояния.

Записывается это следующим образом:

```
#include "mbed.h"

InterruptIn mybutton(USER_BUTTON);
DigitalOut myled(LED1);

void pressed() {
    myled = 1;
}

void released() {
    myled = 0;
 }

int main() {
    mybutton.fall(&pressed);
    mybutton.rise(&released);
    while (1) {}
}

```

В этом примере создаются два объекта mybutton, отвечающий за прерывания от кнопки USER_BUTTON и myled, отвечающий за управление светодиодом LED1.

Затем объявляются две функции – pressed() и released(), в одной из которых светодиод включается, а в другой выключается.

Далее, в функции main, вызываются методы fall и rise у объекта mybutton. В качестве параметров этим

методам передаются ссылки на функции pressed и released (обратите внимание на знак & перед именем функции, он означает, что функция не вызывается, а берется ссылка на нее).

После этого программа переходит в бесконечный цикл while, в котором ничего не происходит.

Если загрузить эту программу в микроконтроллер, то можно пронаблюдать, что при нажатии на кнопку USER_BUTTON светодиод будет загораться, а при отпускании гаснуть. Это происходит потому, что к этим событиям (нажатия и отпускания кнопки) привязаны функции pressed и released. И они вызываются при соответствующих действиях.

Так же, обратите внимание на то, что функция нажатия на кнопку (pressed) привязана к событию fall, то есть, к нисходящему фронту, переходу логического уровня из 1 в 0, а функция отпускания кнопки привязана к событию rise, то есть, к восходящему фронту. Так сделано потому, что как было указано в прошлой работе, вход к которому подключена кнопка подтягивается к логической единице в те моменты, когда кнопка не нажата.

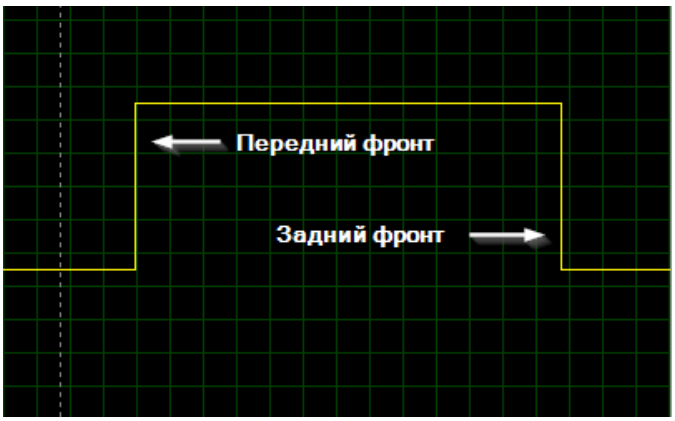


Рис. Передний и задний фронты сигнала

Подключение дополнительных кнопок и светодиодов

На отладочной плате всего одна кнопка и один светодиод доступные для пользователя. Это несколько ограничивает в возможностях, поэтому подключим дополнительные наборы кнопок и светодиодов.

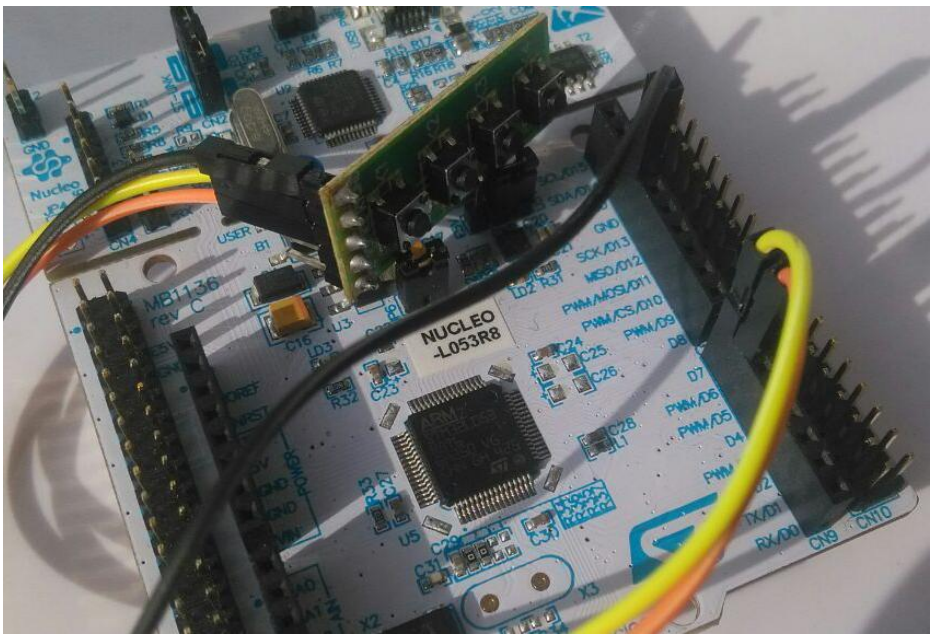
ВАЖНО!

При подключении дополнительных плат ВСЕГДА используйте черный провод для подключения 0U (земли), и красный провод для подключения V+ (питания). При подключении управляющих и сигнальных линий можно использовать остальные цвета проводов.

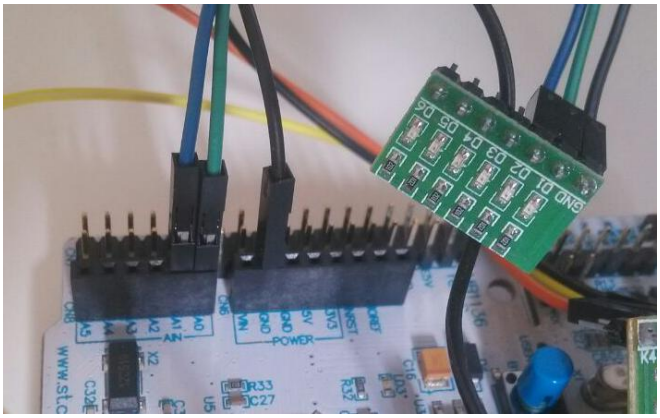
ВАЖНО!

При подключении дополнительных плат ВСЕГДА отключайте отладочную плату от USB!

Сперва подключим пару кнопок. Для этого выберем на плате разъемы которые могут быть цифровыми входами\выходами. Они подписаны как Dxx. И найдем ближайший разъем земли, он подписан как Gnd (ground). И соединим их проводами с соответствующими разъемами на плате кнопок:



Затем подключим пару светодиодов аналогичным образом:



Соответственно, кнопки подключены к входам D7, D6, а светодиоды к выходам A0, A1.

Если посмотреть внимательно на плату кнопок, то можно увидеть (а точнее не увидеть), что на ней нет ничего кроме самих кнопок. А это значит, что отсутствует цепь подтяжки. Это значит, что при отжатой кнопке, вход микроконтроллера будет находиться в подвешенном состоянии, и микроконтроллер будет не в состоянии определить нажата кнопка или отжата. И если мы запустим программу аналогичную вышенаписанной, то увидим, что светодиоды будут мерцать, гореть с разной степенью яркости и не реагировать на нажатия кнопок. Эту проблему можно решить двумя способами:

- При помощи ловкости рук, паяльника и резистора на 50-200 кОм добавить на схему цепь внешней подтяжки
- Программным способом включить уже встроенные в микроконтроллер подтягивающие резисторы

Так как для работы с паяльником в лаборатории требуется особое разрешение, воспользуемся вторым способом. Для этого в классе InterruptIn имеется метод – mode(int PullMode). Он может принимать одно из трех значений:

- PullUp – подтяжка к логической единице
- PullDown – подтяжка к логическому нулю
- PullNone – отключить подтяжку (по умолчанию)

Перепишем программу следующим образом:

```
#include "mbed.h"

InterruptIn btn1(D7);
InterruptIn btn2(D6);
DigitalOut  led1(A0);
DigitalOut  led2(A1);

void btn1Pressed() {
    led1 = 1;
}
void btn1Released() {
    led1 = 0;
}

void btn2Pressed() {
    led2 = 1;
}
void btn2Released() {
    led2 = 0;
}
int main() {
    btn1.mode(PullUp);
    btn2.mode(PullUp);
    btn1.fall(&btn1Pressed);
    btn1.rise(&btn1Released);
    btn2.fall(&btn2Pressed);
    btn2.rise(&btn2Released);
    while (1) {}
}

```

Если ее прошить в контроллер, то по нажатию на каждую из кнопок, будет загораться соответствующий ей светодиод.