Лабораторная работа №4

Внутренние таймеры микроконтроллера

Теоретическая часть

Таймеры предназначены для формирования временных интервалов, позволяя микропроцессорной системе работать в режиме реального времени. Таймеры представляют собой цифровые счётчики, которые подсчитывают импульсы либо от высокостабильного генератора частоты, либо от внешнего источника сигнала, в этом случае таймер называют счетчиком внешних событий. К системной шине микропроцессора таймеры подключаются при помощи параллельных портов/регистров имеющих определенный адрес в адресном пространстве.

Как правило, в микропроцессорной системе в качестве генератора частоты выступает генератор внутренней синхронизации микроконтроллера. Частота генератора задает минимальный временной промежуток, который может определять таймер. Интервалы времени, задаваемые с помощью таймера, могут иметь строго определенные дискретные значения. Разрядность цифрового счётчика таймера определяет максимальный интервал времени, который может задать таймер.

Обычно в микропроцессорных системах используются 16 и 32 разрядные таймеры. Для управления их работой и доступа к текущим значениям используются различные наборы регистров. В зависимости от микропроцессора таймеры могут обладать различными параметрами и режимами работы.

Несмотря на свое название, таймеры могут использоваться не только для измерения временных отрезков. Одним из частых способов использования таймеров является генерация ШИМ (широтно-импульсная модуляция, PWM) сигнала. ШИМ используется для управления мощностью подаваемой на нагрузку путем изменения скважности дискретного сигнала. ШИМ часто используется в электродвигателях, для регулировки яркости подсветки дисплеев и др.

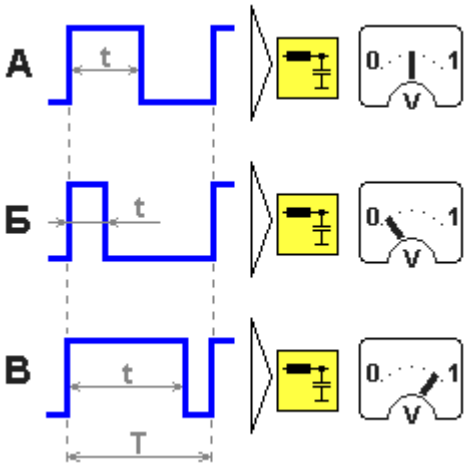


Рисунок 1. Широтно-импульсная модуляция

При использовании ШИМ аналоговый сигнал кодируется путем изменения ширины прямоугольных импульсов. Как правило, при использовании ШИМ **период модуляции** T остается неизменным, а меняется соотношение t/T, которое называется **скважностью** сигнала. После пропускания такого сигнала через фильтр низких частот (отмечен желтым на рис.1) напряжение на выходе будет соответствовать скважности. Если соотношение t/T будет низким (рис.1 А. Ширина высокого уровня значительно меньше ширины низкого), то напряжение на выходе будет так же низким, и лампочка будет гореть тускло. Если соотношение t/T будет приближаться к 1 (рис.1 В. Ширина высокого уровня значительно больше ширины низкого), то напряжение на выходе будет высоким и лампочка будет гореть ярко. Меня **скважность** можно плавно

менять и напряжение на выходе.

Другой областью применения таймеров является подсчет импульсов от внешних источников. Так как при настройке таймера можно указывать в качестве генератора импульсов внешний вход, то мы можем подключать к нему различные устройства передающие информацию при помощи импульсов, например:

- Приборы учета ресурсов: водо-, электро-, газосчетчики
- Поворотные энкодеры (регуляторы передающие направление угол поворота)

Микроконтроллеры STM32 имеют весьма широкий набор таймеров, которые разделяются на 3 основных группы:

- Basic Timers – это самые простые таймеры, имеющие возможность только отсчитывать промежуток времени и генерировать прерывание по его истечении. В микроконтроллерах STM32 это, обычно, таймеры TIM6/TIM7.
- General-Purpose Timers – это таймеры общего назначения (универсальные таймеры). В STM32L053 это TIM2 (16-битный), TIM21 и TIM22 (16-битные), обеспечивающие прямой, обратный и двунаправленный счёт.
- Advanced-Control Timer – расширенный таймер (таймер улучшенного управления), помимо всех функций General-Purpose Timers, позволяет также управлять электродвигателем. В STM32L053 отсутствует

Так же в микроконтроллерах серии L присутствует так называемый низкопотребляющий таймер LowPower Timer который может работать, даже если ядро контроллера находится в выключенном состоянии. Может работать в режимах ШИМ, подсчета импульсов и режиме энкодера.

За настройку всех режимов отвечает более двух десятков регистров. К счастью для вас, среда mbed все делает за вас и предоставляет удобный и интуитивно понятный интерфейс.

Таймеры в среде Mbed

Класс Timer

Для работы с таймерами в Mbed используется несколько различных классов в зависимости от назначения. Для начала рассмотрим базовый класс **`Timer`**. Обявляется он просто:

```
Timer myTimer;
```

Среда автоматически определит какие аппаратные таймеры доступны для текущего контроллера, какие еще не заняты и доступны для использования. Класс Timer имеет следующие методы:

```
  void    start ()
          Запускает таймер
  void    stop ()
          Останавливает таймер
  void    reset ()
          Сбрасывает счетчик в 0
 float    read ()
          Возвращает отсчитанное время в секундах
   int    read_ms ()
          Возвращает отсчитанное время в милисекундах
   int    read_us ()
          Возвращает отсчитанное время в микросекундах
```

Этот класс позволяет измерять время между какими-либо событиями. Например, следующая программа зажигает светодиод если кнопка была нажата два раза в течении 500 мс. :

```
1  #include "mbed.h"
2
3  Timer tim;
4  DigitalOut led(LED1);
5  InterruptIn btn(USER_BUTTON);
6
7  void onButtonClick() {
8      if(tim == 0) {      //если таймер еще не запущен
9          tim.start();    //то запускаем
10     }
11     else {              //иначе
12         if(tim < .5) {  //если прошло меньше 500мс
13             led = !led; //меняем состояние светодиода
14             tim.stop(); //останавливаем таймер
15         }
16         tim.reset();    //сбрасываем таймер в ноль
17     }
18 }
19
20 int main() {
21     btn.rise(&onButtonClick);
22     while(1) {
23     }
24 }
25
```

В этой программе таймер используется для подсчета времени между двумя нажатиями на кнопку. При первом нажатии после запуска программы таймер будет остановлен и будет показывать 0 с, тогда по условию на строке 8, таймер будет запущен и начнет считать время. При втором нажатии, условие стр.8 не будет выполнено и программа перейдет к условию в стр.12. Если к моменту второго нажатия пройдет менее 500 мс (0.5 с), то условие выполнится и программа переключит светодиод и остановит таймер. Если же времени прошло больше, то программа сбросит текущее подсчитанное время и начнет считать время до следующего нажатия, после чего процесс повторится.

Следует отметить, что таймер использует 32битный счетчик для хранения количества микросекунд. Из этого следует, что он не может хранить значения интервалов длиннее чем 2^{31} - 1 мс, что равно примерно 30 минутам. **Для измерения более длинных временных отрезков следует использовать часы реального времени.** Что это и как ними работать будет рассмотрено в следующих работах.

Класс Timeout

Следующий класс для работой с таймером называется **`Timeout`**. Он может быть полезен когда в программе требуется запустить функцию через некоторое время. Например, для проверки нажатия кнопки на дребез контактов.

У класса **`Timeout`** имеется только один метод – **`attach`**, который используется для назначения выполнения функции через указанное время. Например:

```
Timeout tim;
tim.attach(&func, 0.5);
```

В предыдущей работе одной из самых очевидных проблем было то, что нажатия кнопок считывались нечетко и наблюдалось большое количество ложных нажатий. При однократном нажатии на кнопку, программа могла посчитать, что кнопка была нажата 2, 3 и больше раз. Это явление называется дребезгом контактов. Вызывается оно тем, что при нажатии кнопки происходит многократное замыкание-размыкание контактов внутри нее, и каждое из них вызывает прерывание (подробнее можно прочитать в википедии: https://ru.wikipedia.org/wiki/Дребезг_контактов). Это явление можно побороть программным способом, путем измерения состояния входа кнопки через некоторое время после срабатывания перерывания. Вот пример кода:

```
#include "mbed.h"

//определение типа описывающего состояние кнопки
enum ButtonState {
    PRESSED,        //кнопка нажата - 0
    UNPRESSED       //кнопка отжата - 1
};
InterruptIn button(USER_BUTTON);
DigitalOut led(LED1);
Timeout tim;
int counter;   //счетчик проверок кнопки

//функция проверки состояния кнопки,
//вызывается несколько раз при каждом нажатии
void checkButton() {
    if(button == UNPRESSED) {   //если кнопка уже не нажата
        button.enable_irq();    //включаем прерывание
        return;                 //и выходим
    }
    counter ++;                 //иначе - инкрементируем счетчик
    if(counter >= 3) {          //досчитали до 3
        led = !led;             //совершаем действие по нажатию кнопки
        button.enable_irq();    //и разрешаем прерывания
    }
    else {                      //если еще не досчитали до 3
        tim.attach(&checkButton, 0.05); //то повторим проверку через 50 мс.
    }
}

//обработчик нажатия на кнопку
void onButtonClick() {
    button.disable_irq();           //отключаем повторые прерывания
    tim.attach(&checkButton, 0.05); //запустим процесс проверки кнопки через 50мс
    counter = 0;                    //обнуляем счетчик проверок
}

int main()
{
    button.fall(&onButtonClick);
    while (1) { }
}
```

В этой программе происходит следующее:

1. На нажатие кнопки назначается обработчик onButtonClick. Он выполняется каждый раз когда нажимается кнопка
2. При нажатии на кнопку, в onButtonClick происходит 3 действия:
    a. Отключается прерывание кнопки. Это означает, что до тех пор пока прерывания кнопки не будут включены на нажатия кнопки не будет никаких реакций в программе. Это нужно, чтобы при дребезге функция onButtonClick не вызывалась несколько раз. Таким образом мы отфильтровываем лишние сигналы о нажатии кнопки
    b. Затем назначается запуск функции checkButton через 50мс
    c. Счетчик проверок кнопки сбрасывается в 0.
3. 50 мс спустя запускается функция checkButton – стр. 15
4. стр 16. Сперва производится проверка – а нажата ли еще кнопка, если кнопка уже отжата, то считаем, что никакого нажатия не было, и это была какая-то наводка. Выходим из функции
5. стр. 20. Инкрементируем счетчик проверок.
6. стр. 21. Если выполнено 3 проверки, при каждой из которых было установлено, что кнопка была нажата, то выполняем действие: меняем состояние светодиода. И включаем прерывание кнопки, это нужно, чтобы мы могли нажать кнопку еще раз.
7. стр 25. Если же еще не было пройдено 3 проверок, то назначаем функцию проверки еще раз через 50 мс.

С подобным алгоритмом работы, в программе всегда будет фиксироваться только четкое и уверенное нажатие кнопки, проблемы с двойным нажатием или несработкой будут исключены (при условии, физической исправности кнопки).

Так же в дополнительных библиотеках Mbed присутствуют специально предназначенные классы для входов с поддержкой антидребезга – **`DebounceIn`** и **`PinDetect`**, попробуйте разобраться с ними самостоятельно.

Класс Ticker

Следующий класс для работы с таймерами – **`Ticker`**. Он используется, когда необходимо выполнять какие-либо действия с заданным интервалом, например, световая или звуковая индикация.

Использование класса **`Ticker`** совпадает с классом **`Timeout`**. Пример:

```
Ticker tick;
tick.attach(&func, 2.0);
```

Рассмотрим пример программы:

```
#include "mbed.h"

Ticker ticker;
DigitalOut led(LED1);
InterruptIn btn(USER_BUTTON);

//флаг хранящий признак того, что в данный момент светодиод мигает
bool blinking = false;

//каждые 200мс переключаем состояние светодиода
void toggleLed() {
    led = !led;
}

//обработчик нажатия на кнопку
void onButtonClick() {
    blinking = !blinking;   //меняем значение флага на противоположное
    if(blinking) {          //если перешли в состояние - мигать
        ticker.attach(&toggleLed, 0.2); //то подключаем функцию мигания к тикеру
    }
    else {                  //иначе
        ticker.detach();    //отключаем функцию от тикера
    }
}

int main() {
    btn.rise(&onButtonClick);
    while (1) { }
}
```

Работа программы очевидна. После вызова метода **attach** – функция **toggleLed** начинает вызываться каждые 200мс, а после вызова метода **detach** – прекращает вызываться.

Простое мигание светодиодами по маске

На прошлой работе у многих вызвало затруднение мигание светодиодами по маске. Разберем, как можно было сделать все немного проще, написав совсем немного кода.

В этом примере выполним мигание по маске следующего вида:

○○○○○○
○○○○○●
○○○○●●
○○○●●●
○○●●●●
○●●●●●
●●●●●●
○●●●●●
○○●●●●
○○○●●●
○○○○●●
○○○○○●
○○○○○○

Для управления выходами в Mbed можно использовать два подхода.

- Первый, тот который мы рассматривали в прошлых работах, заключается в использовании класса DigitalOut и управлением каждым светодиодом в отдельности.
- Второй способ – управлять группой светодиодов как одним целым, при помощи прямой записи в порт

Как известно, светодиоды подключаются к пинам (Pins) микроконтроллера, все пины объединяются в 16 или 32 разрядные порты ввода вывода (Ports, GPIO). Для работы с портами напрямую, в mbed используется

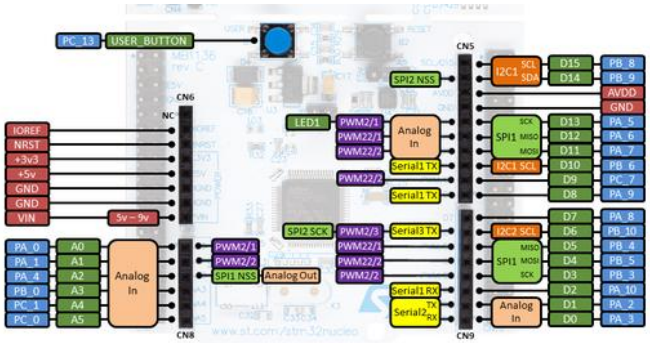
класс **PortOut**:

```
PortOut portA(PortA, 0xF3);
```

В конструкторе **PortOut** принимает 2 параметра.

1. Название порта с которым будет связан объект. В примере – Port A
2. Маска пинов, бинарная маска единицы в которой соответствуют номерам пинов, которыми сможет управлять объект.

Рассмотрим нашу плату:



Синим на схеме подписаны пины выведеные к коннекторам, которые можем использовать. Для того чтобы помигать 6 светодиодами нам нужно выбрать 6 пинов относящихся к одному порту. Возьмем порт А и его пины 0, 1, 4, 5, 6, 7. Мы не можем использовать пины 2, 3, так как они используются для связи по UART с ПК. Подключим к этим выводам последовательно планку со светодиодами, не забывая подключить землю.

Теперь нужно номера пинов преобразовать в битовую маску. В этой маске мы должны установить 1 в разрядах соответствующих номеру пина. Мы используем пины 0, 1, 4, 5, 6, 7, запишем это в бинарном виде:

$$1_7 1_6 1_5 1_4 0_3 0_2 1_1 1_0$$

Теперь, для удобства записи переведем это в 16 систему:

$$0xF3$$

Соответственно, для инициализации объекта порта нам нужно написать:

```
PortOut portA(PortA, 0xF3);
```

После этого, записывая в порт числа, мы сможем включать и выключать светодиоды в соответствии с бинарной записью числа. Например:

```
portA = 1;
```

включит первый светодиод.

```
portA = 2;
```

включит второй светодиод, так как двоичная запись выглядит как 10

```
portA = 0x10;
```

включит третий светодиод, так как двоичная запись выглядит как 10000, а третий светодиод подключен к пину PA.4 (считая с нуля)

Можно включать несколько светодиодов одновременно:

```
portA = 0x12;
```

включит второй и третий светодиоды, так как двоичная запись – 10010.

И так далее.

Теперь мы можем одним шагом зажигать и гасить все нужные светодиоды. Мы можем посчитать все возможные состояния требуемые для выполнения маски, для этого нужно каждый шаг маски записать в бинарном виде по номерам светодиодов, и перевести в шестнадцатеричный вид, получится следующая

таблица:

```
0x00, 0x01, 0x03, 0x13, 0x33, 0x73, 0xf3, 0x73, 0x33, 0x13, 0x03, 0x01 0x00
```

Если мы будем по очередно записывать эти числа в порт, делая паузу, то получим желаемое мигание светодиодов. Например:

```
portA = 0x00;
wait(0.1);
portA = 0x01;
wait(0.1);
portA =  0x03;
wait(0.1);
portA = 0x13;
wait(0.1);
portA = 0x33;
и так далее.
```

Код получается все еще громоздкий и не очень приятный. Чтобы его улучшить мы можем использовать массивы. Создадим массив, который будет содержать все возможные состояния маски мигания и в цикле будем проходить его по кругу:

```
int mask[] = { 0x00, 0x01, 0x03, 0x13, 0x33, 0x73, 0xf3, 0x73, 0x33, 0x13, 0x03, 0x01, 0x00 };

while(1) {
    portA = mask[i];
    i++;
    if(i >= sizeof(mask) / sizeof(mask[0]))
          i = 0; //переходим в начало
}
```

С помощью такого нехитрого метода мы сократили код программы в несколько раз. А самое полезное в этом то, что теперь для того чтобы изменить маску мигания достаточно исправить числа в массиве mask и светодиоды будут мигать по-другому. Это очень полезная техника, которая может иметь множество применений, а не только для мигания светодиодами.

Код программы целиком:

```
#include "mbed.h"

PortOut portA(PortA, 0xF3);

int mask[] = {0x00, 0x01, 0x03, 0x13, 0x33, 0x73, 0xf3,
                    0x73, 0x33, 0x13, 0x03, 0x01, 0x00};

int main() {
    int i = 0;
    while(1) {
        portA = mask[i];
        i++;
        if(i >= sizeof(mask) / sizeof(mask[0]))
            i = 0;
        wait(0.2);
    }
}

```

Задания

Вариант 1. Игра «Саймон говорит»

В этой игре игрок должен нажимать на кнопки в соответствии со случайно загорающимися светодиодами. Игра состоит из раундов и в каждом следующем раунде количество загорающихся светодиодов увеличивается на 1. Раунд состоит из двух фаз. В первой фазе компьютер по очереди зажигает несколько светодиодов. Во второй фазе игрок должен повторить эту очередность нажатиями на кнопки. Игрок проигрывает, если нажимает кнопки в неверном порядке. Игра ведется до проигрыша игрока.

Вариант 2. Игра «Ритм-ниндзя»

В этой игре игрок должен нажимать на кнопки в соответствии с загорающимися в случайном порядке светодиодами. При нажатии на кнопку светодиод должен гаснуть. В начале, светодиоды зажигаются медленно (напр. 1 раз в 2 секунды), и по ходу игры скорость увеличивается. Игрок проигрывает, если не успевает нажать на нужную кнопку до загорания следующего светодиода, либо нажимает на кнопку соответствующую выключенному светодиоду. Каждое включение светодиода должно сопровождаться короткой звуковой индикацией.

Вариант 3. Игра «Быстрый Джо»

Игра для 2-4 игроков. Нажатием USER_BUTTON запускается очередной раунд. После запуска программа через случайное время (5-10 с) зажигает светодиод на отладочной плате. Игроки нажимают кнопку, побеждает тот, кто нажал кнопку раньше всех, загорается светодиод на панели светодиодов соответствующий номеру игрока. При фальстарте светодиод игрока начинает часто мигать.

Вариант 4. Игра «Самый Быстрый Джо»

Нажатием USER_BUTTON запускается очередной раунд. После запуска программа через случайное время (5-10 с) зажигает светодиод на отладочной плате. Игроки начинают часто нажимать на кнопку. Через случайное время (5-10 с) светодиод гаснет. Побеждает тот, кто нажал кнопку большее число раз. Игрок, нажимавший кнопку до или после загорания светодиода, считается проигравшим.

Вариант 5. «Секундомер»

Необходимо создать секундомер, который отображает время прошедшее с момента нажатия на кнопку USER_BUTTON. Время должно отображаться с точностью до 0,1 секунды в двоичном виде на двух 6-светодиодных панелях. Повторное нажатие на кнопку останавливает счет.

Вариант 6. «Регулятор громкости с индикацией»

Вам необходимо разработать регулятор громкости с возможностью увеличения, уменьшения громкости и быстрого отключения с индикацией текущего состояния на светодиодах. Управление производится при помощи поворотного энкодера.

Условия работы:

- Имеется 6 уровней громкости отображаемой на светодиодах.
    - 0 включенных светодиодов – 0% громкость
    - 6 включенных светодиодов – 100% громкость
- Поворот по часовой стрелке увеличивает громкость, поворот против часовой стрелки – уменьшает.
- Нажатие на энкодер включает режим **“Mute”**, временное отключение звука. При нажатии светодиоды плавно гаснут по очереди. Затем первый светодиод начинает мигать раз в секунду. При повторном нажатии индикация плавно переходит в основной режим. При этом настроенный уровень громкости должен сохраняться.

Принцип работы энкодера.

У него имеется два выхода А и Б, при повороте ручки на каждом из них возникает импульс. При этом, при повороте в одну сторону импульс сперва возникает на ножке А, а через некоторое время на ножке Б, а при повороте в другую сторону – сперва на ножке Б, а затем на ножке А. Соответственно в зависимости от того, по какому выходу возникает импульс можно определить в какую сторону поворачивается ручка.

Третий выход работает как обычная кнопка.

Вариант 7. «Электронный дайс D6»

Вам необходимо разработать устройство выдающее случайное число (номер светодиода) от 1 до 6 по запросу. Условия работы:

- В исходном состоянии все светодиоды погашены
- По нажатию на кнопку светодиоды начинают последовательно мигать с частотой 100 Гц.
- Постепенно частота должна снижаться пока не остановится окончательно
- Каждый раз должно выпадать случайное число

Вариант 8. «Мигание»

К плате необходимо подключить 6 светодиодов и поворотный энкодер. При запуске программы все светодиоды мигают с одинаковой частотой – 1 Гц. Далее программа работает по следующему алгоритму:

- При нажатии на энкодер «выбирается» первый светодиод. Это индицируется горением светодиода в течении 3 с.
- При повороте энкодера выбранный светодиод меняет частоту мигания. При повороте по часовой стрелке частота мигания возрастает, при повороте против часовой стрелки частота мигания снижается
- При повторном нажатии выбирается следующий светодиод

Описание принципа работы энкодера см в варианте 6.

Вариант 9. «Мигание 2»

Подключить к контроллеру 6 светодиодов. После запуска симуляции по светодиодам должен пробегать огонек с нарастающей скоростью. По достижении определенной скорости, она должна начать снижаться, а потом снова возрастать. Использовать функцию wait нельзя.