Практическое занятие №2
Тема: «Типовые алгоритмы при работе с таймерами МК»

Цель работы: приобрести практические навыки по использованию таймеров с платой Arduino.

Последовательность выполнения работы:

– Изучить теоретические сведения, приведенные в практическом занятии.

– Собрать схемы на макетной плате, иначе при отсутствии набора Arduino в web-приложениях (https://wokwi.com/projects/new/arduino-uno или https://www.tinkercad.com/) для приведенных примеров.

– Запрограммировать микроконтроллер согласно тексту, указанному в примере.

– Выполнить задание для самостоятельной работы.

СОДЕРЖАНИЕ ОТЧЕТА

– Название практического занятия, его цель.

– Написанный программный код для скетчей: вставить в отчёт текстом, Courier New, 12 одинарный отступ без абзацев.

– Вывод о проделанной работе.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Плата Arduino позволяет быстро и минимальными средствами решить самые разные задачи. Но там где нужны произвольные интервалы времени (периодический опрос датчиков, высокоточные ШИМ сигналы, импульсы большой длительности) стандартные библиотечные функции задержки не удобны. На время их действия скетч приостанавливается и управлять им становится невозможно. В подобной ситуации лучше использовать встроенные AVR таймеры.

Что такое таймер?

Как и в повседневной жизни в микроконтроллерах таймер это некоторая вещь, которая может подать сигнал в будущем, в тот момент, который вы установите. Когда этот момент наступает, вызывается прерывание микроконтроллера, напоминая ему что-нибудь сделать, например выполнить определенный фрагмент кода.

Таймеры, как и внешние прерывания, работают независимо от основной программы. Вместо выполнения циклов или повторяющегося вызова задержки millis() или delay() вы можете назначить таймеру делать свою работу, в то время как ваш код делает другие вещи.

Итак, предположим, что имеется устройство, которое должно что-то делать, например мигать светодиодом каждые 5 секунд. Если не использовать таймеры, а писать обычный код, то надо установить переменную в момент зажигания светодиода и постоянно проверять не наступил ли момент ее переключения. С прерыванием по таймеру вам достаточно настроить прерывание, и затем запустить таймер. Светодиод будет мигать точно вовремя, независимо от действий основной программы.

Как работает таймер?

Он действует путем увеличения переменной, называемой счетным регистром. Счетный регистр может считать до определенной величины, зависящей от его размера. Таймер увеличивает свой счетчик раз за разом пока не достигнет максимальной величины, в этой точке счетчик переполнится и сбросится обратно в ноль. Таймер обычно устанавливает бит флага, чтобы дать знать, что переполнение произошло.

Можно проверять этот флаг вручную или сделать таймерный переключатель — вызывать прерывание автоматически в момент установки флага. Подобно всяким другим прерываниям вы можете назначить служебную подпрограмму прерывания (Interrupt Service Routine или ISR), чтобы выполнить заданный код, когда таймер переполнится. ISR сама сбросит флаг переполнения, поэтому использование прерываний обычно лучший выбор из-за простоты и скорости.

Чтобы увеличивать значения счетчика через точные интервалы времени, таймер надо подключить к тактовому источнику. Тактовый источник генерирует постоянно повторяющийся сигнал. Каждый раз, когда таймер обнаруживает этот сигнал, он увеличивает значение счетчика на единицу. Поскольку таймер работает от тактового источника, наименьшей измеряемой единицей времени является период такта. Если вы подключите тактовый сигнал частотой 1 МГц, то разрешение таймера (или период таймера) будет:

T = 1 / f (f это тактовая частота)
T = 1 / 1 МГц = 1 / 10^6 Гц
T = (1 ∗ 10^-6) с

Таким образом разрешение таймера одна миллионная доля секунды. Хотя вы можете применить для таймеров внешний тактовый источник, в большинстве случаев используется внутренний источник самого чипа.

Типы таймеров

В стандартных платах Arduino на 8 битном AVR чипе имеется сразу несколько таймеров. У чипов Atmega168 и Atmega328 есть три таймера Timer0, Timer1 и Timer2. Они также имеют сторожевой таймер, который можно использовать для защиты от сбоев или как механизм программного сброса. Вот некоторые особенности каждого таймера.

Timer0:

Timer0 является 8 битным таймером, это означает, что его счетный регистр может хранить числа вплоть до 255 (т. е. байт без знака). Timer0 используется стандартными временными функциями Arduino такими как delay() и millis(), так что лучше не запутывать его если вас заботят последствия.

Timer1:

Timer1 это 16 битный таймер с максимальным значением счета 65535 (целое без знака). Этот таймер использует библиотека Arduino Servo, учитывайте это если применяете его в своих проектах.

Timer2:

Timer2 — 8 битный и очень похож на Timer0. Он используется в Arduino функции tone().

Timer3, Timer4, Timer5:

Чипы ATmega1280 и ATmega2560 (установлены в вариантах Arduino Mega) имеют три добавочных таймера. Все они 16 битные и работают аналогично Timer1.

Конфигурация регистров

Для того чтобы использовать эти таймеры в AVR есть регистры настроек. Таймеры содержат множество таких регистров. Два из них — регистры управления таймера/счетчика содержат установочные переменные и называются TCCRxA и TCCRxB, где x — номер таймера (TCCR1A и TCCR1B, и т. п.). Каждый регистр содержит 8 бит и каждый бит хранит конфигурационную переменную. Вот сведения из даташита Atmega328:

TCCR1A								
Бит	7	6	5	4	3	2	1	0
0x80	COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10
ReadWrite	RW	RW	RW	RW	R	R	RW	RW
Начальное значение	0	0	0	0	0	0	0	0

TCCR1B								
Бит	7	6	5	4	3	2	1	0
0x81	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
ReadWrite	RW	RW	R	RW	RW	RW	RW	RW
Начальное значение	0	0	0	0	0	0	0	0

Наиболее важными являются три последние бита в TCCR1B: CS12, CS11 и CS10. Они определяют тактовую частоту таймера. Выбирая их в разных комбинациях можно приказать таймеру действовать на различных скоростях. Вот таблица из даташита, описывающая действие битов выбора:

CS12	CS11	CS10	Действие
0	0	0	Нет тактового источника (Timer/Counter остановлен)
0	0	1	clk_io/1 (нет деления)
0	1	0	clk_io/8 (делитель частоты)
0	1	1	clk_io/64 (делитель частоты)
1	0	0	clk_io/256 (делитель частоты)
1	0	1	clk_io/1024 (делитель частоты)
1	1	0	Внешний тактовый источник на выводе T1. Тактирование по спаду

По умолчанию все эти биты установлены на ноль.

ЗАДАНИЕ

Допустим необходимо, чтобы Timer1 работал на тактовой частоте с одним отсчетом на период. Когда он переполнится, вы хотите вызвать подпрограмму прерывания, которая переключает светодиод, подсоединенный к ножке 13, в состояние включено или выключено. Для этого примера запишем Arduino код, но будем использовать процедуры и функции библиотеки avr-libc всегда, когда это не делает вещи слишком сложными.

Сначала инициализируем таймер:

```
// avr-libc library includes
#include <avr/io.h>
#include <avr/interrupt.h>
#define LEDPIN 13

void setup()
{
    pinMode(LEDPIN, OUTPUT);
    // инициализация Timer1
    cli(); // отключить глобальные прерывания
    TCCR1A = 0; // установить TCCR1A регистр в 0
    TCCR1B = 0;
    // включить прерывание Timer1 overflow:
    TIMSK1 = (1 << TOIE1);
    // Установить CS10 бит так, чтобы таймер работал при
тактовой частоте:
    TCCR1B |= (1 << CS10);
    sei();  // включить глобальные прерывания
}
```

Регистр TIMSK1 это регистр маски прерываний Таймера/Счетчика1. Он контролирует прерывания, которые таймер может вызвать. Установка бита TOIE1 приказывает таймеру вызвать прерывание когда таймер переполняется. Подробнее об этом позже.

Когда вы устанавливаете бит CS10, таймер начинает считать и, как только возникает прерывание по переполнению, вызывается ISR(TIMER1_OVF_vect). Это происходит всегда когда таймер переполняется.

Дальше определим функцию прерывания ISR:

```
ISR(TIMER1_OVF_vect)
{
    digitalWrite(LEDPIN, !digitalRead(LEDPIN));
}
```

Сейчас мы можем определить цикл loop() и переключать светодиод независимо от того, что происходит в главной программе. Чтобы выключить таймер, установите TCCR1B=0 в любое время.

Как часто будет мигать светодиод?

Timer1 установлен на прерывание по переполнению и давайте предположим, что вы используете Atmega328 с тактовой частотой 16 МГц. Поскольку таймер 16-битный, он может считать до максимального значения (2^16 – 1), или 65535. При 16 МГц цикл выполняется 1/(16 ∗ 10^6) секунды или 6.25e-8 с. Это означает что 65535 отсчетов произойдут за (65535 ∗ 6.25e-8 с) и ISR будет вызываться примерно через 0,0041 с. И так раз за разом, каждую четырехтысячную секунды. Это слишком быстро, чтобы увидеть мерцание.

Если мы подадим на светодиод очень быстрый ШИМ сигнал с 50% заполнением, то свечение будет казаться непрерывным, но менее ярким чем обычно. Подобный эксперимент показывает удивительную мощь микроконтроллеров — даже недорогой 8-битный чип может обрабатывать информацию намного быстрей чем мы способны обнаружить.

Делитель таймера и режим CTC

Чтобы управлять периодом, можно использовать делитель, который позволяет поделить тактовый сигнал на различные степени двойки и увеличить период таймера. Например, вы бы хотели мигания светодиода с интервалом одна секунда. В регистре TCCR1B есть три бита CS устанавливающие наиболее подходящее разрешение. Если установить биты CS10 и CS12 используя:

```
TCCR1B |= (1 << CS10);
TCCR1B |= (1 << CS12);
```

Частота тактового источника поделится на 1024. Это дает разрешение таймера 1/(16 ∗ 10^6 / 1024) или 6.4e-5 с. Теперь таймер будет переполняться каждые (65535 ∗ 6.4e-5с) или за 4,194с. Это слишком долго.

Но есть и другой режим AVR таймера. Он называется сброс таймера по совпадению или CTC. Вместо счета до переполнения, таймер сравнивает свой счетчик с переменой которая ранее сохранена в регистре. Когда счет совпадет с этой переменной, таймер может либо установить флаг, либо вызвать прерывание, точно так же, как и в случае переполнения.

Чтобы использовать режим CTC надо понять, сколько циклов вам нужно, чтобы получить интервал в одну секунду. Предположим, что коэффициент деления по-прежнему равен 1024.

Расчет будет следующий:

```
(target time) = (timer resolution) * (# timer counts + 1)

(# timer counts + 1) = (target time) / (timer resolution)
(# timer counts + 1) = (1 s) / (6.4e-5 s)
(# timer counts + 1) = 15625
(# timer counts) = 15625 - 1 = 15624
```

Необходимо добавить дополнительную единицу к числу отсчетов потому что в CTC режиме при совпадении счетчика с заданным значением он сбросит сам себя в ноль. Сброс занимает один тактовый период, который надо учесть в расчетах. Во многих случаях ошибка в один период не слишком значима, но в высокоточных задачах она может быть критичной.

Функция настройки setup() будет такая:

```
void setup()
{
    pinMode(LEDPIN, OUTPUT);
    // инициализация Timer1
    cli(); // отключить глобальные прерывания
    TCCR1A = 0; // установить регистры в 0
    TCCR1B = 0;
    OCR1A = 15624; // установка регистра совпадения
    TCCR1B |= (1 << WGM12); // включение в CTC режим
    // Установка битов CS10 и CS12 на коэффициент деления
1024
    TCCR1B |= (1 << CS10);
    TCCR1B |= (1 << CS12);
    TIMSK1 |= (1 << OCIE1A);  // включение прерываний по
совпадению
    sei(); // включить глобальные прерывания
}
```

Также нужно заменить прерывание по переполнению на прерывание по совпадению:

```
ISR(TIMER1_COMPA_vect)
{
     digitalWrite(LEDPIN, !digitalRead(LEDPIN));
}
```

Сейчас светодиод будет зажигаться и гаснуть ровно на одну секунду. А в цикле loop() можно делать что-угодно. Пока вы не измените настройки таймера, программа никак не связана с прерываниями. У вас нет ограничений на использование таймера с разными режимами и настройками делителя.

Программный код:

```
// Arduino таймер CTC прерывание
// avr-libc library includes
#include <avr/io.h>
#include <avr/interrupt.h>
#define LEDPIN 13

void setup()
{
    pinMode(LEDPIN, OUTPUT);

    // инициализация Timer1
    cli();  // отключить глобальные прерывания
    TCCR1A = 0;   // установить регистры в 0
    TCCR1B = 0;

    OCR1A = 15624; // установка регистра совпадения

    TCCR1B |= (1 << WGM12);  // включить CTC режим
    TCCR1B |= (1 << CS10); // Установить биты на коэффициент
деления 1024
    TCCR1B |= (1 << CS12);

    TIMSK1 |= (1 << OCIE1A);  // включить прерывание по
совпадению таймера
    sei(); // включить глобальные прерывания
}

void loop()
{
```

```
    // основная программа
}

ISR(TIMER1_COMPA_vect)
{
    digitalWrite(LEDPIN, !digitalRead(LEDPIN));
}
```

САМОСТОЯТЕЛЬНАЯ РАБОТА

Ответьте на контрольные вопросы:

1. Что такое таймер в Arduino?
2. Какие типы таймеров существуют в Arduino?
3. Как настроить таймер на определённое время?
4. Можно ли использовать таймер для генерации сигналов определённой частоты?