Апаратні та програмні переривання

М.А. Ходукін

Дана стаття призначена для студентів освітньо-кваліфікаційного рівня «бакалавр» спеціальності 121 «Інженерія програмного забезпечення» денної та заочної форм навчання при виконанні лабораторних робіт з дисципліни «Архітектура комп'ютера та вбудовані мікропроцесорні системи з використанням Arduino», також може використовуватися при проведенні занять у предметно-науковому гуртку «Конструювання на Arduino».

У реальній програмі необхідно одночасно здійснювати багато різних дій. Крім того часто необхідно проводити реакцію на зовнішні події, вимірювати інтервали часу, тощо. Всі такі операції виконуються циклічно і паралельно, із різними періодами циклів, жодну із них не можна призупинити.

Зручним методом роботи із такими подіями буде режим переривань. У такому режимі за сигналом запиту переривання робота основної програми призупиняється, а управління передається підпрограмі обслуговування переривання. Після обслуговування переривання управління передається основній програмі і вона виконується із місця зупинки. Для основної програми виконання підпрограми обслуговування переривань невидиме. Короткий час роботи такої підпрограми не буде впливати на виконання основної програми.

Контролери ATMega мають багато різних джерел переривань частина котрих використана у середовищі Arduino. Наприклад можна так настроїти систему, щоб кожні 2 мс викликалась підпрограма обслуговування переривання таймера 2, що запускає підпрограму користувача. Встановлення режиму і часу спрацювання таймера Arduino проводиться зазвичай через апаратні регістри мікроконтролера. Для полегшення програмування існує безліч бібліотек, що полегшують використання переривань.

Наприклад бібліотека **MsTimer2** призначена для конфігурування переривання від таймера 2 мікроконтролера. Вона має всього три функції:

MsTimer2 :: set (unsigned long ms, void (*f) ( ))

Функція встановлює час переривання у мс. З таким періодом буде викликатись функція обробник переривання **f( )**. Вона повинна бути об'явлена як void (не повертає нічого) та не мати аргументів.

MsTimer2 :: start() – функція дозволяє переривання від таймера.

MsTimer2 :: stop() – функція забороняє переривання від таймера.

Простий приклад з паралельною обробкою сигналу можна розглянути на вже знайомої нам кнопки. В скетчі, що виконує аналогічні функції, але використовує переривання, функція **setup( )** задає час циклу переривання за таймером 2 мс і вказує ім'я обробника переривання **timerInterrupt**. Функція обробки сигналу кнопки **button1.scanState( )** викликається в обробнику переривання таймера кожні 2 мс.

Таким чином, стан кнопки оброблюється паралельним процесом. А в основному циклі програми перевіряється ознака кліка кнопки та змінюється

стан світлодіода. Сам скетч, за умови використання бібліотек[1], виглядає досить легким та зрозумілим.

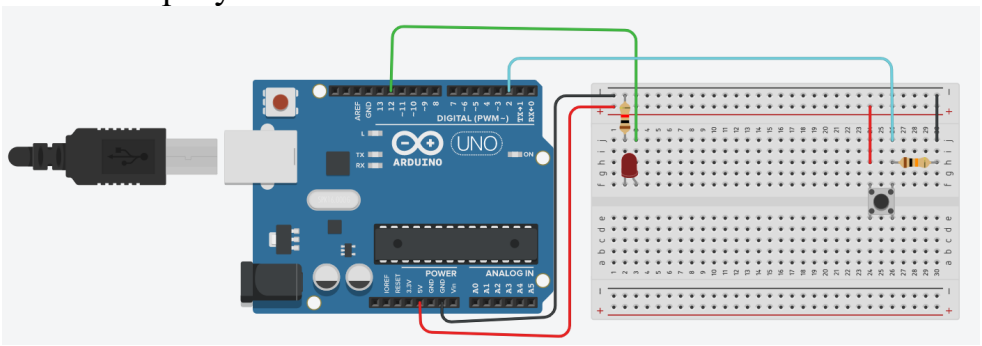

Рис. 4.29 Схема керування світлодіодом за допомогою кнопки

Приклад коду_1:

```
//  Нажатие на кнопку меняет состояние светодиода
#include <MsTimer2.h> // подключаем библиотеку обработеки прерывания от таймера
#include <Button2.h> // подключаем библиотеку работы с кнопкой (см. лекцію)
#define LED 6     // светодиод подключен к выводу 6
#define BUTTON 2  // кнопка подключена к выводу 2
Button button(BUTTON, 15);  // создание объекта - кнопка

void setup() {
        pinMode(LED, OUTPUT);                   // определяем вывод светодиода как выход
        MsTimer2::set(2, timerInterupt);        // задаем период прерывания по таймеру 2 мс
        MsTimer2::start();                      // разрешаем прерывание по таймеру
}
void loop() {                                   // управление светодиодом
        if ( button.flagClick == true ) {       // был клик кнопки
                button.flagClick= false;        // сброс признака
                digitalWrite(LED, ! digitalRead(LED));  // инверсия состояния светодиода
        }
}
// обработчик прерывания
void  timerInterupt() {
        button.scanState();       // вызов метода ожидания стабильного состояния для кнопки
}
```

Інший приклад, коли для визначення моменту деякої вхідної події (наприклад, натискання кнопки) використовується такий код:

```
void loop() {
                if (digitalRead(inputPin) ==LOW) {
                  // Выполнить какие-то действия
                }
}
```

Цей код постійно перевіряє рівень напруги на контакті **inputPin**, і коли **digitalRead** повертає **LOW**, виконуються якісь дії, позначені коментарем *// Выполнить какие-то действия*. Це цілком робоче рішення, але якщо всередині функції **loop** потрібно виконати велику кількість інших операцій, то всі ці операції займуть потрібен час, тому є можливість проґавити[2] коротке

[1] Приклад створення та використання бібліотек розглядалося на четвертої лекції.

[2] Насправді пропустити факт натискання на кнопку майже неможливо, тому що за мірками мікроконтролера вона залишається дуже довго натиснутою.

натискання на кнопку, поки процесор буде зайнятий чимось іншим.

А ще є короткі імпульси від датчика, які можуть тривати мільйонні частки секунди. Ось для прийому таких подій і слід використовувати **апаратні переривання** (**hardware interrupts**). Саме вони визначають функції, які будуть викликатись за цими подіями, незалежно від того, чим зайнятий мікроконтролер.

В Arduino UNO лише два контакти пов'язані з апаратними перериваннями, через що вони використовуються дуже економно. У Leonardo таких контактів п'ять, на великих платах, таких як Mega2560, їх набагато більше, а в Due всі контакти підтримують можливість переривання.

Табл. 4.4. Контакти апаратних переривань плат Arduino

	INT 0	**INT 1**	**INT 2**	**INT 3**	**INT 4**	**INT 5**
ATmega 328/168 (Nano, UNO, Mini)	D2	D3	–	–	–	–
ATmega 32U4 (Leonardo, Micro)	D3	D2	D0	D1	D7	–
ATmega 2560 (Mega)	D2	D3	D21	D20	D19	D18
Due	–	–	–	–	–	–

Команди роботи з перериваннями:

attachInterrupt(pin, function, state); – підключить переривання
detachInterrupt(pin); – вимкнути переривання

Ще раз звернемося до схеми на Рис. 4.29. Доки кнопка не натиснута на контакті D2 "висить" LOW, а в той момент, коли контакти змикаються рівень напруги на ньому підніметься до HIGH.

Приклад коду_2:

```
int led = 12;
void setup() {
        pinMode(led, OUTPUT);
        attachInterrupt(0, button, HIGH);
}
void loop() {
                // Выполнить какие-то действия
}
void button() {
        digitalWrite(led,  HIGH);
}
```

Окрім налаштування контакту LED – *pinMode(led, OUTPUT);* на роботу в режимі цифрового виходу, функція *setup* за допомогою ще одного рядка – *attachInterrupt(0, button, HIGH);* пов'язує функцію *button()* з перериванням "*0"*. Тепер у відповідь на кожне переривання автоматично буде викликатись ця функція.

Розглянемо більш детально аргументи функції, що тут викликається:

attachInterrupt(0, button, HIGH);

Перший аргумент – *0*. Це номер переривання. В Arduino UNO переривання 0 пов'язане з контактом D2, а переривання 1 – з контактом D3. В інших моделях Arduino ці переривання можуть бути пов'язані з іншими контактами (див. табл. 4.4).

Другий аргумент – *button.* Це ім'я функції, що повинна викликатись для обробки переривання. Вона визначається далі у скетчі. До таких функцій (підпрограм обробки переривань – **Interrupt Service Routine, ISR**), діють особливі вимоги. Вони не можуть мати параметри і нічого не повинні повертати. У цьому є певний зміст: навіть при тому, що вони викликаються в різних місцях у скетчі, немає жодного рядка коду, що здійснює прямий виклик **ISR**, тому немає жодної можливості передати їм параметри або отримати значення, що повертається.

Останній параметр функції *attachInterrupt* — це константа (в нашому випадку HIGH). Вона означає, що підпрограма обробки переривання буде викликатись лише за рівень напруги на контакті D2 HIGH (що і відбудеться в момент натискання кнопки). Взагалі існує кілька режимів (див. табл. 4.5).

Найчастіше використовуються режими переривань RISING (за позитивним перепадом) та FALLING (за негативним перепадом).

Табл. 4.5. Режими обробки переривань плат Arduino

Режим	Дія	Опис
LOW	Переривання генерується при рівні напруги LOW	У цьому режимі підпрограма обробки переривань буде викликатись постійно, поки на контакті зберігається низький рівень напруги
RISING	Переривання генерується при перепаді напруги з рівня LOW до рівня HIGH	—
FALLING	Переривання генерується при перепаді напруги з рівня HIGH до рівня LOW	—
HIGH	Переривання генерується при рівні напруги HIGH	Цей режим підтримується тільки в моделі Arduino Due, він так як і режим LOW, рідко використовується на практиці
CHANGE	Переривання генерується за будь-якої зміни рівня сигналу	