Лабораторная работа № 3

Прерывания в Arduino

Цель лабораторной работы: познакомиться с прерываниями на платформе Arduino. Научиться настраивать и использовать прерывания в своей программе.

Теоретическая часть

Прерывание - сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания. Прерывание извещает процессор о наступлении высокоприоритетного события, требующего прерывания текущего кода, выполняемого процессором. Процессор отвечает приостановкой своей текущей активности, сохраняя свое состояние и выполняя функцию, называемую обработчиком прерывания (или программой обработки прерывания), которая реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код (Рисунок 1).

Установив обработчик прерываний в программе, контроллер сможет реагировать на включение или выключение кнопки, нажатие клавиатуры, мышки, получение новых данных по UART, I2C или SPI. Такие прерывания называются внешними **аппаратными.**

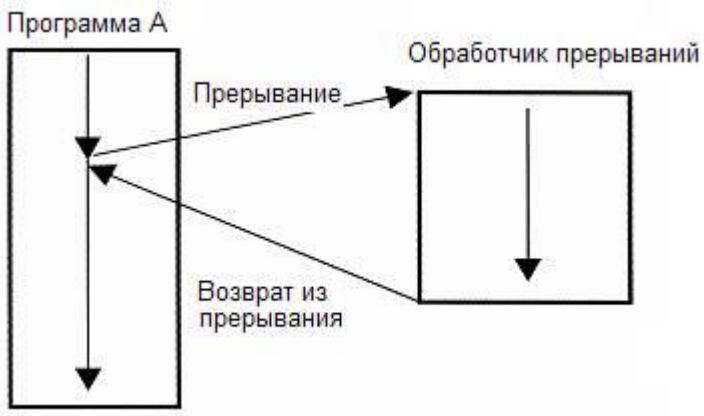

Рисунок 1 – Диаграмма работы прерывания

External hardware interrupt – это прерывание, вызванное изменением сигнала на pin микроконтроллера. В случае использования возможностей внешних аппаратных прерываний вычислительное ядро микроконтроллера не занимается опросом pin и не тратит на это время. Этим занимается отдельный модуль микроконтроллера, отвечающий за обработку сигналов прерываний. Как только сигнал на пине изменяется – микроконтроллер получает сигнал и немедленно приступает к обработке прерывания, а затем возвращается к продолжению основной работы.

Чаще всего прерывания используются для детектирования коротких событий – импульсов, или даже для подсчёта их количества, не нагружая основной код. Аппаратное прерывание может поймать короткое нажатие кнопки или срабатывание датчика во время сложных долгих вычислений или задержек в коде. Например, pin опрашивается **независимо от основного кода**. Также прерывания могут будить микроконтроллер из режимов энергосбережения, когда большая часть периферийных модулей отключена.

В Arduino аппаратные прерывания могут генерировать только определённые пины (Таблица 1):

Таблица 1 – Распиновка прерываний Arduino

МК / номер прерывания	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

Поскольку прерывания имеют свой номер, который отличается от номера пина, то бывает полезно пользоваться функцией:

digitalPinToInterrupt(pin)

Данная функция принимает в качестве параметра **pin** - номер пина, который настроен на прерывание и возвращает номер прерывания.

Подключается прерывание при помощи функции:

attachInterrupt(pin, handler, mode) , где

pin – номер прерывания
handler – имя функции-обработчика прерывания
mode – режим работы прерывания:
LOW – срабатывает при низком уровне сигнала на пине
RISING – срабатывает при изменении сигнала на пине с 0 на 1
FALLING – срабатывает при изменении сигнала на пине с 1 на 0
CHANGE– срабатывает при любых изменениях сигнала

Также прерывание можно отключить при помощи функции

detachInterrupt(pin) , где

pin – номер прерывания.

Для глобального запрета прерывания существует функция:

noInterrupts()

А снова разрешить прерывания можно при помощи функции:

interrupts()

Замечание! Функция noInterrupts() остановит также прерывания таймеров, при этом перестанут работать все функции времени и генерация ШИМ.

При работе с прерываниями нужно обязательно учитывать следующие важные ограничения:

- Функция – обработчик не должна выполняться слишком долго: Arduino не может обрабатывать несколько прерываний одновременно. Пока выполняется ваша функция-обработчик, все остальные прерывания не учтутся и можно пропустить важные события. В обработчике можно лишь установить флаг события, а в функции **loop()** – проверять флаг и обрабатывать его.
- Переменные, изменяемые в прерывании должны быть объявлены как **volatile (**например: volatile boolean btnState = 0**)**. Переменная должна быть объявлена **volatile**, когда её значение может быть изменено чем-либо за пределами того участка программы, где она объявлена, например, параллельно выполняющимся процессом. В Arduino единственным местом, где это может проявиться, является участок программы, ассоциированный с прерываниями, вызванный программой обработки прерываний.
- Не рекомендуется использовать большое количество прерываний (старайтесь не использовать более 6-8). Большое количество разнообразных событий требует серьезного усложнения кода и ведет к ошибкам.
- В обработчиках категорически нельзя использовать **delay()**. Механизм определения интервала задержки использует таймеры, а они тоже работают на прерываниях, которые заблокирует ваш обработчик. В итоге все будут ждать всех и программа зависнет. По этой же причине нельзя использовать протоколы связи, основанные

на прерываниях. Также не желательно использовать функции **millis()** и **micros()** - они не изменяются внутри обработчика прерываний.

Рассмотрим простой пример использования прерываний: определим функцию-обработчик, которая при изменении сигнала на pin 2 Arduino Uno переключит состояние pin 13, к которому подключен на плате светодиод.

Пример 1

```
#define PIN_LED 13

volatile boolean actionState = LOW;

void setup() {
     pinMode(PIN_LED, OUTPUT);
     attachInterrupt(0, myEventListener, CHANGE);
}

void loop() {
// В функции loop ничего не делаем, т.к. весь код обработки
событий будет в функции myEventListener
}

void myEventListener() {
     actionState != actionState; //
     digitalWrite(PIN_LED, actionState);
}
```

Рассмотрим второй пример, в котором в прерывании считаются нажатия кнопки, а в основном цикле они выводятся с задержкой в 1 секунду. Работая с кнопкой в обычном режиме, совместить такой вывод с задержкой – невозможно:

Пример 2

```
volatile int counter = 0;  // переменная-счётчик
void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(0, buttonTick, FALLING);
}

void buttonTick() {
  counter++;  // + нажатие
}

void loop() {
  Serial.println(counter);  // выводим
  delay(1000);              // ждём
}
```

Данный код считает нажатия даже во время задержки в 1 секунду.

Если прерывание отлавливает какое-то событие, которое можно обрабатывать не сразу, то лучше использовать следующий алгоритм:

- В обработчике прерывания поднимаем флаг
- В функции **loop()** проверяем флаг, если поднят – сбрасываем его и выполняем нужные действия.

Пример 3

```
volatile boolean iFlag = false;   // флаг
void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(0, btnFall, FALLING);
}

void btnFall() {
  iFlag = true;   // подняли флаг
}

void loop() {
  if (iFlag) {
    iFlag = false;    // сбрасываем флаг
    // совершаем действия
    Serial.println("Falling is done!");
  }
}
```

Прерывания по таймеру

Таймером называется счетчик, который производит счет с некоторой частотой, получаемой из процессорных 16 МГц. Можно произвести конфигурацию делителя частоты для получения нужного режима счета. Также можно настроить счетчик для генерации прерываний при достижении заданного значения.

Таймер и прерывание по таймеру позволяет выполнять прерывание один раз в миллисекунду. В Atmega328 имеется 3 таймера – **Timer0**, **Timer1** и Timer2. Timer0 используется для генерации прерываний один раз в миллисекунду, при этом происходит обновление счетчика, который передается в функцию **millis()**. Этот таймер является восьмибитным и считает

от 0 до 255. Прерывание генерируется при достижении значения 255. По умолчанию используется тактовый делитель на 65, чтобы получить частоту, близкую к 1 кГц.

Для сравнения состояния на таймере и сохраненных данных используются регистры сравнения. В данном примере код будет генерировать прерывание при достижении значения 0xAF на счетчике.

```
OCR0A = 0xAF;
TIMSK0 |= _BV(OCIE0A);
```

Требуется определить обработчик прерывания для вектора прерывания по таймеру. Вектором прерывания называется указатель на адрес расположения команды, которая будет выполняться при вызове прерывания. Несколько векторов прерывания объединяются в таблицу векторов прерываний. Таймер в данном случае будет иметь название TIMER0_COMPA_vect. В этом обработчике будут производиться те же действия, что и в loop ().

```
SIGNAL(TIMER0_COMPA_vect) {
unsigned long currentMillis = millis();

sweeper1.Update(currentMillis);

if(digitalRead(2) == HIGH) {
sweeper2.Update(currentMillis);
led1.Update(currentMillis);
}

led2.Update(currentMillis);
led3.Update(currentMillis);

}

//Функция loop () останется пустой.

void loop()

{

}
```

Практическая часть

1. Разработать схему и программу для устройства, которое будет иметь возможность изменять яркость светодиодов в зависимости от показаний датчика освещенности или аналогового ик-датчика расстояния. По нажатию на кнопку вызывать обработчик прерывания, который активирует следующий светодиод.

2. Разработать схему содержащую одноразрядный семисегментный индикатор, отсчитывающий нажатия на кнопку. Должны быть учтены одинарные и двойные нажатия на кнопку с использованием прерываний.

3. Реализовать программу одновременной работы мерцающего светодиода (частоту выбрать произвольную) и вращающегося сервопривода. По нажатию на кнопку (вызов прерывания) сервопривод должен обнулить положение.