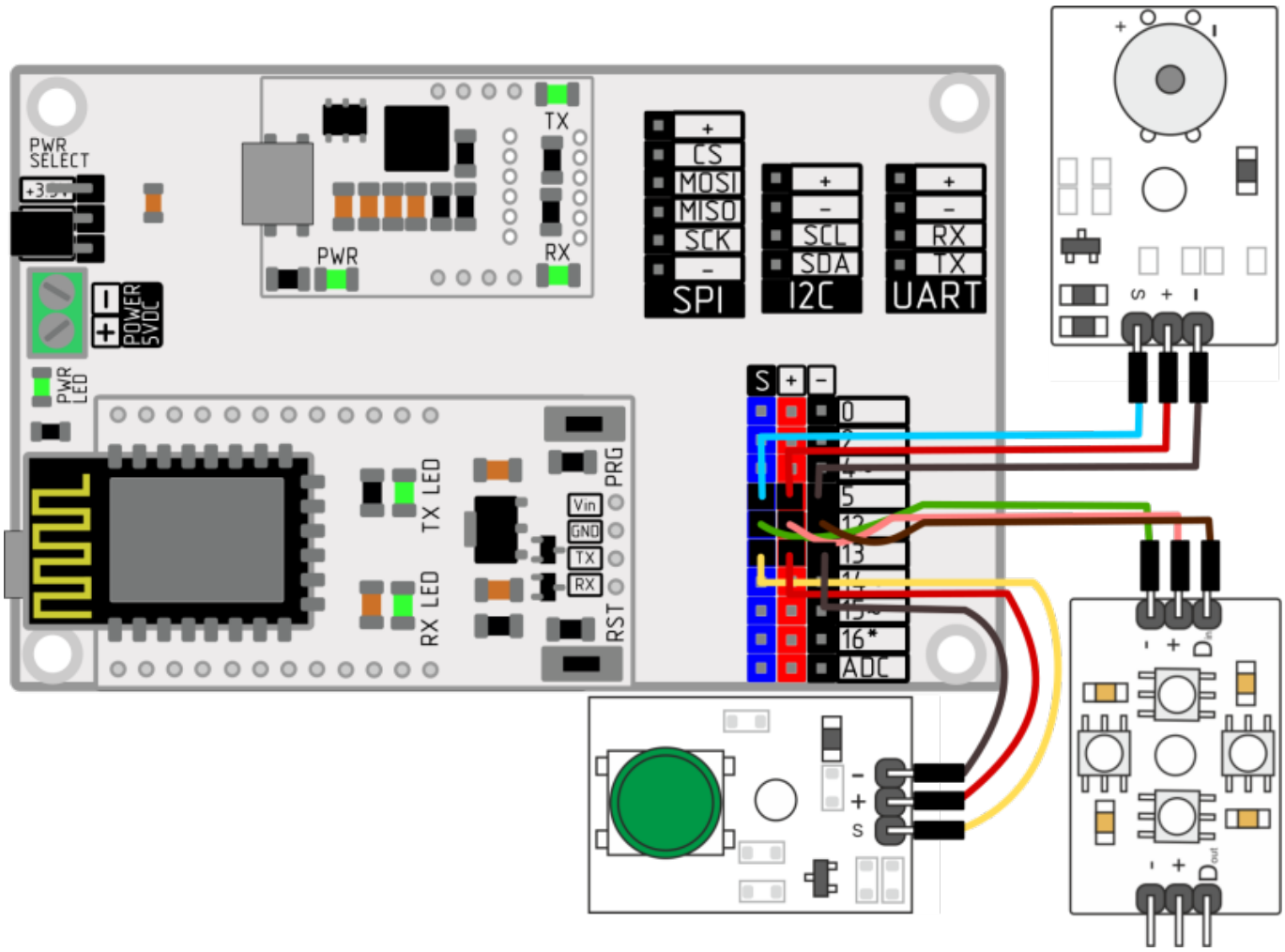
Урок 20. Время работы контроллера

Video

Схема сборки:



При выполнении скрипта в прошлом уроке, Вы, вероятно, заметили, что при нажатии на кнопку, часто фиксируется ни одно нажатие, а несколько. Это происходит из-за “дребезга”. То есть при нажатии на кнопку, ток не сразу стабилизируется, а в очень коротком промежутке времени, то появляется, то исчезает.

Самым простым способом избавиться от дребезга является засечь время, в момент когда кнопка была нажата, и проверить, если очередное нажатие появилось в течении очень короткого времени (например, 200 миллисекунд, или 0,2 секунды), значит считаем что это “дребезг” и не реагируем на него. В противном случае, считаем что произошло нажатие на кнопку.

Итак, чтобы понять, сколько прошло времени от старта работы контроллера, воспользуемся модулем время - `time` и его функцией `ticks_ms()`. Сохраним время в переменную:

```
start_time = time.ticks_ms()  # запоминаем начальное время
```

Затем в обработчике прерывания, “достанем” переменную времени работы.

```
global start_time  # "достать" переменную
```

Затем получим разницу между временем, которое запомнили, и текущим временем работы. Это делается с помощью функции `ticks_diff(время1, время2)`, где время1 - текущее время, а время2 - время старта.

```
delta = time.ticks_diff(time.ticks_ms(), start_time)
```

Если эта разница больше 200 миллисекунд, то в этом случае увеличим счётчик нажатий. Изменённый код предыдущего урока будет выглядеть следующим образом:

```
# импорт модулей
from machine import Pin
import time

# выводы к которым подключены платы:
but = Pin(13, Pin.IN)  # кнопка
irq_q = 0  # счётчик количества прерываний
start_time = time.ticks_ms()  # запоминаем начальное время

# функция, которая будет вызвана при возникновении прерывания
def callback(pin):

    global irq_q  # "достать" переменную
    global start_time  # "достать" переменную
    delta = time.ticks_diff(time.ticks_ms(), start_time)

    if delta > 200:
        irq_q = irq_q + 1  # увеличение количества прерываний
        print("Количество прерываний: ", irq_q)
        start_time = time.ticks_ms()

# подключение прерывания на вывод кнопки
```

```
but.irq(trigger=Pin.IRQ_RISING, handler=callback)

# основной цикл программы
while True:
    pass  # ничего не делать
```

Загрузите этот код на контроллер. Попробуйте теперь нажимать на кнопку, и увидите что кнопка стала работать стабильно. Счётчик тоже работает предсказуемо.

А теперь изменим код работы сирены (урок 16), чтобы она включалась и отключалась по нажатию кнопки. Мы также будем пользоваться прерываниями и будем учитывать время работы контроллера. Помимо этого введём переменную-состояние работы сирены (флаг). Если её значение истинно, то сирену нужно включить, если значение ложно - переменную нужно выключить.

Остальные принципы работы будут из предыдущих уроков. В обработчике прерывания “достанем” переменные, проверим время прошедшее с нажатия кнопки, и если оно больше 200 мс, изменим переменную-флаг. Код сирены будет следующим:

```
# импорт модулей
from neopixel import NeoPixel
from machine import Pin, PWM
import time

# выводы к которым подключены платы:
but = Pin(13, Pin.IN)  # кнопка
addr_leds = Pin(12)  # адресные светодиоды
buz = PWM(Pin(5, Pin.OUT))  # зуммер

# переменные для определения цветов
dark = (0, 0, 0)  # не горит ничего
red = (255, 0, 0)  # красный цвет
blue = (0, 0, 255)  # синий цвет
colors = [red, blue, red, blue]  # список цветов для каждого из 4 светодиодов

# создание списка светодиодов
leds = NeoPixel(addr_leds, 4)

# вспомогательные переменные
freqs = [494, 262]  # список с частотами нот для зуммера
start_time = time.ticks_ms()  # время начала работы
now_is_on = False  # флаг запуска сирены

# функция включения сирены
def turn_on():
    colors.reverse()  # развернуть список с цветами светодиодов
    freqs.reverse()  # развернуть список с частотами нот

    for led_number in range(4):  # цикл для установки цвета каждого светодиода
        leds[led_number] = colors[led_number]
    leds.write()  # включить светодиоды
```

```
    buz.freq(freqs[0])  # установить частоту зуммера
    buz.duty(512)  # установить заполнение зуммера в 512
    time.sleep(0.7)  # задержка на 0,7 секунд
    buz.duty(0)  # установить заполнение в 0
    time.sleep(0.3)  # задержка на 0,3 секунды


# функция выключения светодиодов и зуммера
def turn_off():
    for led_number in range(4):  # цикл по светодиодам
        leds[led_number] = dark
    leds.write()  # включить светодиоды
    buz.duty(0)  # установить заполнение в 0

# обработчик прерывания
def callback(pin):

    global now_is_on  # "достать" переменную
    global start_time  # "достать" переменную
    delta = time.ticks_diff(time.ticks_ms(), start_time)  # разность времени

    if delta > 200:
        now_is_on = not now_is_on  # сменить флаг
        start_time = time.ticks_ms()  # пересохранить время


# подключение прерывания к выводу кнопки
but.irq(trigger=Pin.IRQ_RISING, handler=callback)

# основной цикл программы
while True:

    if now_is_on:
        turn_on()
    else:
        turn_off()
```

Попробуйте теперь загрузить данный скрипт на контроллер и Вы увидите что сирену теперь можно включать и отключать по кнопке.

Запомнить:

- Узнать время с начала работы (в миллисекундах) контроллера можно из модуля time с помощью функции `ticks_ms()`
- Разница между временем работы вычисляется с помощью функции `ticks_diff()`.
- Существует функция , возвращающая время работы в микросекундах `ticks_us()`.

Предыдущий урок

Следующий урок