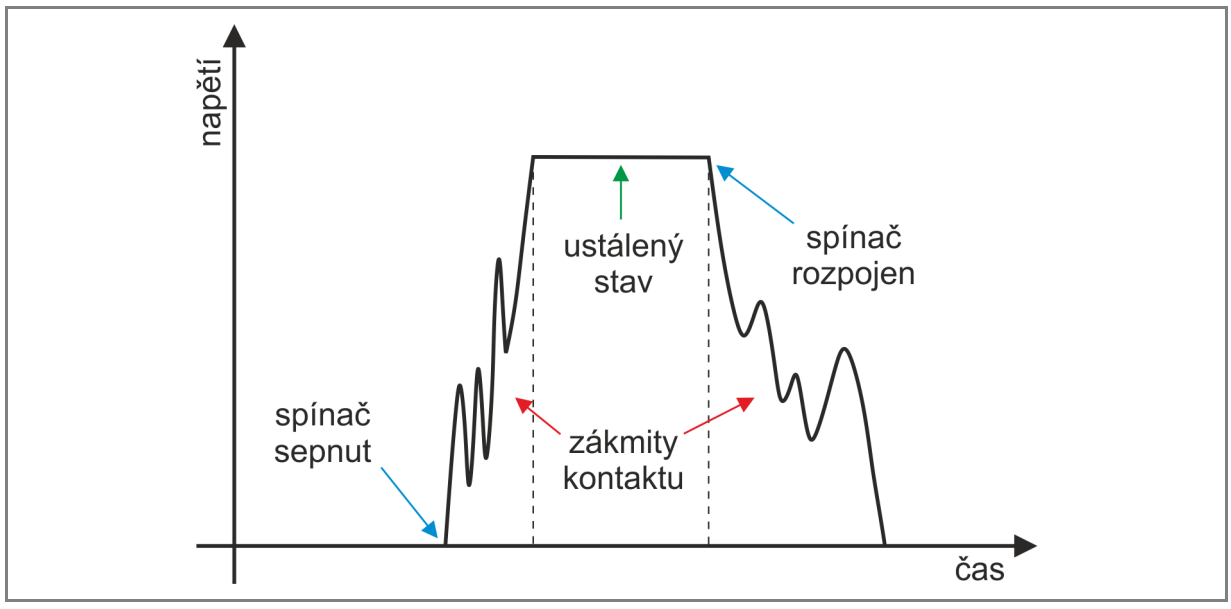
ButtonDebounce

Knihovna ButtonDebounce je určena pro ošetření zákmitů mechanických kontaktů spínačů.

Co je bouncing?

Mechanické kontakty při spínání generují zákmity (bouncing). Po změně stavu mechanického kontaktu, membránových tlačítek, mikrospínačů, jazýčkových kontaktů a podobných, nedojde okamžitě k trvalému sepnutí či rozepnutí kontaktů, ale zpočátku, po dobu jednotek až desítek milisekund, dochází ještě k zákmitům, při kterých je obvod kontaktem opakovaně spínán a opět rozpojován. Pokud spínač slouží k rozsvícení světla nebo aktivaci bzučáku, můžeme tyto počáteční zákmity ignorovat, dříve nebo později se kontakty „uklidní“ a výstup zůstane trvale sepnut nebo rozpojen. Pokud ale potřebujeme vyhodnotit například počet sepnutí daného spínače, je odstranění zákmitů nezbytností.

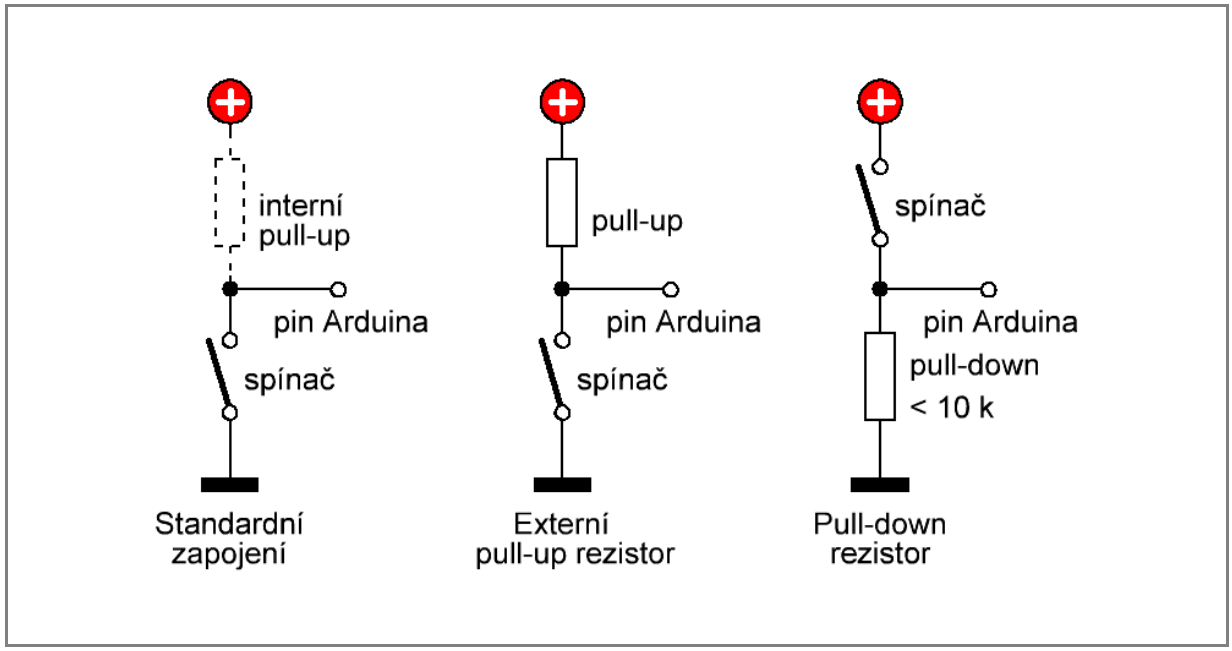


Knihovna spínač opakovaně vzorkuje a změnu jeho stavu vrátí jen v případě, kdy setrvá v novém stavu při osmi po sobě jdoucích kontrolách.

Třída ButtonDebounce

Konstruktor

Vytvoří jednu pojmenovanou instanci třídy ButtonDebounce, aktivuje pin pro připojení spínače a nastaví dobu prodlevy, potřebnou pro uklidnění zákmitů kontaktu spínače. Konstruktor zároveň připojí k určenému pinu interní pull-up rezistor.
Je očekáváno připojení spínače mezi vybraný pin a GND. Pokud potřebujete spínač připojit mezi pin a napájecí napětí, je nutno připojit také externí pull-down rezistor o hodnotě menší, než 10 kiloohmů.
Externí pull-up rezistor o hodnotě 1 až 10 kiloohmů se doporučuje připojit i v případě, že je v okolí spínače velké elektrické rušení, nebo je spínač vzdálen od Arduina.



Syntaxe:

```
ButtonDebounce button(pin, delay);
```

Parametry:

pin *(int)* – číslo pinu, na který bude připojen spínač.
Konstruktor k tomuto pinu při vytváření instance připojí interní pull-up rezistor.
delay *(unsigned log)* – čas, po který se budou testovat zákmity kontaktu v milisekundách (ms).

Příklad:

```
// spínač na pinu 10 s nastavenou prodlevou 100 ms
ButtonDebounce button(10, 100);
```

Členské funkce (metody)

update()

Výkonná funkce, která aktualizuje okamžitý stav spínače (sepnuto / rozpojeno).

Syntaxe:

```
button.update();
```

Parametry:

Funkce nemá žádné parametry.

Návratová hodnota:

Funkce nic nevrací.

Příklad:

```
void loop()
{
  buttonUp.update();  // dotaz na stav spínače, pojmenovaného buttonUp
  buttonDown.update();  // dotaz na stav spínače, pojmenovaného buttonDown
}
```

Poznámka:

Pro správnou činnost je nutno funkci update() volat samostatně pro každý spínač co možná nejčastěji, ideálně ve smyčce loop().

state()

Vrací logickou úroveň na pinu, ke kterému je spínač připojen.

Syntaxe:

```
button.state();
```

Parametry:

Funkce nemá žádné parametry.

Návratová hodnota:

0 = LOW

1 = HIGH

Příklad:

```
if(button.state() == LOW)  // LOW je totéž jako 0, HIGH totež jako 1
  {
    Serial.println("Sepnuto!");
  }
```

setCallback()

Pokud se změní stav spínače, bude volána funkce, určená parametrem.

Syntaxe:

```
button.setCallback(function);
```

Parametr:

function *(const int)* – jméno funkce, která bude volána při změně stavu spínače.

Návratová hodnota:

Funkce nic nevrací.

Příklad:

```
#include <ButtonDebounce.h>
ButtonDebounce button(10, 250);

void setup()
{
  Serial.begin(115200);
}

void loop()
{
  button.update();
  if(button.state() == LOW)
  {
    Serial.println("Sepnuto!");
```

```
  }
}
```

Nebo podobné s využitím volání uživatelem vytvořené funkce buttonChanged():

```
#include <ButtonDebounce.h>
ButtonDebounce button(10, 250);

void buttonChanged(const int state)
{
  Serial.println("Logicka uroven na vstupu je: " + String(state));
}

void setup()
{
  Serial.begin(115200);
  button.setCallback(buttonChanged);
}

void loop()
{
  button.update();
}
```