Arduino – tlačítka

NSWI170: Lab 03

Patrik Dokoupil

Credit: Martin Kruliš

Shrnutí 2. úlohy

- Problémy:
  1. „Hard-coded“ hodnoty
     - `for (int i = 0; i < 4; i++)`
     - `if (current_led == 3)`
  2. Opakované zhasnutí všech LED v každém volání **`loop`**
  3. Vícenásobné volání **`millis()`** v **`loop`**
  4. Zápis na neexistující pin
     - **`digitalWrite(pins[i])`**
  5. Komplikovaný design
     - **`switch-case`**/**`if-else`** s několika stavy (co když bude 20? 100?
  6. include <funshield.h>

Lze **`num_leds`** resp. **`num_leds - 1`**

V budoucích úlohách může vést na problémy s časováním

Pozor na hodnotu **`i`** !

Pravidlo č. 3

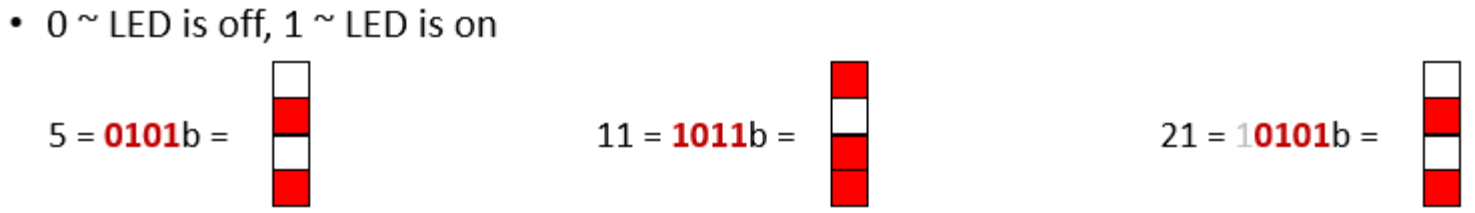
- # 3 **DRY – Don‘t Repeat Yourself**
  - Neopakujte (copy-paste) Váš kód
    - Když kód kopírujete, je obtížné v něm dělat změny (nutno na několika místech)
    - Úpravami pak snadno zanesete chyby (zapomenete změnit jeden z výskytů kódu)
    - Pokud je spousta téměř stejných bloků pouze s malým rozdílem, pak ten rozdíl většinou není vidět na první pohled
  - Raději používejte cykly/funkce apod.
    - Např. pro inicializaci pinů
  - Pozor, že chcete sjednocovat pouze podobný/stejný kód
    - Větší odlišnosti často vedou na spoustu if-else bloků, což není ideální
  - Detaily na webu

Rychlé zopakování práce s LED

- PINy je nutno inicializovat jako výstupní
  - `pinMode(led1_pin, OUTPUT);`
- Změnu (rozsvícení/zhasnutí) LED lze provést pomocí digitálního zápisu
  - `digitalWrite(led1_pin, ON);`
  - `digitalWrite(led1_pin, OFF);`
- Používejte konstanty z funshield.h!
  - Pro ON/OFF, i pro jednotlivé PINy

Cvičení

- T1 Bitový LED encoder
  - Napište funkci která na vstupu dostane číslo (`int`) a nastaví LED podle dolních 4 bitů tohoto čísla



  - Co se vám bude hodit:
    - Bitové operace: **&** (AND), **|** (OR), **~** (negace)
    - Bitové posuny: **<<** (posun vlevo) , **>>** (posun vpravo)
    - Arduino má funkce pro manipulaci s bity, je však vhodné výše zmíněné operace znát

Cvičení

- T2: Kontinuální počítání
  - Vytvořte counter který se neustále zvyšuje (modulo 16)
    - Zvýšení probíhá v rozumném intervalu (např. 1s)
    - Stejně jako na minulém cvičení -> nepoužívejte **`delay()`**!
  - Pomocí funkce z T1 pak využijte LED pro zobrazení hodnoty counteru

Tlačítka

- Podobně jako u LED bude nutné nastavit mód pinů
  - **`pinMode(button1_pin, INPUT)`**
- Stav tlačítka lze získat pomocí **`digitalRead`**
  - **`digitalRead(button1_pin)`**
  - Funguje zde inverzní logika, **`LOW`** znamená, že je tlačítko stisknuté
    - Konstanty z funshield.h
  - Stav je nutné kontrolovat v hlavním loopu
    - **Pozor**, že tlačítka nejsou perfektní a může docházet k tzv. „bouncingu“, tj. krátký oscilační efekt jako kdyby tlačítko bylo stisknuto a povoleno (klidně několikrát) v krátkém časovém okamžiku. Tento problém zatím můžete ignorovat.
      - Lze řešit např. tak, že přidáte „debouncing“ interval, který začne běžet po stisknutí tlačítka a během kterého ignorujete všechny změny.

Cvičení

- T3: Zvyšte counter když je tlačítko stisknuté
  - Counter se bude zvyšovat pouze když je první tlačítko stisknuté
  - Rychlost zvyšování zůstává stejná
- Drobné vylepšení: counter je 1x zvýšen právě v okamžiku stisku (změna stavu) a pak pokračuje v pravidelných intervalech
  - Začátek intervalu je v okamžiku změny stavu tlačítka

Cvičení

- T4
  - Counter již nechceme automaticky zvyšovat
  - Zvýšíme jej pouze 1x pokaždé když nastane stisk tlačítka
    - Zajímá nás změna stavu z „nestisknuto“ na „stisknuto“
    - Nápověda: poslední známý stav tlačítka si lze uložit do globální proměnné a pak sledovat jak se stav mění

Cvičení

- T5: Možnost snižovat counter
  - Pomocí druhého tlačítka umožněte snížit counter
  - Tedy bude se chovat stejně jako první tlačítko ale místo zvýšení provede snížení
  - Pozn. Chceme dekrement modulo 16, ten lze implementovat jako

    ```
    x = (x + 15) % 16
    ```

  - Zamyslete se nad tím, jestli mezi funkcí obou tlačítek vidíte nějaký vzor
    - Nejspíše uvidíte, že dělají +- totéž, snažte se zredukovat duplicity v kódu pomocí dekompozice do znovupoužitelných funkcí

Domácí úloha

- Counter s chytrými tlačítky
  - Stále platí, že po stisknutí budou tlačítka counter zvyšovat/snižovat
  - Pokud však bude tlačítko stisknuto po delší dobu, tak se bude counter neustále zvyšovat resp. Snižovat
    - Tedy budete mít 2 konstanty
      - 1. jak dlouho musí být tlačítko stisknuto než se sepne fáze kontinuálního snižování/zvyšování counteru.
      - 2. jaký bude interval změny
    - Nepoužívejte **`delay()`**!
    - Pozor že máte více než 1 tlačítko -> bude nutné časovat pro každé z nich zvlášť
  - Konstanty a detaily v ReCodExu

Cvičení*

- Extension 1: Větší counter
  - Counter bude mít hodnotu mezi 0 a 256
  - LEDky budou postupně zobrazovat dolní 4 bity a horní 4 bity counteru
    - Stisknutí tlačítka č. 3 změní zobrazení (dolní 4 bity -> horní 4 bity, resp. Naopak)
    - Pokud není tlačítko stisknuto, provede se změna automaticky vždy po uplynutí 1s
- Extension 2: Třetí tlačítko bude přepínat mezi jednotlivými implementacemi (chováním prvních dvou tlačítek) counteru - tedy těmi, které jsme implementovali v T3, T4 a domácí úloze.
- Extension 3: Debouncing – viz. slide 8
  - Zkuste experimentovat s různými délkami debouncing intervalu (1-20ms)

Lepší, čitelnější kód

- Globální proměnné
  - Jak již bylo zmíněno, většinou se jim chceme vyhnout
    - U Arduina to nejde úplně snadno kvůli rozdělení do **`setup()`** a **`loop()`** a nemožnosti mezi nimi něco sdílet
  - Rozumné použití glob. proměnných
    - Když už globální proměnné budete používat, tak:
      - Pouze pro uchování stavu který musí žít déle než jedno volání **`loop()`** (např. chcete si jej pamatovat i v další iteraci)
      - Ke glob. Proměnným přistupujte pouze v **`setup()`** a **`main()`** a do všech ostatních funkcí je dejte jako parametr (nebo referenci)
    - Globální konstanty (**`constexpr`**) jsou OK

Lepší, čitelnější kód

- Logické celky lze uskupit do struktur
  - Související proměnné lze seskupit do struktury
  - Struktury mohou mít i chování (metody)

```
struct Button {
  int pin;
  bool down;
  unsigned long whenPressed;
};

struct Button buttons[3];

void doSomethingWithButton(Button &button) {
  if (button.isDown) ...
}
```

Proměnné nesoucí stav tlačítka

Místo několika polí (int pins[], bool downs[], ..) pak bude jen jedno pole tlačítek

Funkce pracující s tlačítky pak budou mít parametr s referencí na tlačítko (alternativně lze některou funkcionalitu zabalit do metod)

Lepší, čitelnější kód

- Alternativa k **struct** je **class** (třída)
- V C++ je preferováno použití **class** nad použitím **struct**
  - Rozdíly jsou malé, 2 nejdůležitější rozdíly jsou
    - Defaultní viditelnost členských proměnných u **struct** je public, u **class** private
    - Zděděné proměnné mají jinou viditelnost (public inheritance vs private inheritance) – spíše zajímavost, na cvičení to s největší pravděpodobností nevyužijete.

```
class Button {
private:
  bool down;

public:
  bool isDown() {
    return down;
  }
};
```