

Arduino

7-segmentový displej

NSWI170: Lab 04

Patrik Dokoupil

Credit: Martin Kruliš

Shrnutí 3. úlohy

- Problémy:
 - Chybějící konstanty / nepoužití konstant z funshield.h (LOW/HIGH)
 - Dekompozice
 - Copy&paste dvou bloků ovládajících každé tlačítko zvlášť
 - Spousta kódu v **loop()**
 - Funkce které dělají moc věcí najednou (př. **increment_counter** která zvýší counter a pak jej vykreslí na LED)
 - Velká spousta zdánlivě nesouvisejících stavových proměnných
 - Ideální bylo použít **struct/class**, nebo alespoň pole
 - Logika pro counter uvnitř tlačítka

Sériová linka

- Lze využít pro komunikaci s (host) PC
 - **Arduino IDE -> Tools -> Serial Monitor**
 - Inicializace v **begin()** pomocí **Serial.begin(9600)**
 - Rychlost 9600 baudů (b/s)
 - Další podporované hodnoty [na webu](#)
- Hodí se pro "ladění"
 - Umožňuje výpisy přes sériovou linku do okna se Serial Monitorem
 - **Serial.println()**, **Serial.print()**
- Lze vypisovat např. hodnoty proměnných, kudy kód běží apod.
- Lze použít i opačným směrem -> zadání/čtení vstupu pomocí **Serial.read()**
 - Teď to nejspíše nevyužijete
- ~~POZOR, že ReCodEx to nebere, je nutné výpisy smazat (deaktivovat) před odevzdáním~~

Pravidlo č. 4

• # 4 Omezené používání globálních proměnných

- Obecně jsou špatné a nechceme je vůbec, v Arduinou nemáme moc na výběr
- Používejte jen pro stav s životností delší než jedno volání `loop()`
- Ideálně k glob. prom. přistupujte jen z `setup()` a `loop()`, jinde přes parametr
- Globální konstanty jsou OK, detaily [na webu](#)

Takhle ne!

```
int global_state;  
int local_state;  
void f1() {  
    // Some state update  
    ++global_state;  
}  
void loop() {  
    f1();  
    local_state = ...;  
    if (cond(local_state)) {  
        // Do something  
    }  
}
```

Funkce modifikuje glob. proměnnou

`local_state` se mění v každé iteraci, nedává smysl aby byla globální

Z volání `f1()` není jasné, že `f1()` má side-effect v podobě změny `global_state`

OK (pro Arduino)

```
int global_state;  
void f1(int & state) {  
    // Some state update  
    ++state;  
}  
void loop() {  
    f1(global_state);  
    int local_state = ...;  
    if (cond(local_state)) {  
        // Do something  
    }  
}
```

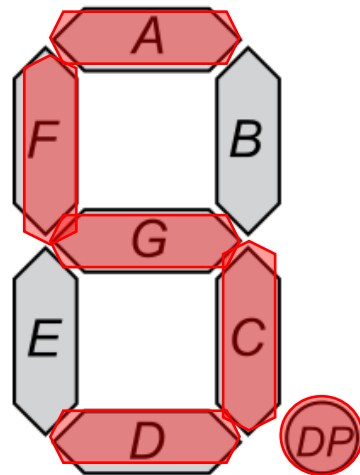
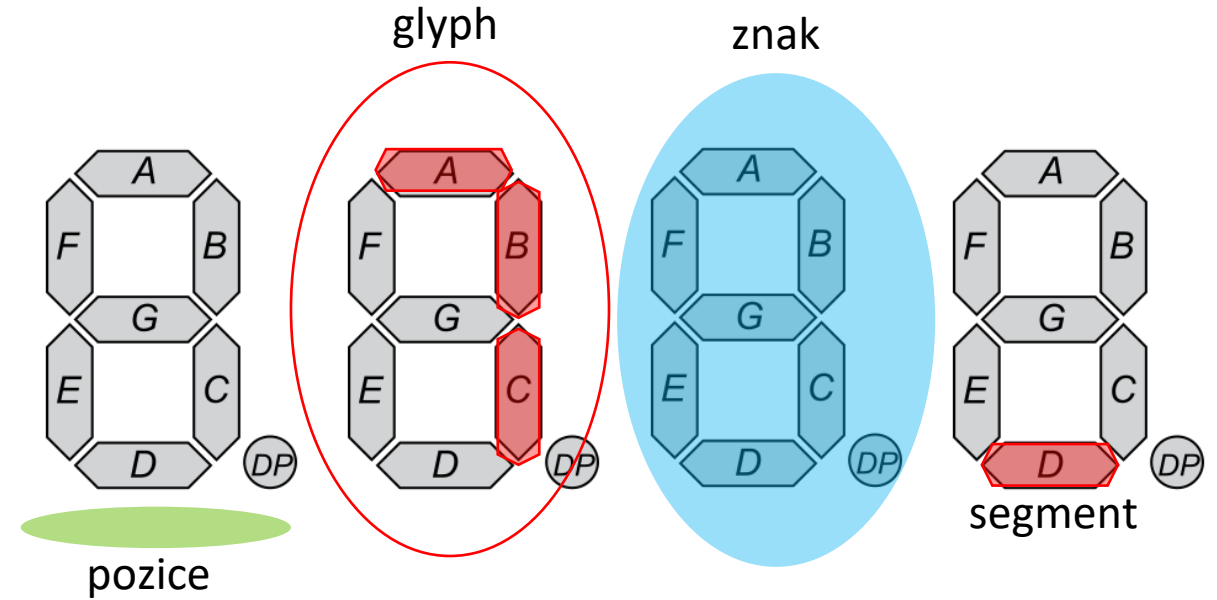
`f1` pracuje s parametrem

Někdy lze také
`int f1(int state) {
 return state + 1;
}`
Pozor na kopii

`local_state` je skutečně lokální

Segmentový displej

- 4-místný
- Ovládání jednotlivých segmentů
 - V rámci jednotlivých znaků
 - Segmenty A, B, C, D, E, F, G, DP jsou kódovány v bytu (opět inverzní logika)



Index bitu

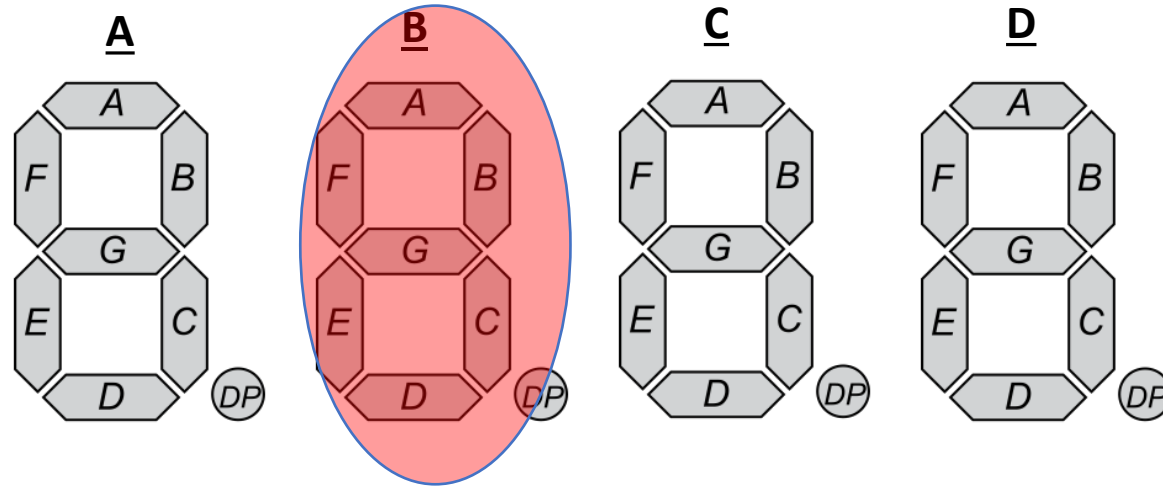
Odpovídající segment

Hodnota bitu

7	6	5	4	3	2	1	0
DP	G	F	E	D	C	B	A
0	0	0	1	0	0	1	0

Segmentový displej

- Výběr znaku resp. pozice se opět kóduje pomocí bitů
 - Pro **MSBFIRST** se mapují na dolní 4 bity (0 – 3)



Index bitu

Odpovídající ZNAK

Hodnota bitu

7	6	5	4	3	2	1	0
				<u>D</u>	<u>C</u>	<u>B</u>	<u>A</u>
				0	0	1	0

Pozor! Zde naopak 1 znamená aktivní

Segmentový displej - zápis

- Pomocí posuvného „shift“ registru
 - Posuvný registr 74HC959D – sériový vstup (posloupnost bitů), paralelní výstup „SIPO“
- Piny
 - `latch_pin` signalizující začátek/konec
 - `clock_pin, data_pin` sloužící k zápisu
 - Nutno všechny nastavit pomocí `pinMode(*_pin, OUTPUT)` ;
- `latch_pin`
 - `digitalWrite(latch_pin, LOW)` ; // zavírá „zámek“ ≈ začátek zápisu
 - `digitalWrite(latch_pin, HIGH)` ; // otevírá „zámek“ ≈ konec zápisu (výstupy kopírují stav posuvných registrů)
 - Analogie se zámkem je spíše pomůcka, ve skutečnosti je podstatný zápis LOW->HIGH
- Posílání dat posuvným registrům
 - `shiftOut(data_pin, clock_pin, MSBFIRST, value)` ;
 - **MSBFIRST** určuje posílání bitu od nejvyššího k nejnižšímu (alternativa je **LSBFIRST**)

Segmentový displej - zápis

- Inicializace
 - Inicializovat piny
 - Smazat displej (zhasnout všechny segmenty na všech pozicích)
- Pro zobrazení glyphu na pozici
 - `digitalWrite(latch_pin, LOW);`
 - Poslat masku glyphu (`shiftOut`)
 - Poslat byte s maskou pro vybraný znak/pozici (`shiftOut`)
 - Lze aktivovat i více pozic, pak bude na všech stejný glyph
 - `digitalWrite(latch_pin, HIGH);`
- Možná implementace:

```
void write_glyph_bitmask(byte glyph_mask, byte pos_mask) {  
    digitalWrite(latch_pin, LOW);  
    shiftOut(data_pin, clock_pin, MSBFIRST, glyph_mask);  
    shiftOut(data_pin, clock_pin, MSBFIRST, pos_mask);  
    digitalWrite(latch_pin, HIGH);  
}
```


Cvičení

- T1: Funkce `write_glyph(byte glyph, int pos)`
 - Na danou pozici zapíše daný glyph
 - Na rozdíl od `write_glyph_bitmask` nebere masku, ale pozici 0..3
 - Číslování pozic
 - Zprava – hodí se pro čísla
 - Zleva – hodí se pro text
- T2: Funkce `write_digit(int digit, int pos)`
 - Na danou pozici zapíše danou číslici (0..9)

Cvičení

- T3: Jednociferný counter
 - Vezměte si counter ze 3. cvičení, upravte rozsah na 0-9 (mod 10)
 - Hodnota counteru se zobrazí na poslední (pravé) pozici displeje
- T4: Jednociferný counter s nastavitelnou pozicí
 - Základ stejný jako u T3
 - Pomocí tlačítka č. 3 umožněte změnu pozice pro zobrazení counteru (D -> C -> B -> A -> D -> ...)

Domácí úloha

- Víceciferný counter (0 – 9999)
 - Zobrazení na displej, vždy je zobrazena právě 1 cifra (zatím žádný multiplexing)
 - První tlačítko provede inkrement aktuálně zobrazení cifry
 - Druhé tlačítko dekrement
 - Zobrazená cifra/pozice se mění pomocí třetího tlačítka
 - U čísel kratších než 4 cifry budou úvodní nuly
 - Detaily vč. videa jsou k dispozici v ReCodExu

Cvičení*

- Extension 1: Animace hada
 - Had postupně projde kolem obvodu celého displeje
 - V každém okamžiku svítí právě jeden segment
 - Konstantní rychlost pohybu, celé kolečko má trvat 1s
- Extension 2: Nachystejte si bit. masky pro písmena
 - Mix lower case / upper case
 - Zkuste pomocí těchto masek postupně vykreslit „hello world“
 - Vykreslení na jedné pozici, písmeno za písmenem