Začínáme s Arduinem

46 řešených příkladů a 123 námětů pro vlastní pokusy

Sada „Začínáme s Arduinem“ je určena pro všechny, kteří chtějí proniknout do základů aplikací a programování této platformy. Na příkladech ukazuje řešení typických situací a problémů souvisejících s použitím jednotlivých připojených vstupních a výstupních prvků. Vedle řešených příkladů dává i náměty na vlastní pokusy, příbuzné s uvedenými příklady. Příručka není kompletní učebnicí, která by beze zbytku pokrývala všechny dostupné příkazy a popis jejich činnosti, dokonce často se ani nesnaží o optimální způsob řešení problémů z hlediska rychlosti nebo délky programu. Spíše se vždy snaží vybrat způsob, který je snadno pochopitelný a vede k naučení něčeho dalšího. Příklady na sebe navazují a vyžadují zvládnutí znalostí získaných v předchozích příkladech.

Sada „Začínáme s Arduinem“ předpokládá základní znalosti elektroniky a digitální techniky na úrovni zapojení obvodu s několika součástkami podle schématu, funkce základních součástek (rezistoru, kondenzátoru, LED) a zapojení (vlastnosti sériového a paralelního řazení), pojmů jako napětí, proud, bit, byt, logická hodnota a úroveň, logických operací a podobně. Obsahuje vše potřebné k vyzkoušení příkladů a ještě něco navíc, jednak aby měl uživatel rezervu, jednak aby mohl svoje pokusy mírně rozšířit bez nutnosti hned dokupovat součástky.

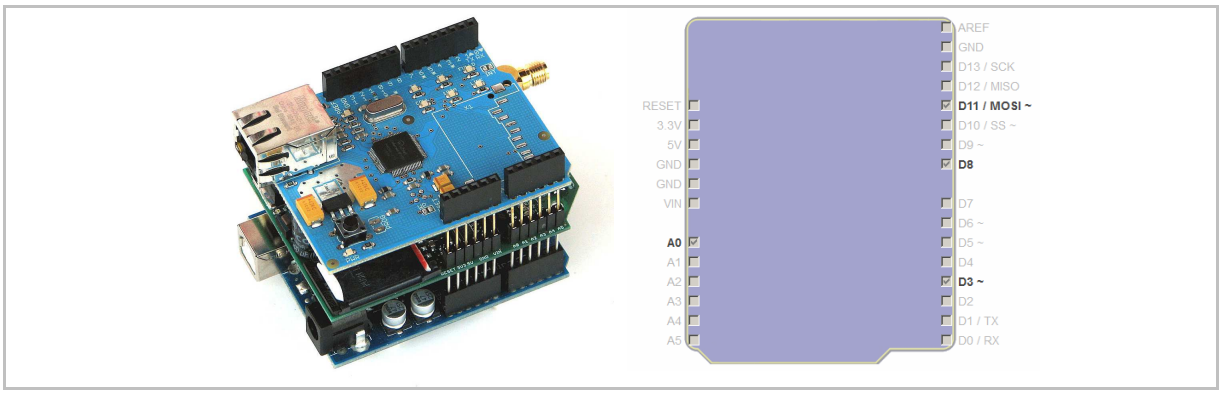
Co je vlastně Arduino

Projekt Arduino vznikl v roce 2004 v Itálii. Cílem bylo usnadnit návrh a výrobu prototypů zařízení s mikrokontroléry ATMega od firmy Atmel a to především v rámci výuky studentů. Na první pohled poněkud záhadný název dostala tato open source platforma od svých tvůrců Massima Banziho a Davida Cuartiellese po Arduinovi Ivrejském, významné historické postavě z města Ivrea, kde projekt vznikl.

Základní deska Arduina obsahuje mikrokontrolér a konektory, na které jsou vyvedeny jeho vstupy a výstupy, dále obvody napájení a rozhraní USB pro komunikaci s počítačem PC. Existuje více variant, my budeme pracovat s verzí Arduino Uno R3, založené na mikrokontroléru ATMega328P. Uno je nástupcem starší verze Arduino Duemilanove. Verze Mega 2560 nebo Mega ADK jsou výkonnější a také větší, poskytují více vstupů a výstupů, naopak třeba verze Mini, Fio a Nano jsou rozměrově podstatně menší a některé méně výkonné.

Kromě standardní řady typů Arduino existují i klony jiných výrobců s názvem končícím většinou na „duino“, například FreDuino, LABDuino nebo Boarduino.

Důležitou myšlenou je to, že základní desku Arduina většinou nepoužíváme samostatně a pokud ano, tak spíše pro účely seznámení s možnostmi a programováním. Na desku se přímo do konektorů zasouvají další specializované desky, kterým říkáme shieldy.



Jednoduchý shield může obsahovat třeba jen masivnější svorkovnice pro připojení vodičů k výstupům mikrokontroléru nebo kousek univerzální desky s plošným spojem, na kterou si můžeme osadit obvody podle vlastního návrhu. O něco složitější shield může obsahovat obvody, které posílí a oddělí vývody mikrokontroléru, aby například zkrat nebo napěťové špičky nezničily přímo Arduino, ale mnohem levnější a snadno vyměnitelné posilovače signálu. Specializované shieldy se postarají o komunikaci s počítačovou sítí a internetem, řízení stejnosměrných nebo krokových motorů, zobrazení dat na LCD displeji, měření různých fyzikálních veličin a podobně.

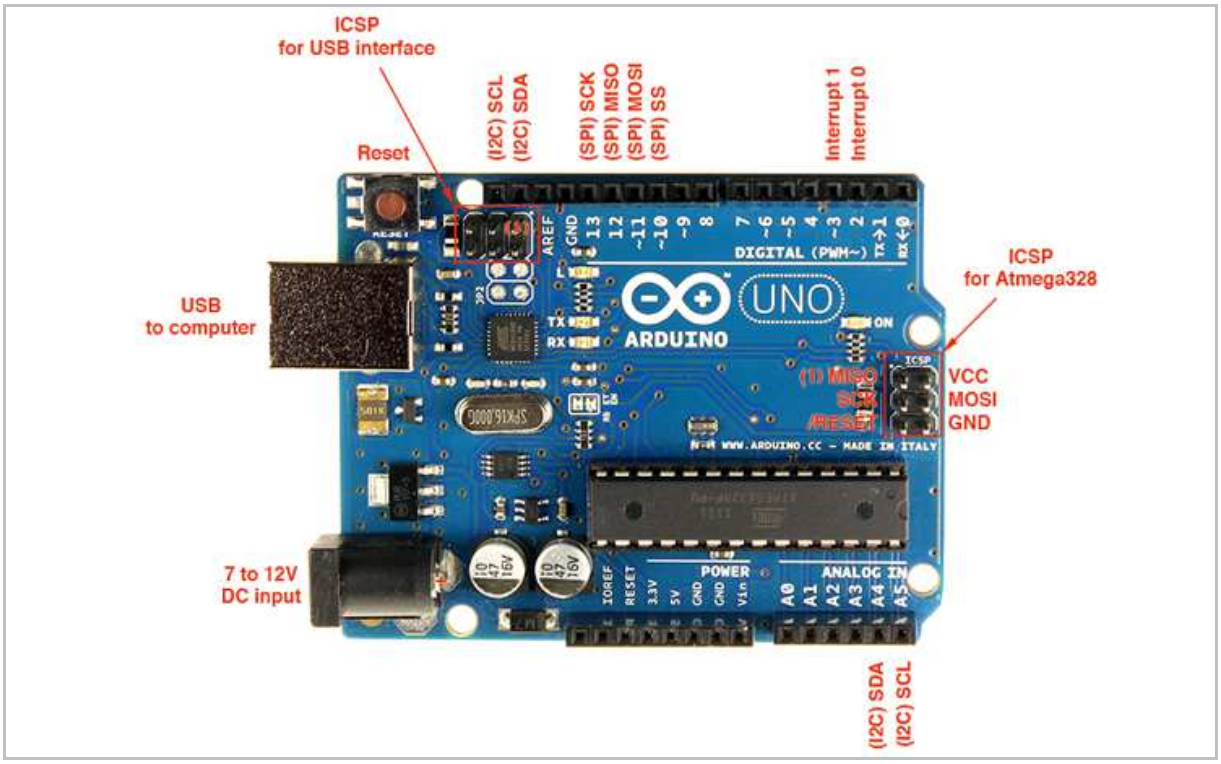
Většina chytrých shieldů je vyrobena tak, že umožňuje nasunutí dalšího shieldu nad sebe, takže lze tvořit celé „věže“ desek s různým účelem. Aby byl základní mikrokontrolér schopen takovou kombinaci využít, musí platit, že dvě různé desky nepotřebují využívat stejné vývody. K posouzení kompatibility slouží „mapy“ využívaných pinů.

Základní deska Arduina je vždy dole v sestavě. Některé shieldy jsou navrženy tak, že na ně už nejde připojit další; ty jsou většinou jednoúčelové pro výuku a musí se použít buď samostatně, nebo jako vrchní v sestavě. Žádný přehled existujících shieldů nemůže být úplný, jeden z rozsáhlejších najdete na internetových stránkách *http://www.shieldlist.org/* .

Arduino UNO R3

Při vývoji prakticky vždy využijeme konektor USB typu B, který slouží především k programování mikrokontroléru v Arduinu z počítače. Kromě toho se také využívá jako nejjednodušší způsob napájení Adruina přímo napětím +5V z PC a tak jej také budeme většinou využívat při popisovaných pokusech.

Samostatný napájecí vstup se souosým konektorem je určen pro napájení napětím 6 až 20 V, nicméně doporučený rozsah napětí je kvůli spolehlivé funkci stabilizátorů a přijatelným výkonovým ztrátám menší, 7 až 12 V.



K dispozici je celkem 14 digitálních vstupů/výstupu, z nichž 6 může pracovat v režimu PWM jako nastavitelná pulzně šířková regulace třeba pro řízení chodu stejnosměrných motorů nebo improvizované D/A převodníky. Mezi digitálními vývody najdeme rozhraní I^2C a SPI, dva vstupy mohou pracovat jako interrupty, další lze využít pro sériovou komunikaci. Na samostatný konektor je vyvedeno šest analogových vstupů pro A/D převodníky s rozsahem 0 až 5 V (0 až 1,1 V nebo externě určeným) a rozlišením 10 bit (1 024 hodnot). Analogové vstupy mohou být použity i jako digitální. Digitální výstupy mohou být zatíženy proudem maximálně 40 mA, výstup 3,3 V proudem 50 mA. Doporučená velikost proudové zátěže výstupů je přibližně poloviční. Napětí na digitálních vstupech nesmí překračovat meze napájení mikrokontroléru (5 V).

Mikrokontrolér ATMega328 pracující na frekvenci 16 MHz má 32 kB flash paměti na program, z toho 0,5 kB je trvale zabráno programem (loaderem) Arduina. Lze využít i 2 kB paměti SRAM a 1 kB EEPROM.

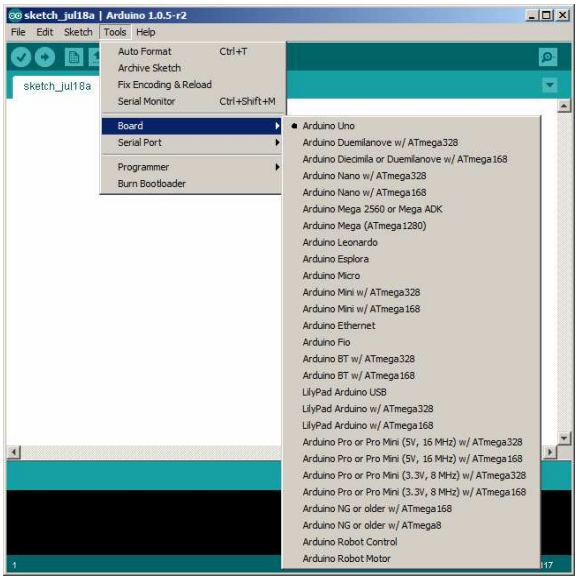
Prostředí pro psaní programů

K programování Arduina slouží jeho vlastní jazyk vycházející z jazyka Wiring. V mnoha ohledech je velmi podobný jazyku C/C++. Ten, kdo zná alespoň trochu C, nebude mít s programováním Arduina žádné potíže, a naopak, práce s Arduinem je výbornou průpravou pro budoucí náročnější programování v C/C++.

Základních příkazů jazyka není mnoho, nicméně můžeme využít širokou paletu dostupných knihoven podprogramů a funkcí, které přičleníme k našemu programu, aniž bychom museli detailně znát, jak přesně fungují. Stačí vědět, jak těmto podprogramům předat nutné údaje a jak nám poskytnou svůj výstup. Později si můžeme knihovny pro své učely i napsat a nabídnout je ostatním k využití.

Prostředí pro psaní programů a jejich přenos do Arduina si stáhneme z internetových stránek projektu *http://www.arduino.cc/* . Budeme pracovat v Arduinu IDE verze 1.0.5, což je osvědčené jednoduché prostředí, jehož výhodou je mimo jiné to, že stejně vypadá i funguje na počítačích s operačním systémem Windows (32 i 64 bit), Linux i Mac OS X. Následující popis platí pro Windows 7.

- Stáhneme si instalační program nebo si připravíme ten, který je na CD v sadě. Spustíme instalaci, ponecháme všechny části pro instalaci zatržené.
- Potvrdíme adresář pro instalaci, případně vybereme nebo založíme jiný. Po dotazu necháme instalovat i USB ovladače.
- Po skončení uzavřeme instalační okno. Na ploše přibyla ikona Arduino.
- V položce „Zařízení a tiskárny“ se objevilo nové zařízení – Arduino Uno. Poznamenáme si číslo sériového portu, který mu byl přidělen (COM xx). Uzavřeme systémová okna.



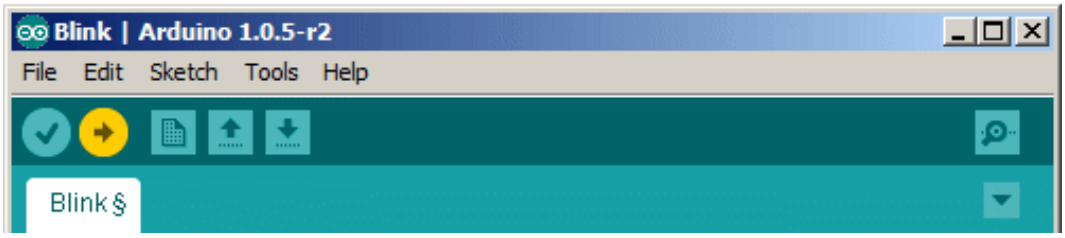
- Připojíme modul Arduina k počítači a počkáme, než se začnou automaticky instalovat ovladače. Po chvíli instalační program skončí s tím, že zařízení bylo úspěšně nainstalováno.
- Spustíme program. V menu Tools – Serial Port ověříme případně nastavíme aktuální COM port. V menu Tools – Board ověříme případně nastavíme typ Arduina – Arduino Uno. Tím je příprava prostředí hotová.

První program – blikání LED

Máme spuštěný program Arduino 1.0.5-r2 a Arduino připojené USB kabelem k počítaci, na desce svítí zelená LED jako signalizace napájení. Zatím nemusíme připojovat nic jiného. Využijeme toho, že přímo na desce je žlutá LED označená L a připojená k výstupu 13, pokusíme se ji rozblikat. V programu zvolíme jeden z připravených příkladů File – Examples – Basics – Blink a hotový ukázkový program se nám načte.

```
/*
BLINK
TURNS ON AN LED ON FOR ONE SECOND, THEN OFF FOR ONE SECOND, REPEATEDLY.
THIS EXAMPLE CODE IS IN THE PUBLIC DOMAIN.
*/
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:

int led = 13;
// the setup routine runs once when you press reset:

void setup() {
// initialize the digital pin as an output.
pinMode(led, OUTPUT);
}
// the loop routine runs over and over again forever:

void loop() {
digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
delay(1000); // wait for a second
digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}
```

Tak jak program je, aniž bychom mu zatím rozuměli, se jej pokusíme nahrát do Arduina. K tomu slouží tlačítko se šipkou vpravo na horní liště programu (na obrázku je žluté). Po jeho stisku dole v okně postupně proběhne modrý ukazel postupu programování a žlutá LED na Adruinu by se měla pomalu rozblikat s periodou 2 s. Podíváme se podrobně na vzorový program, pokusíme se mu porozumět, naučit se použité příkazy, a přepsat si program do srozumitelnější podoby.



Struktura programu

Základní struktura programu se skládá nejméně ze dvou částí, z nichž první se týká nastavení a po spuštění programu se provede jen jednou (void setup), druhá se bude provádět cyklicky stále dokola (void loop). Bloky příkazů, které jsou v těchto částech, se ohraničují složenými závorkami {…}. Jak kulaté závorky určené pro předávané parametry tak složené závorky ohraničující příkazy jsou povinné a uvádí se i tehdy, není-li mezi nimi nic. Každá otevřená závorka musí být uzavřena. Špatné závorkování je jednou nejčastějších chyb při psaní programu. V editoru existuje jednoduchá pomůcka, která pomáhá závorkování zviditelnit. Postavíme-li kurzor vedle otevřené (uzavřené) závodky, příslušná závorka tvořící s ní pár se zobrazí v malém rámečku. Jednotlivé příkazy se oddělují středníkem.

Základní struktura vypadá tedy takto:

```
void setup ()
  { … příkazy; …}

void loop ()
  { … příkazy; …}
```

Komentáře

Je-li někde na řádku dvojice lomítek za sebou //, znamená to, že text od tohoto místa až do konce řádku je (řádkový) komentář. Nic, co je za lomítky uvedené, se nebere jako program, a tedy se ani nepřenáší do Arduina a nezabírá místo v paměti. V zápisu se text komentáře zobrazuje šedou barvou. Tento zápis komentářů se používá nejčastěji k vysvětlení, co předchozí příkaz dělá.

Chceme-li mít komentář na více řádků, můžeme i na mnoho řádků za sebou uvést //, je to však hodně neúsporné. Lepší způsob je ten, že na začátek textu, dáme znaky /*, a na konec znaky */. Vše mezi tím se stane (blokovým, víceřádkovým) komentářem.

Blokový komentář používáme jednak pro rozsáhlejší popisy třeba v úvodu programu, jednak při programování, chceme-li nějaký úsek programu dočasně vyřadit z činnosti. Stane-li se řádkový komentář (uvedený //) součástí blokového komentáře, nevadí to.

Komentáře je nutné používat s mírou. Žádné komentáře nebo velmi málo komentářů dělá program nesrozumitelným a špatně pochopitelným, přemíra komentářů jej dělá nečitelným.

Příklady:

```
// toto je příklad řádkového komentáře
/*
toto je příklad víceřádkového
tj. blokového komentáře
*/
```

Proměnné, deklarace proměnných

Místo v paměti pro uložení číselné hodnoty (nebo textu) pojmenováváme, abychom jej mohli v programu opakovaně používat a měnit jeho obsah. Jména proměných volíme pokud možno krátká a výstižná, aby vyjadřovala a připomínala účel použití.

Pro začátek si vystačíme s jediným typem proměné „int“ (integer), která slouží k uložení celočíselné hodnoty v rozsahu -32 768 až +32 767 (v paměti mikrokontroléru zabírá dva byty).

Každou proměnnou musíme deklarovat, tj. před blokem, v němž bude používána, uvést, jakého bude typu a jak se bude nazývat, případně do ní můžeme rovnou přiřadit nějakou konkrétní hodnotu. Číselná proměnná, do níž nebyla přiřazena hodnota, má po deklaraci hodnotu 0. Pokud má být proměnná platná pro celý program, musíme ji deklarovat před celým programem, tedy i před funkcí s inicializací (void setup). Proměnná deklarovaná uvnitř funkce nebo bloku bude platit (bude viditelná) jen v rámci dané funkce (bloku) a bude-li program mimo tuto část, proměnná nebude existovat.

Příklady:

```
int doba;          // deklaruje proměnnou doba typu integer, hodnotu jí nepřiřazuje (=0)
int doba = 1000;  // deklaruje proměnnou doba typu integer, přiřazuje jí hodnotu 1 000
```

Přiřazení do proměnné

Pro vložení hodnoty do proměnné použijeme přiřazovací příkaz. Na levé straně je jméno proměnné, následuje znak = a na pravé straně je konstanta (zadané číslo), proměnná nebo libovolný matematický výraz (ty poznáme později).

Příklady:

```
pocet = 3;              // do proměnné pocet uloží konstantu, konkrétně číslo 3
pocet = celkem;         // do proměnné počet uloží obsah proměnné celkem
pocet = celkem * 2 -1   // do proměnné počet uloží výsledek operace (obsah proměnné
                        // celkem vynásobený dvěma a zmenšený o jedna)
```

Nastavení digitálních vstupů/výstupů

Digitální piny Arduina mohou být použity jako vstupy nebo výstupy. Arduino po spuštění automaticky nastaví všechny digitální piny jako vstupy, což je bezpečnější, ale nepracuje s nimi jako se vstupy. Chceme-li některé z nich použít jako výstupy, musíme je předem nastavit příkazem pinMode. Příkaz má dva parametry, první je číslo pinu, kterého se má týkat, druhý je konstanta „OUTPUT“ (tj. výstup). Chceme-li v programu přepnout pin do režimu vstupu, použijeme konstantu „INPUT“ (tj. vstup).

Příklady:

```
pinMode(12,OUTPUT); // nastaví digitální vývod 12 jako výstup
pinMode(12,INPUT);     // nastaví digitální vývod 12 jako vstup
```

Ovládání digitálních výstupů

Příkazem digitalWrite, který má dva parametry, číslo pinu a stav, nastavíme libovolný z digitálních výstupů. Stavem HIGH se rozumí logický stav H (napětí +5 V), stavem LOW se rozumí logický stav L (napětí 0 V).

Příklady:

```
digitalWrite(12,HIGH);  // nastaví předem deklarovaný výstup na pinu 12 do H
digitalWrite(11,LOW);   // nastaví předem deklarovaný výstup na pinu 11 do L
```

Příkaz digitalWrite můžeme použít nejen na výstup, ale také na předem deklarovaný vstup. V takovém případě příkaz digitalWrite(pin, HIGH) způsobí připojení vnitřního pull-up rezistoru k danému pinu, digitalwrite(pin, LOW) odpojí vnitřní pull-up rezistor daného pinu. Stejný efekt zapnutí pullup rezistorů má příkaz pinMode(pin, INPUT_PULLUP).

Co to jsou ty pull-up rezistory? Standardně je vstup ve stavu vysoké impedance, tj. i nepatrný proud stačí na to, aby se dostal do stavu H nebo L nebo naopak. Takový stav je velmi náchylný na rušení, na snímání falešných hodnot, není-li vývod připojen. Pull-up rezistor má hodnotu 20 až 50 kΩ a vnitřně připojí vstup na +5 V, takže zajistí, že bez připojení je na něm vždy přečtena hodnota H. Jsou-li rušivé vlivy extrémně silné, někdy ani tyto vnitřní pull-up rezistory nestačí na spolehlivé „ukotvení“ logické hodnoty vstupu, potom musíme připojit vnější pull-up rezistory s menším odporem, například 1 kΩ. Ve standardní situaci ale interní pull-up rezistory vyhovují.

Příklad:

```
pinMode(12,INPUT);          // nastaví digitální vývod 12 jako vstup
digitalWrite(12,HIGH);      // pro vývod 12 jako vstup zapne pull-up rezistor
```

Čekání

Nejjednodušší způsob, jak lze zajistit, aby program určitou dobu čekal, je použít funkci delay. Funkce má jeden číselný parametr a trvá tak dlouho, kolik odpovídá zadané hodnotě v tisícinách sekundy.

Příklad:

```
delay(100);         // čeká 100 ms = 0,1 s
delay(3600000);  // čeká 3 600 000 ms = 1 hod  … (teotericky s přesností 0,001 s)
```

Po dobu provádění delay mikrokontrolér nemůže dělat nic jiného, žádná jiná část programu nemůže běžet, nemůže kontrolovat stisk tlačítka, … Tento způsob čekání je možný a jednoduchý, ale poněkud nešťastný. Časem poznáme další způsoby, které během čekání (časování) dovolí mikrokontroléru věnovat se jiné činnosti.

První program jinak

Teď už máme probrané všechny příkazy, které se v našem prvním programu převzatém z příkladů od výrobce vyskytly. Je na čase zkusit přepsat stejný program do srozumitelnější podoby. Protože má program sloužit výhradně k rozblikání LED na Arduinu a ta má pevně danou adresu (pin 13), nemusíme ani používat proměnnou pro adresování LED.

```
// BLIKANI – PROGRAM ROZBLIKA LED NA ARDUINU (PIN 13) S PERIODOU 2 s

void setup(){
  pinMode(13,OUTPUT);     // inicializuje pin 13 jako vystup
}

void loop(){
  digitalWrite(13,HIGH);  // rozsvitit LED
  delay(1000);            // pockat 1s
  digitalWrite(13,LOW);   // zhasnout LED
  delay(1000);            // pockat 1s
}                         // konec programu
```

Pokusíme se upravit program tak, aby blikání bylo rychlé – pět bliknutí za jednu sekundu. Nejprve si spočítáme potřebné časy. Pět bliknutí během 1 s = 1 000 ms, na jedno připadá 200 ms. Doba zapnutí a vypnutí LED bude stejná (poloviční), po 100 ms.

Nový program bude vypadat takto:

```
// BLIKANI2 – PROGRAM ROZBLIKA LED NA ARDUINU (PIN 13) 5X ZA SEKUNDU

void setup(){
  pinMode(13,OUTPUT);     // inicializuje pin 13 jako vystup
}

void loop(){
  digitalWrite(13,HIGH);  // rozsvitit LED
  delay(100);             // pockat 0,1s
  digitalWrite(13,LOW);   // zhasnout LED
  delay(100);             // pockat 0,1s
}                         // konec programu
```

Nabízí se otázka, jak rychle by LED blikala, pokud bychom čekání úplně odstranili. Dost rychle, přibližně frekvencí 117 kHz, 117000x za sekundu. Arduino sice nepracuje s vysokou hodinovou frekvencí, jeho 16 MHz je v porovnání třeba s počítačem PC nepatrná hodnota, ale většina jednoduchých úkonů mu na rozdíl od PC trvá jen jeden jediný takt hodin, takže výsledným výkonem překvapí.

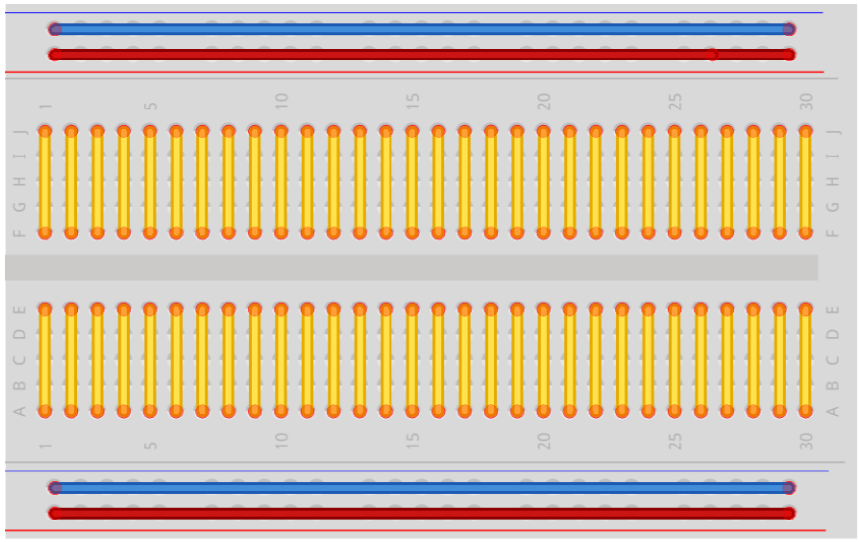
Náměty:

- Upravte program tak, aby LED blikala 50x za sekundu, a to asymetricky. Jednou bude doba svícení mnohem menší než doba zhasnutí (asi 10x), podruhé doba zhasnutí mnohem menší než doba svícení (asi 10x). Vnímá lidské oko toto blikání? Jak se liší vnímání svitu LED v obou případech?
- Upravte program tak, aby LED velmi krátce blikla jednou přibližně za 3 s. Vyzkoušejte, jaké nejkratší bliknutí zřetelně vidíte. Potom blikání obraťte, v dlouhém svitu bude krátké zhasnutí. Jak se liší délka nejkratšího rozvícení a nejkratšího zhasnutí LED, které jsou dobře vidět?
- Od blikání s frekvencí 10 Hz (delay(50)) zkoušejte postupně blikání, které má stejnou dobu svitu jako zhasnutí, zrychlovat. Najděte mez frekvence, kdy už blikání nevnímáte.

Práce s kontaktním polem

Pro další pokusy už budeme používat kontaktní pole. Pole má nahoře a dole dvě dlouhé spojené linie, které se využívají pro napájení, mohou (ale nemusí) být i označeny červenou barvou (+) a modrou barvou (- , zem, GND). Ostatní otvory jsou propojené v pěticích svisle, kolmo na napájecí linie.

Propojení ukazuje následující obrázek, který, stejně jako všechny další, vznikl v programu Fritzing.



K propojování slouží vodiče opatřené koncovkami. Snažíme se zasunovat kontakty kolmo, jinak hrozí jejich ulomení. Pokud do pole umísťujeme integrovaný obvod, dáváme jej „přes střední drážku“ a pokud možno výřezem vlevo, aby číslování vývodů pouzdra odpovídalo směru číslování sloupců na kontaktním poli.

Není to nutné, ale orientaci na zapojené desce usnadní, pokud si zvykneme na určité barevné konvence. Záporný pól napájení (zem) rozvádíme modrými vodiči, kladný pól napájení červenými nebo oranžovými. Signály vedeme ostatními barvami a snažíme se, aby ty s podobným významem měly stejnou barvu, například vstupy od tlačítek žlutými vodiči, výstupy k LED bílými, … Vodičů není neomezené množství, takže tyto konvence nejde v praxi vždy dodržet, ale podobné zvyky dost zrychlí orientaci v zapojení a omezí chyby.

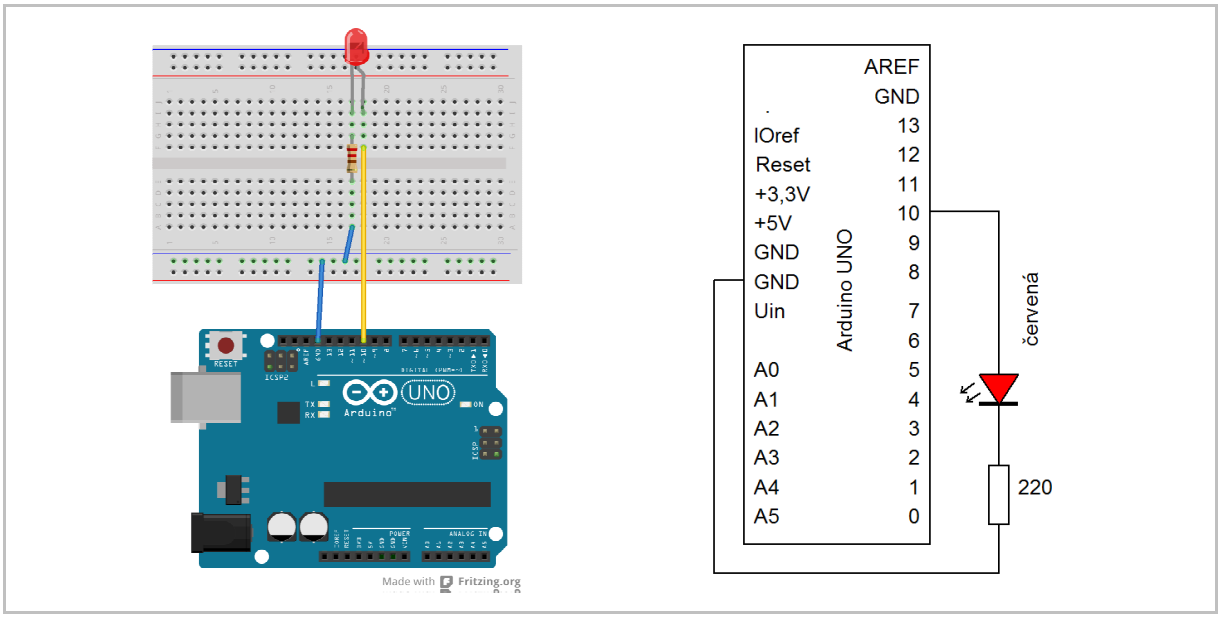
Kontaktní pole zvládne všechny zde uvedené úlohy. Při vlastních pokusech musíme dbát na dvě věci. V první řadě nesnažit se do pole silou „vmáčknout“ vývody součástek větší než otvory v poli. Otvory v plastu jde ještě s trochou síly zvětšit, ale pružinové kontakty pod nimi se vytáhnou a jsou potom nespolehlivé. Druhá věc je proudové přetížení kontaktů. Do 200 mA není žádný problém, nad 500 mA už mohou kontakty hřát tak, že se plast zkroutí. Výkonové obvody (samozřejmě s jiným napájením než z Arduina) je třeba řešit mimo kontaktní pole.

Ovládáme LED

Máme k dispozici všechny znalosti nutné k tomu, abychom si mohli troufnout na úlohy typu simulace silničního semaforu, výstražného světla na železničním přejezdu, „běžícího světla“ nebo „Kitt“ efekt. Nejprve si však zopakujeme rozblikání jediné LED, ovšem už na kontakním poli.

Jedna blikající LED

Obvod zapojíme podle obrázku, červená LED je připojena na digitální vývod 10, proud je omezen rezistorem 220 Ω.

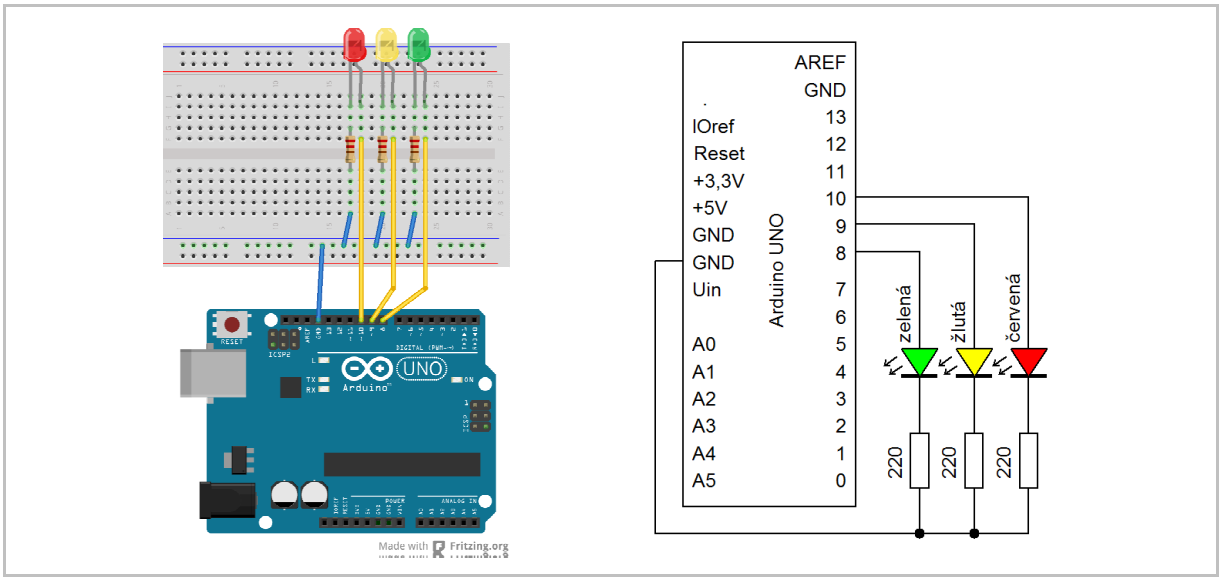


Program bude velmi podobný jako předchozí, necháme ale červenou LED svítit 5 s a pak ji na 1 s zhasneme. Aby šlo jednoduše změnit pin, na který je LED připojená, bude jeho označení uloženo na začátku programu do proměnné „cervena“.

```
// BLIKANI3 PROGRAM ROZBLIKA LED NA ARDUINU (PIN 10) 5 s / 1 s

int cervena = 10;

void setup(){
  pinMode(cervena,OUTPUT);     // inicializuje pin pro cervenou LED
}

void loop(){
  digitalWrite(cervena,HIGH); // rozsvitit cervenou LED
  delay(5000);                // pockat 5s
  digitalWrite(cervena,LOW);  // zhasnout cervenou LED
  delay(1000);                // pockat 1s
}                             // konec programu
```

Semafor

Zapojení z předchozího příkladu ponecháme, jen k němu přidáme další dvě LED, žlutou na výstup 9 a zelenou na výstup 8. Semafor bude svítit proti reálnému trochu zrychleně v následujícím časování: 5 s červená, 1 s žlutá, 5 s zelená, 1 s žlutá … a cyklus se opakuje.




```
// SEMAFOR CERVENA / ZLUTA / ZELENA / ZLUTA S CASY 5 / 1 / 5 / 1 s

int cervena = 10;
int zluta = 9;
int zelena = 8;

void setup(){
  pinMode(cervena,OUTPUT);     // inicializuje pin pro cervenou LED
  pinMode(zluta,OUTPUT);       // inicializuje pin pro zlutou LED
  pinMode(zelena,OUTPUT);      // inicializuje pin pro zelenou LED
}

void loop(){
  digitalWrite(zluta,LOW);     // zhasnout zlutou LED
  digitalWrite(cervena,HIGH); // rozsvitit cervenou LED
  delay(5000);                 // pockat 5s
  digitalWrite(cervena,LOW);   // zhasnout cervenou LED
  digitalWrite(zluta,HIGH);    // rozsvitit zlutou LED
  delay(1000);                 // pockat 1s
  digitalWrite(zluta,LOW);     // zhasnout zlutou LED
  digitalWrite(zelena,HIGH);   // rozsvitit zelenou LED
  delay(5000);                 // pockat 5s
  digitalWrite(zelena,LOW);    // zhasnout zelenou LED
  digitalWrite(zluta,HIGH);    // rozsvitit zlutou LED
  delay(1000);                 // pockat 1s
}                              // konec programu
```

Náměty:

- Upravte předchozí program tak, aby při přechodu z červené na zelenou svítila žlutá současně s červenou, v opačném přechodu zůstane program tak, jak je.
- Upravte původní program tak, aby zelená před koncem svícení 3x krátce blikla a pak rovnou přešla do červené bez fáze žluté. Při přechodu z červené na zelenou bude svítit žlutá stejně jako v původním příkladu.

V předchozím programu jsme pro zápis pinů, na nějž jsou jednotlivé LED připojené, použili proměnnou a hodnotu v ní. Když bude potřeba změnit pin, stačí změnit hodnotu na jednom jediném místě v programu. To je správný způsob. Když se ale podíváme na program, nikde se hodnota v uvedených proměnných nemění, takže proměnné byly vlastně použity zbytečně. Stačilo uvést piny jako konstanty pomocí symbolického pojmenování.

Symbolická jména, konstanty

Příkaz #define má dva parametry uváděné bezprostředně za něj a oddělené jen mezerou, první je zvolené jméno, druhý hodnota, kterou se symbolické jméno v programu nahradí. Příkaz se neodděluje (neukončuje) středníkem! Předdefinované konstanty jsou třeba již používané HIGH, LOW, INPUT nebo OUTPUT.

Příklad:

```
#define cervena 10          // místo jména červená použije program hodnotu 10
#define pi 3,141593         // místo jména pi se vezme hodnota Π na 6 desetinných míst
```

Čtení (digitálního) stavu ze vstupu

Opakem příkazu digitalWrite je digitalRead. Má jeden parametr, a to označení pinu, z něhož se má přečíst stav. Vrací hodnotu HIGH nebo LOW. Je důležité, aby čtený pin měl jednoznačnou logickou hodnotu. Typickým případem, kdy není jeho stav jednoznačný, je volný nezapojený pin ve třetím stavu (inicializovaný jako vstup bez pullup). Může-li k takové situaci dojít, je nutné alespoň zapnout pull-up rezistor a “ukotvit“ stav pinu k hodnotě HIGH.

Příklad:

```
blikani = digitalRead(spinac);  // do proměnné blikani uloží HIGH nebo LOW podle stavu
                                // na pinu spinac v okamžiku provádění příkazu
```

Větvení programu pomocí IF … ELSE

Činnost lze jednoduše vysvětlit na zápisu: **if (podmínka) {příkaz1} else {příkaz2}**

Jestliže uvedená podmínka platí, provede se příkaz1, jestliže neplatí, provede se příkaz2. Právě jedna z těchto dvou možností vždy nastane. Příkaz nemusí být jednoduchý, může jít i o mnoho příkazů, celou část programu. Místo podmínky může být i proměnná (typu boolean), která nabývá hodnot true (pravda) nebo false (nepravda), podmínka může být i relativně složitá.

Příklad:

```
if (digitalRead(spinac) == LOW) digitalWrite(led,HIGH) else digitalWrite(led,LOW)
// jestliže na pinu označeném spinač je hodnota LOW (logická hodnota L, nízká
// úroveň napětí), pak na pin označený led pošli úroveň HIGH (rozsviť LED), pokud ne,
// na pin označený led pošli úroveň LOW (zhasni LED))
```

Možné zápisy porovnání (relační operátory):

= =	je rovno
!=	není rovno
<	je menší než
>	je větší než
<=	je menší nebo rovno
>=	je větší nebo rovno

Složené podmínky

Jedna podmínka nemusí stačit, v tom případě můžeme psát složitější složené podmínky. Z hlediska programu je podmínka výrazem, který nabývá logické hodnoty pravda nebo nepravda, true nebo false. Mají-li být splněny dvě podmínky současně, použijeme mezi nimi operátor konjunkce (AND), stačí-li, když bude splněna jen jedna z nich, operátor disjunkce (OR), pokud se má nějaká logická hodnota otočit, operátor negace (NOT).

K zápisu použijeme následující symboly:

&&	konjunkce (musí platit oba výrazy)
||	disjunkce (platí alespoň jeden z výrazů nebo oba)
!	negace (z true udělá false a naopak)

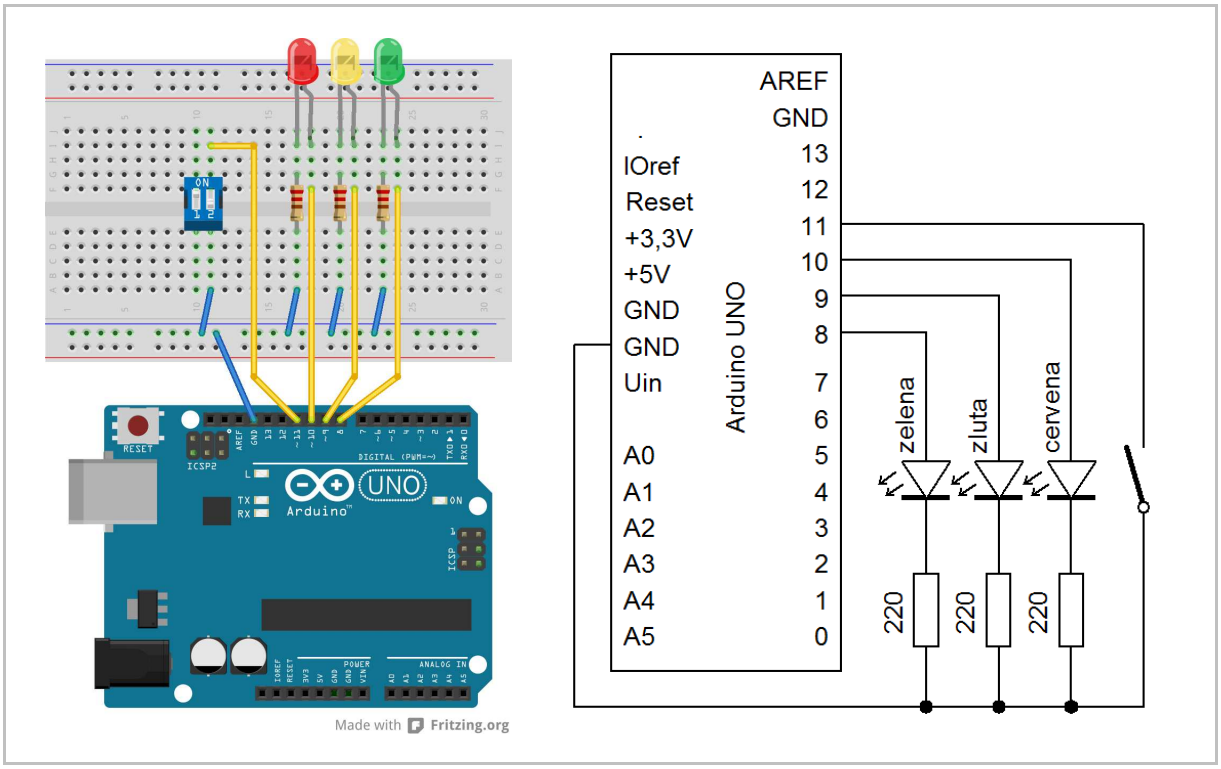
Příklad:

(x > 5) && (x < 10)	aby byl výraz pravdivý, musí být x z intervalu od 5 do 10, meze 5 a 10 do intervalu nepatří
(x >= 5) || (x < 10)	výraz je pravdivý vždy, každé číslo je větší než 5 nebo menší než 10

Dále se pokusíme funkci semaforu zdokonalit, zapojení vybavíme vypínačem. Bude-li vypínač zapnutý, přejde semafor do režimu pravidelného blikání žluté. K tomu budeme potřebovat umět číst stav pinu (vypínače, tlačítka) a větvit program podle dané podmínky. Jako spínač použijeme jednu sekci čtyřnásobného DIL spínače.

Rozšířená verze semaforu

Zapojení doplníme o spínač podle obrázku nebo schématu. Spínač bude spojovat pin 11 k zemi. Bude-li sepnutý, bude na vstupu jasně definovaná úroveň, pokud bude rozpojený, nastal případ, kdy je vlastně pin 11 nezapojený. Aby byla i v tomto případě jeho úroveň jednoznačně daná, připojíme vnitřní pull-up rezistor pro pin 11. Je-li spínač rozpojený, bude semafor fungovat tak jako dosud, je-li zapnutý, bude blikat žluté světlo.




```
// SEMAFOR2 FUNKCE SEMAFORU ROZSIRENA O BLIKANI ZLUTE

#define cervena 10              // pin pro cervenou LED
#define zluta 9                 // pin pro zlutou LED
#define zelena 8                // pin pro zelenou LED
#define spinac 11               // pin pro spinac

void setup() {                  // inicializace
  pinMode(cervena,OUTPUT);      // vystup pro cervenou LED
  pinMode(zluta,OUTPUT);        // vystup pro zlutou LED
  pinMode(zelena,OUTPUT);       // vystup pro zelenou LED
  pinMode(spinac,INPUT_PULLUP);// vstup pro spinac (s pull-up)
}

void loop() {                           // smycka programu
  if (digitalRead(spinac) == HIGH){     // precti spinac, je-li vypnut:
    digitalWrite(zluta,LOW);            // zhasnout zlutou LED
    digitalWrite(cervena,HIGH);         // rozsvitit cervenou LED
    delay(5000);                        // pockat 5s
    digitalWrite(cervena,LOW);          // zhasnout cervenou LED
    digitalWrite(zluta,HIGH);           // rozsvitit zlutou LED
    delay(1000);                        // pockat 1s
    digitalWrite(zluta,LOW);            // zhasnout zlutou LED
    digitalWrite(zelena,HIGH);          // rozsvitit zelenou LED
    delay(5000);                        // pockat 5s
    digitalWrite(zelena,LOW);           // zhasnout zelenou LED
    digitalWrite(zluta,HIGH);           // rozsvitit zlutou LED
    delay(1000);                        // pockat 1s
  } else {                              // je-li spinac zapnut
```

```
    digitalWrite(zluta,LOW);          // zhasnout zlutou LED
    delay(500);                       // pockat 0,5s
    digitalWrite(zluta,HIGH);         // zhasnout zlutou LED
    delay(500);                       // pockat 0,5s
    }                                 // konec bloku pro else
}                                     // konec programu
```

Náměty:

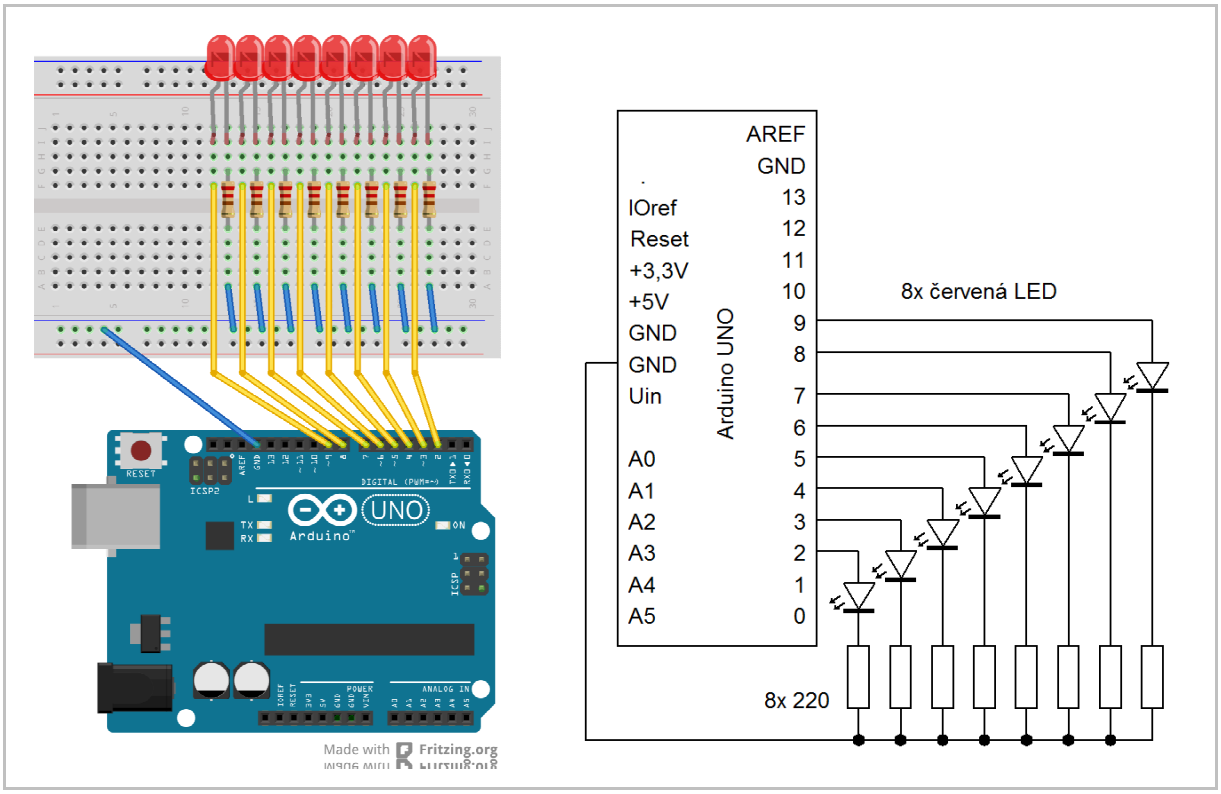
- Upravte program pro semafor tak, aby žluté přerušované světlo svítilo při vypnutém spínači a barvy se střídaly při zapnutém
- Navrhněte zapojení i program pro simulaci výstražného světla na železničním přejezdu. Je-li spínač sepnutý, bliká střídavě pravé a levé červené světlo, je-li vypnutý, bliká pomalu bílé (žluté) světlo.
- Ve stejném zapojení jako v bodě 2 upravte program tak, aby jedna LED blikala s periodou 2 s a druhá LED s periodou 1 s.

Běžící světlo (světelný had)

Jedním z velmi oblíbených efektů jsou různé varianty „běžícího světla“. Použijeme všech osm červených LED, vytvoříme z nich řadu a zapojíme na piny 2 až 9 (viz obrázek a schéma). Program nejprve napíšeme jen s využitím příkazů, které jsme zatím poznali. Je velmi jednoduchý, nicméně poměrně dlouhý. Následně se pokusíme program zkrátit pomocí cyklů a možná ještě efektivněji.

```
// HAD1 - EFEKT BEZICIHO SVETLA

#define cas 125

void setup() {              // inicializace
  pinMode(2,OUTPUT);        // piny 2 až 9 jako vystup
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
}

void loop() {               // smycka programu
  digitalWrite(2,HIGH);     // bliknuti 1. LED
  delay(cas);
  digitalWrite(2,LOW);
  digitalWrite(3,HIGH);     // bliknuti 2. LED
  delay(cas);
  digitalWrite(3,LOW);
  digitalWrite(4,HIGH);     // bliknuti 3. LED
  delay(cas);
  digitalWrite(4,LOW);
  digitalWrite(5,HIGH);     // bliknuti 4. LED
  delay(cas);
  digitalWrite(5,LOW);
  digitalWrite(6,HIGH);     // bliknuti 5. LED
```

```
  delay(cas);
  digitalWrite(6,LOW);
  digitalWrite(7,HIGH);     // bliknuti 6. LED
  delay(cas);
  digitalWrite(7,LOW);
  digitalWrite(8,HIGH);     // bliknuti 7. LED
  delay(cas);
  digitalWrite(8,LOW);
  digitalWrite(9,HIGH);     // bliknuti 8. LED
  delay(cas);
  digitalWrite(9,LOW);
}                           // konec programu
```



Cyklus for

Je-li předem známo, kolikrát se má určitá činnost opakovat, je to typický případ pro aplikaci for cyklu. For cyklus si založí svou řídící proměnnou, která je platná jen v rámci tohoto cyklu. Potřebuje meze hodnot pro tuto proměnnou a předpis, co se má s proměnnou po každém průchodu cyklem udělat. Zní to složitě, na příkladu pro pětinásobné opakování příkazu to bude názornější:

```
for (int i = 1; i <= 5; i = i + 1)
  {příkaz}
```

V tomto příkladu je for cyklem definovaná proměnná typu integer s názvem „i“ a na začátku se do ní přiřadí hodnota 1. Cyklus se opakuje dokud platí, že hodnota „i“ je menší nebo rovná zadané druhé mezi cyklu, v tomto případě 5. Po každém průchodu se obsah proměnné „i“ zvýší o jednu.

Porovnání jsme už používali, v tomto případě se však uplatnil i jednoduchý výpočet, nejčastěji přičítání a odečítání hodnoty 1. Arduino stejně jako jazyk C používá v těchto případech vedle standardního zápisu (pro promennou „i“)

i = i + 1 nebo i = i – 1

velmi často krátký zápis téhož

i++ nebo i--

Předchozí zápis for cyklu tedy může být kratší for (int i = 1; i <= 5; i++) {příkaz} a celý program přepsaný pomocí for cyklu bude vypadat takto:

```
// HAD2 EFEKT BEZICIHO SVETLA

#define cas 125

void setup() {              // inicializace
  for (int i=2; i<=9; i++){
    pinMode(i,OUTPUT);      // piny 2 až 9 jako vystup
  }
}

void loop() {               // smycka programu
  for (int i=2; i<=9; i++){
    digitalWrite(i,HIGH);   // bliknuti LED i
    delay(cas);
    digitalWrite(i,LOW);
  }                         // konec bloku for
}                           // konec programu
```

Je zřejmé, že for cyklus zkrátí program tím víc, čím víc příkazů je v něm a čím vícekrát se opakují. Řídící proměnnou uvnitř cyklu můžeme používat, jinak by to ani většinou nemělo smysl, ale NIKDY bychom neměli uvnitř cyklu měnit její hodnotu. Program to sice dovolí, ale vede to k závažným a špatně odhalitelným chybám.

Náměty:

- Změňte program pro běžící světlo využívající for cykly tak, aby světlo běželo v opačném směru.
- Změňte program pro běžící světlo využívající for cykly tak, aby nesvítila v jednom okamžiku jen jedna LED, ale aby se „posunoval“ svit dvou LED vedle sebe.
- Změňte program pro běžící světlo využívající for cykly tak, aby světlo neustále přebíhalo sem a zpět mezi konci řady. Tomuto efektu se někdy také říká „Kitt efekt“ nebo „Knight Rider efekt“ podle stejnojmenného televizního seriálu, v němž byl použit.
- Změňte program pro běžící světlo využívající for cykly tak, aby cyklus začínal svitem prostředních dvou LED a pak se světlo rozbíhalo do obou stran současně.

Efekty pro pokročilejší

Existuje ještě jeden způsob, jak lze v podobném případě jako je efekt běžícího světla program zkrátit a zjednodušit, zejména pokud se využívají jen skupiny pinů 0 až 7 nebo 8 až 13. Tento způsob ovšem vyžaduje dobrou znalost, na co je ten který vývod mikrokontroléru připojený, které piny využívá Arduino pro svoje vnitřní účely a není rozumné je nijak ovlivňovat. Nesprávné užití registrů může způsobit rozsáhlé a obtížně odhalitelné chyby, proto je větší využívání registrů technikou, k níž by měli sáhnout až pokročilejší uživatelé.

Registry portů

Arduino zná tři porty, které jsou označeny písmeny. Port D je nejdostupnější, tvoří jej digitální piny 0 až 7 odpovídající bitům 0 až 7 portu. Musíme však brát v úvahu, že bity 0 a 1 standardně slouží k sériové komunikaci s PC. Port B tvoří digitální piny 8 až 13 jako bity 0 až 5, zbývající dva bity (6 a 7) nejde v Arduinu použít. Port C tvoří analogové piny 0 až 5, zbývající bity 6 a 7 nejde u Arduina Uno použít.

Ke každému z uvedených portů se váží tři předdefinové registry. Registr DDRx (kde x je označení portu) slouží pro bitové určení, který z pinů bude vstupem (hodnota 0) a který výstupem (hodnota 1). Druhý registr je PORTx (kde x je označení portu), ten slouží k zápisu dat na danou osmici (nebo menší sestavu) pinů. Zápis se pochopitelně projeví jen na pinech, které byly předem definovány jako výstup. Registr lze i číst. Třetí z registrů je PINx, ten je pouze pro čtení a přečteme z něj stav pinů definovaných jako vstupy.

Použít operace s registry místo standardního digitalWrite / digitalRead obecně podstatně zhoršuje srozumitelnost programu, zhoršuje možnosti ladění a brání přenositelnosti programu na jiné typy Arduin (mikrokontrolérů). Má to smysl zejména v několika speciálních případech:

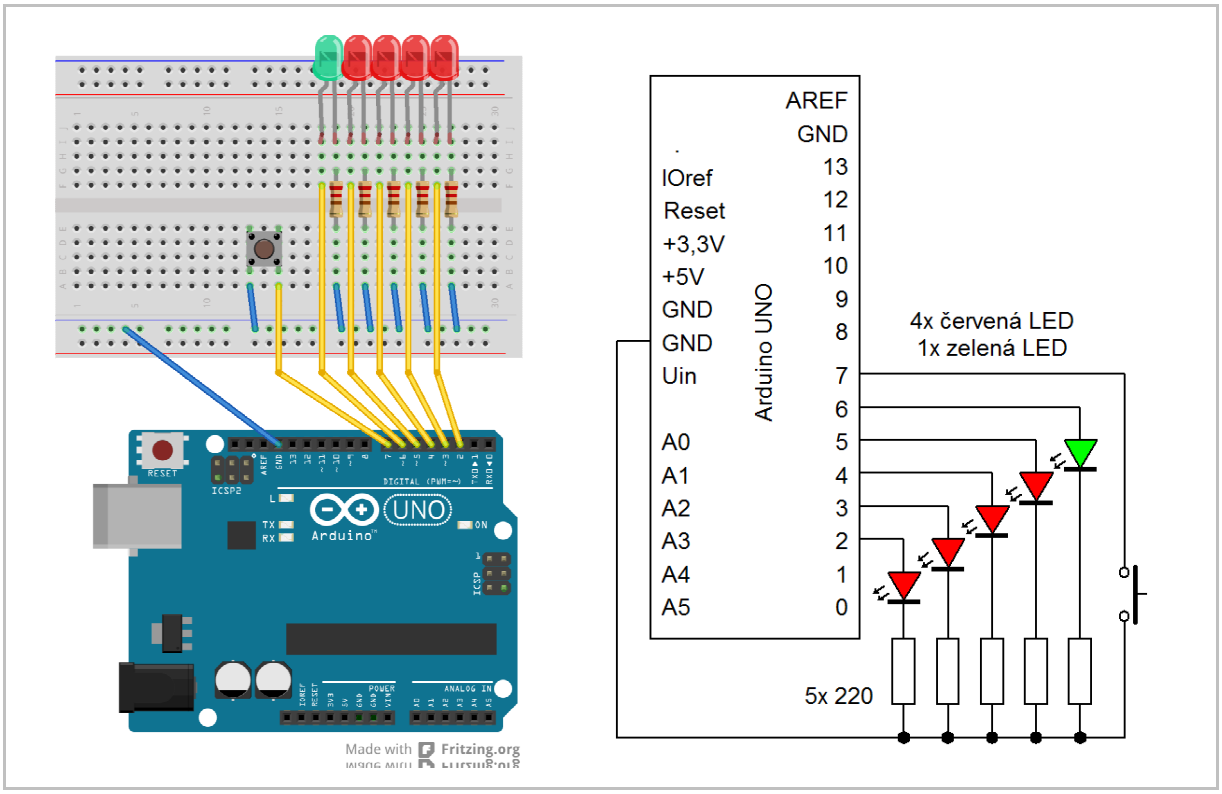
- když je nutné, aby se logické úrovně výstupů měnily naprosto synchronně (současně) a vadí postupná práce s nimi, i když rozdíly jsou v řádu 0,1 μs
- když je nutné, aby se více digitálních vstupů četlo zcela synchronně ve stejném okamžiku
- když je nezbytné zkrátit program doslova o pár bytů a je možné standardní funkce digitalWrite / digitalRead zcela vynechat, operace s registry jsou v přeloženém kódu mnohem kratší
- když je nezbytné přesně načasovat rychlý běh programu (týká se úloh běžících v reálném čase), operace s registry jsou velmi rychlé

Ukážeme si použití na následujícím příkladu. Chceme, aby 6 LED připojených na piny 8 až 13 (což odpovídá portu B) střídavě blikalo (schema není uvedno, je obdobou příkladu s běžícím světlem). Musíme tyto piny přepnout na výstup a pak na ně střídavě posílat hodnotu HIGH a LOW. Standardní přístup (s vynecháním čekání) bude mít 18 příkazů (6 pro nastavení pinů na výstupy, 6 pro vyslání první série hodnot a 6 pro vyslání druhé série hodnot). Obsluha pomocí registrů bude vypadat třeba takto:

```
// STRIDAVE BLIKANI 6 LED NA PORTU B POMOCI REGISTRU

void setup() {DDRB = B111111;} // inicializuje piny 8-13 na vystup

void loop() {                  // smycka programu
  PORTB = B101010;             // vysle na piny 8-13 prvni stav
  delay(250);                  // pocka 250ms
  PORTB = B010101;             // vysle na piny 8-13 druhy stav
  delay(250);                  // pocka 250ms
}                              // konec programu
```

Časování programu bez delay()

Vyjdeme ze zapojení úlohy s běžícím světlem. Vyjmeme LED na pinech 6 až 9, ponecháme LED na pinech 2 až 5, k pinu 6 doplníme zelenou LED. Zapojíme tlačítko tlačítko na pin 8. Zapojení je na obrázku a schématu.



Aritmetické operace

Již jsme víceméně intuitivně používali aritmetické operace sčítání (+) a odčítání (-). Další možné operace, které můžeme používat, jsou pochopitelně násobení (*) a dělení (/). Z často používaných operací si ještě jmenujme je zbytek po dělení neboli modulo (%).

Při výpočtech musíme dávat pozor na použité typy proměnných, protože výsledek je vždy toho typu, který má větší z typů použitých operandů. Například když provedeme sčítání dvou proměnných A a B, obě budou typu integer, bude i výsledek typu integer. Bude-li A=10 000 a B=20 000, bude výsledek roven 30 000 jak bychom čekali, protože se tento výsledek ještě vejde do typu integer. Pokud ale bude A=20 000 a B=20 000, pak výsledek nebude 40 000 (rozsah int je -32 768 až 32 767), ale proměnná přeteče a my dostaneme výsledek -25 535. Bude-li ale jedna pro proměnných třeba typu unsigned long, bude výsledek v pořádku a typu unsigned long.

Analogicky musíme dávat pozor na přetečení při násobení. Při standardním dělení dvou čísel typu integer dostaneme celou část výsledku (desetinná místa se do integeru „nevejdou“), nikoli zaokrouhlení.

Operace v pohyblivé řádové čárce nejsou úplně přesné. Přesvědčit se o tom můžeme třeba tím, že výsledek výpočtu 1/13*13 porovnáme s číslem 1. V praxi využíváme spíš test, zda dvě srovnávané hodnoty leží blízko sebe v malém intervalu.

Přehled číselných typů:

byte	Zabírá 1 byt, celočíselné hodnoty 0 až 255
int (short)	Zabírá 2 byty, celočíselné hodnoty -32 768 až +32 767
unsigned int (word)	Zabírá 2 byty, celočíselné hodnoty 0 až +65 535
long	Zabírá 4 byty, celočíselné hodnoty -2 147 483 648 až +2 147 483 647
unsigned long	Zabírá 4 byty, celočíselné hodnoty 0 až +4 294 967 295
float (double)	Zabírá 4 byty, čísla s plovoucí desetinnou čárkou od -3,402 823 5E+38 do 3,402 823 5E+38

Čteme čas – millis()

Millis je funkce bez parametru (i tak se kulaté závody musí uvést), která vrací hodnotu času od spuštění programu v milisekundách, hodnota je typu unsigned long (rozsah 0 až 4 294 967 295). K přetečení této proměnné při počítání času dojde přibližně po 50 dnech.

Příklad:

```
unsigned long cas;    // definice proměnné cas typu unsigned long
cas = millis();       // do proměnné cas se přiřadí aktuální hodnota v ms
```

A znovu blikání LED

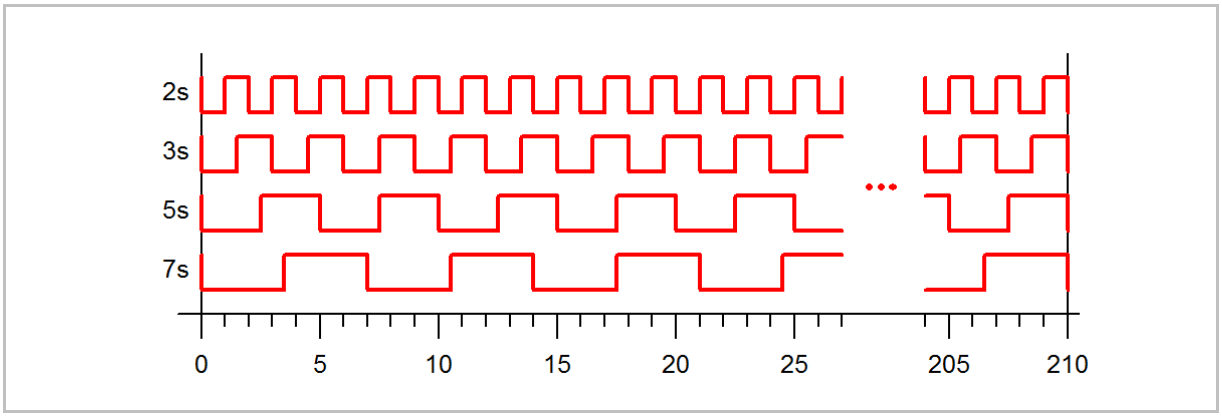
Nejprve si zkusíme jednoduché blikání 1x za sekundu s jednou LED připojenou na pin 2. Využijeme toho, že Arduino počítá čas uplynulý od spuštění programu a tato hodnota je nám dostupní prostřednictvím funkce millis. Pro odměření intervalu od daného okamžiku si přečteme a uložíme čas, potom cyklicky čteme čas, odečteme uloženou hodnotu, a jakmile je výsledek větší nebo roven požadovanému intervalu, akci ukončíme. Pokud jde o generování kmitů, můžeme využít kontroly zbytku po dělení.

```
// BLIKANI4 - LED NA PIN 2 S FREKVENCI 1Hz BEZ DELAY

void setup() {                  // inicializace
  pinMode(2,OUTPUT);            // pin 2 na vystup
}

void loop() {                   // smycka programu
  if (millis() % 1000 < 500){ // kontrola času, zbytek po deleni
    digitalWrite(2,HIGH);       // prvni polovina periody H
  } else {
    digitalWrite(2,LOW);}       // druha polovina periody L
// ... dalsi prikazy ...
}                               // konec programu
```

Důležité na tomto způsobu časování je to, že program může ve smyčce vykonávat další příkazy. Když budou trvat déle než 1 ms, o něco se sníží přesnost délky kmitů, nic víc. Pulzy časované přes delay jsou všechny přesně stejné, ale jejich frekvence neodpovídá tak úplně tomu, co chceme, pulzy časované přes millis se drobně liší jeden od druhého, ale ve větším časovém úseku má jejich frekvence i fáze krystalovou přesnost.

Využijeme celé zapojení. Naší další úlohou bude zařídit, aby první LED blikala s periodou 2 s, druhá 3 s, třetí 5 s a čtvrtá 7 s. Blikání by mělo mít střídu (poměr svitu a vypnutí) přibližně 1:1. Současně s tím požadujeme, aby kdykoli podržíme tlačítko na pinu 7 stisknuté, rozsvítila se zelená LED.

Tuto úlohu nelze řešit s použitím delay, protože v průběhu čekání nelze číst stav tlačítka. I přes to se zkusme zamyslet, jak by se s pomocí delay daly obsloužit alespoň blikající LED. Nejprve si nakreslíme graficky průběh stavů všech čtyř blikajících LED.



Sekvence se opakuje po 210 s, „rastr“, v němž dochází ke změnám stavu, má 0,5 s. Museli bychom pro každý časový úsek, v němž jsou všechny 4 signály stabilní, použít delay správné délky, a mezi nimi změnit stav jednoho nebo více výstupů. Kolik by jich bylo? Moc. Jen v prvních 10 sekundách průběhu najdeme takových stavů 15. Přidání dalšího výstupu prodlouží program mnohonásobně. Zkusme to tedy raději pomocí millis, jeden výstup navíc by znamenal nárůst programu o 1 řádek (nebo nejvýš o 4, budeme-li důsledně psát jeden příkaz na řádek).

Jeden průběh smyčky programu trvá méně než 190 μs a v každém průběhu se čte tlačítko a upravuje stav zelené LED podle něj, reakce LED na stisk není delší než 1/5000 s.

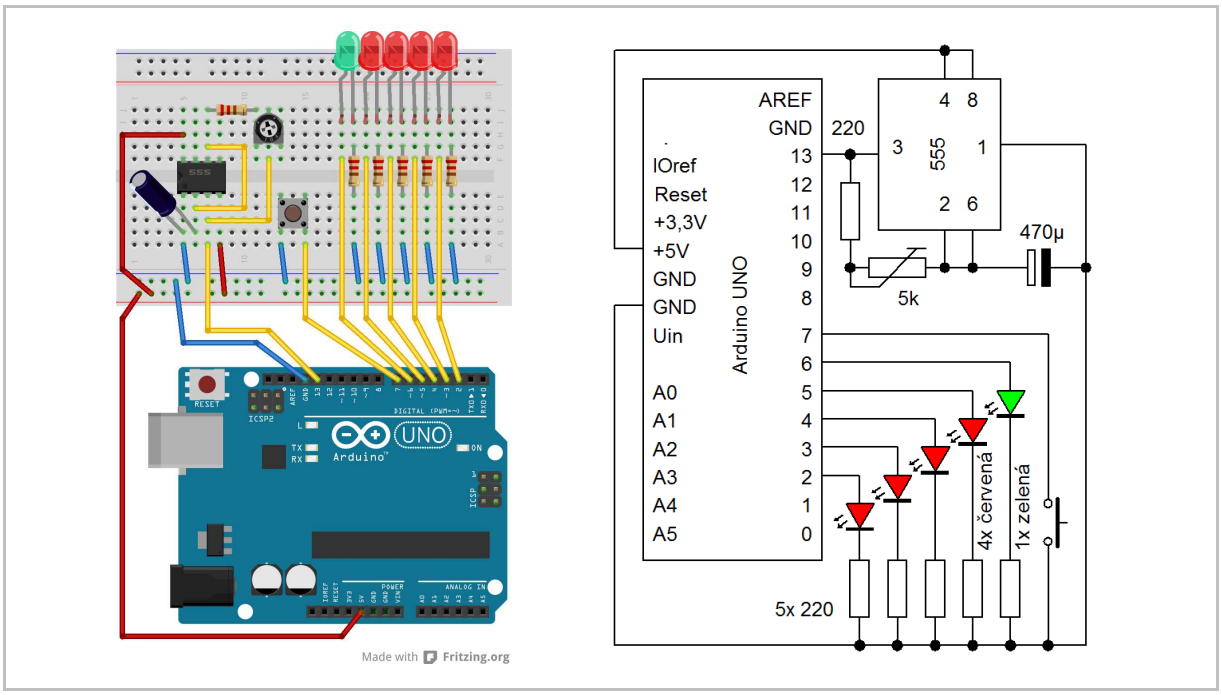
```
// BLIKANI5 - LED NA PIN 2-5, PERIODA 2/3/5/7S + TLACITKO

unsigned long cas;          // promenna pro ulozeni casu

void setup() {              // inicializace
  pinMode(2,OUTPUT);        // pin 2 vystup - cervena
  pinMode(3,OUTPUT);        // pin 3 vystup - cervena
  pinMode(4,OUTPUT);        // pin 4 vystup - cervena
  pinMode(5,OUTPUT);        // pin 5 vystup - cervena
  pinMode(6,OUTPUT);        // pin 6 vystup - zelena
  pinMode(7,INPUT_PULLUP);  // pin 7 vstup tlacitka
}

void loop() {               // smycka programu
  cas = millis();           // nacteni casu do pameti
  if (cas % 2000 < 1000){digitalWrite(2,HIGH);}else{digitalWrite(2,LOW);}
  if (cas % 3000 < 1500){digitalWrite(3,HIGH);}else{digitalWrite(3,LOW);}
  if (cas % 5000 < 2500){digitalWrite(4,HIGH);}else{digitalWrite(4,LOW);}
  if (cas % 7000 < 3500){digitalWrite(5,HIGH);}else{digitalWrite(5,LOW);}
  // kmity postupne pro 2000, 3000, 5000 a 7000 ms
  if (digitalRead(7)){digitalWrite(6,LOW);}else{digitalWrite(6,HIGH);}
                            // obsluha zelene LED
}                           // konec programu
```

Náměty:

- Upravte program tak, aby čtyři červené LED počítaly čas v binárním kódu (první bliká s periodou 1 s, druhá 2 s, třetí 4 s, čtvrtá 8 s, výchozí stav jsou všechny LED zhasnuté). Tlačítko má stejnou funkci jako dosud, při stisku se rozsvítí zelená LED.
- Přidejte do zapojení dvě červené LED na piny 8 a 9 a předchozí program rozšiřte tak, aby binárně počítalo všech 6 červených LED.
- Upravte program tak, aby se vytvořil efekt běžícího světla. (Buď bude potřeba použít dvě porovnání času pro jednu LED nebo použít složené podmínky – zbytek je v intervalu od … do respektive je větší než … a současně je menší než …).

Pomocný generátor s obvodem 555

Integrovaný obvod 555 je asi nejznámější a nejčastěji používaný, když je potřeba vytvořit nějaké kmity s frekvencí od 0,1 po 100 000 Hz. Budeme jej využívat v případě, kdy budeme chtít nějak zpracovat signál, který zcela zjevně vznikl mimo Arduino. Předchozí zapojení si doplníme o generátor kmitů s nastavitelnou periodou v rozsahu asi 0,25 až 7 s obvodem 555.



Jakmile připojíme napájení, rozbliká se žlutá LED označená L na Arduinu, protože výstup našeho generátoru je přiveden na pin 13, který je současně spojen s touto LED. Pokusíme se rozblikat zelenou LED našeho zapojení tak, aby kopírovala výstup z generátoru. Vyzkoušíme rozsah regulace kmitočtu kmitů trimrem.

```
// BLIKANI6 - LED NA PIN 6, PERIODA DLE 555

void setup() {                  // inicializace
  pinMode(6,OUTPUT);            // pin 6 vystup - zelena
  pinMode(13,INPUT);            // pin 13 vstup od 555
}

void loop() {                   // smycka programu
  if (digitalRead(13)==HIGH){   // cteni vstupu
    digitalWrite(6,HIGH);       // kopirovani na vystup
    }else{
    digitalWrite(6,LOW);
  }
}                               // konec programu
```

Náměty:

- Upravte program tak, aby zelená LED v době, kdy je výstup z časovače 555 v úrovni H, rychle blikala. Pomalé kmity tedy určuje časovač, rychlé program.
- Vyměňte kondenzátor v zapojení za jiný s kapacitou 10 μ, blikání se změní na velmi rychlé. Upravte původní program tak, aby výsledek byl podobný jako v předchozím námětu, ale zapojení pracovalo opačně (rychlé kmity určuje časovač 555, pomalé jsou určeny programem).
- Nastavte v původním zapojení trimem pomalé kmity. Zelená LED kopíruje výstup z časovače. Signál z časovače náběžnou hranou (přechodem z L do H) spouští efekt běžícího světla, světlo jednou rychle přeběhne od LED na pinu 5 po LED na pinu 2 a pak se opět čeká na další náběžnou hranu. Co se stane, když zkrátíme periodu časovače tak, že bude kratší než doba přeběhu světla?
- Podobně jako v předchozím námětu zelená LED v původním zapojení kopíruje výstup z časovače a náběžná hrana signálu z časovače spouští běžící světlo od LED na pinu 5 po LED na pinu 2. Nyní ale navíc sestupná hrana signálu z časovače (změna z H na L) spustí běžící světlo v opačném směru, tedy od LED na pinu 2 po LED na pinu 5.
- Zelená LED nebude využita. Upravte program tak, aby Arduino počítalo pulzy přicházející z generátoru a zobrazovalo výsledek v binární soustavě na červených LED na pinech 2 až 5
- Vytvořte efekt běžícího světla přes všechny LED, od první zelené (pin 6) po poslední červenou (pin 2). Rychlost běžícího světla bude určena generátorem s 555, každá náběžná hrana signálu z generátoru posune světlo o jeden krok.
- Totéž jako v předchozím námětu, ale běžící světlo se posune při každé náběžné i každé sestupné hraně signálu z generátoru
- Zpomalte kmity z generátoru na minimum. Bude využita jen zelená LED a tlačítko. Sestavte vlastní program, zelená LED bude kopírovat stav z výstupu generátou, ale současně stisk tlačítka změní její stav (rozsvítí, je-li zhasnuto, zhasne, je-li rozsvíceno). Aby se stav mnohokrát a víceméně náhodně neměnil, je důležité testovat ne stav tlačítka, ale první změnu (sestupnou hranu). Snažte se pomocí tlačítka udržet zelenou LED zhasnutou.

Sériová komunikace

Zatím mělo naše Arduino jedinou možnost, jak projevit svou činnost, rozsvítit nebo zhasnout LED. Sériová komunikace přes USB kabel, kterým se přenáší program z PC do Arduina, ale nemusí sloužit jen k tomuto účelu. Může také přenášet informace z programu Arduina na zobrazení do PC nebo vysílat z PC povely, které ovlivní činnost programu. K tomu budeme potřebovat několik nových funkcí.

Inicializace Serial.begin()

V první řadě je nutné nastavit parametry sériové komunikace. Jsou dva, první vyjadřuje přenosovou rychlost v bitech za sekundu. Typické hodnoty jsou 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 a 115200 Bd. Standardně nastavená rychlost je 9600 Bd a není-li k tomu důvod, nebudeme ji měnit. Druhý parametr vyjadřuje počet přenášených bitů, paritu a počet stopbitů. Například SERIAL_8N1 znamená přenos 8 bitů nez parity (No) a s jedním stopbitem. Toto je také standardní nastavení a není-li k tomu důvod, zejména při komunikaci s PC, nebudeme režim měnit. V takovém případě se druhý parametr vůbec nemusí uvádět.

Příklad:

```
Serial.begin(9600);    // Nastaví přenosovou rychlost 9600 Bd
```

„Tisk“ přenosem do počítače Serial.print(), Serial.println()

Tyto funkce odešlou po USB kabelu do PC obsah parametru. Obsahem může být číslo (konstanta) včetně formátování, znak, text, byte, obsah proměnné a podobně. Serial.print odesílá pouze to, co uvedeme v parametru, Serial.println za konec přidá znak přechodu na nový řádek (odřádkuje). Pozor při zadávání čísel, desetinným oddělovačem je tečka, ne čárka, jako v češtině! Odesílaná hodnota může být jen jedna, čárka odděluje údaj formátování!

Příklady:

```
Serial.print(13);              // odešle do PC číslo 13 (zobrazí se číslice 1 a 3)
Serial.print(„A“)              // odešle do PC znak A
Serial.print(„Toto je text“)   // odešle do PC text (řetězec) Toto je text
Serial.print(7.149152)         // odešle do PC číslo 7.15 (zaokrouhlí na 2 des. místa)
Serial.print(7.149152,4)       // odešle do PC číslo 7.1492 (zaokrouhlí na 4 des. Místa)
Serial.print(7.149152,0)       // odešle do PC číslo 7 (zaokrouhlí na celé číslo)
Serial.print(86,BIN)           // odešle do PC číslo 1010110 (číslo 86 v binárním tvaru)
Serial.print(86,HEX)           // odešle do PC číslo 56 (číslo 86 v hexadecimálním tvaru)
```

Od počítače k Arduinu – Serial.available()

Tato funkce nemá parametr. Vrací počet bytů, které přišly z PC, jsou uloženy v bufferu a zatím se nepřečetly. Pokud vrátí hodnotu 0, není připraveno nic ke čtení. Maximální počet bytů, které se uloží do bufferu, je 64. Slouží jen ke kontrole, jestli jsou nějaké znaky připravené, tato funkce sama znaky nečte!

Příklad:

if (Serial.available() > 0) {znak = Serial.read();}

// jestliže jsou v bufferu nepřečtená data, přečti jeden byte a ulož ho do proměnné znak

Od počítače k Arduinu – Serial.read()

Tato funkce nemá parametr. Přečte z bufferu jeden byt a vrátí jej jako hodnotu. Před použitím je vhodné funkcí Serial.available ověřit, že v bufferu je něco uloženo, jinak vrací hodnotu 255.

Příklad:

if (Serial.available() > 0) {znak = Serial.read();}

// jestliže jsou v bufferu nepřečtená data, přečti jeden byte a ulož ho do proměnné znak

Monitor sériové linky

Abychom byli schopni vidět, co nám Arduino po sériové lince posílá, případně mu také něco poslali, potřebujeme monitor sériového portu. Je možné použít jakýkoli z podobných programů, přímo v prostředí pro psaní programu pro Arduino ale máme jeden monitor a budeme jej využívat. Program si zavoláme buď přes menu Tools – Serial Monitor nebo kombinací kláves Ctrl+Shift+M.

V horní části okna je řádka, kam můžeme napsat to, co chceme odeslat. Největší část plochy okna zabírá prostor, do něhož se vypisují přijatá data. V dolním řádku máme vlevo možnost zapnout nebo vypnout automatické rolování výpisu, vpravo nastavení rychlosti komunikace, ta musí být stejná jako jsme zadali do programu Arduina.

Výpis textu

V prvním příkladu nepotřebujeme žádné připojené součástky, necháme Arduino jen cyklicky po sekundě odesílat slovo „Ahoj“, vypisovat se bude pokaždé na nový řádek.

```
// PRENOSY1 – DO PC – AHOJ

void setup() {                // inicializace
  Serial.begin(9600);         // seriovy port 9600Bd
}

void loop() {                 // smycka programu
  Serial.println(„Ahoj“);     // odeslat text Ahoj
  delay(1000);                // pockat 1s
}                             // konec programu
```

Všimneme si, jak v pravidelných intervalech blikne žlutá LED Tx na Arduinu, která signalizuje odeslání dat po sériové lince. Analogicky, LED Rx signalizuje příjem dat z PC. Můžeme napsat nějaký text do monitoru a odeslat jej, LED blikne, údaje sa uloží do bufferu, ale nejsou Arduinem přečteny.

Náměty:

- Programem v Arduinu vypište (pošlete na monitor) následující obrazce složené z jediného znaku „x“:

XXXXXXX XXXXXXX XXXXXXX XXXXXXX XXXXXXX XXXXXXX	X XX XXX XXXX XXXXX XXXXXX	X XX XXX XXXX XXXXX XXXXXX

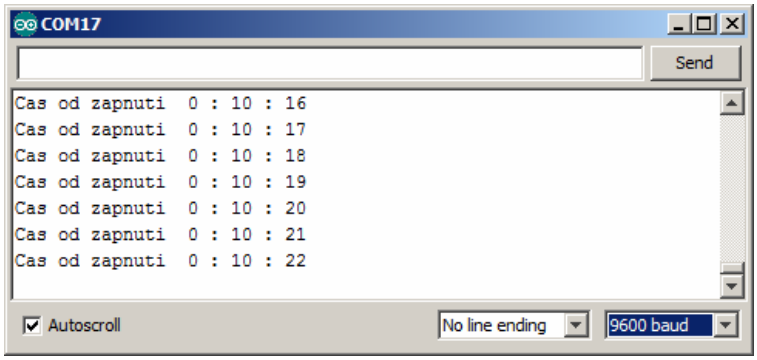
- Stejné obrazce vytvořte pomocí příkazu Serial.print (Serial.println) s výpisem jediného znaku a cyklů for.

Výpis času

Jako druhý krok zkusíme vypisovat po sekundách čas od spuštění programu. Využijeme funkci millis, ale ta poskytuje údaj v ms, my jej předevedeme na standardní formát hodiny:minuty:sekundy. V rámci jednotlivých čísel ale formátování (jednotky pod jednotky, desítky pod desítky) už dál dělat nemusíme.

```
// PRENOSY2 – DO PC – VYPIS CASU

unsigned long cas;                        // promenna pro ulozeni casu

void setup() {                            // inicializace
  Serial.begin(9600);                     // seriovy port 9600Bd
}

void loop() {                             // smycka programu
  cas = millis();                         // nacteni casu
  if (cas % 1000 == 0){                   // kazdou sekundu
    Serial.print(„Cas od zapnuti „);      // uvodni text
    Serial.print(cas / 3600000);          // cele hodiny
    cas = cas % 3600000;                  // odecist hodiny
    Serial.print(„ : „);                  // oddelit
    Serial.print(cas / 60000);            // cele minuty
    Serial.print(„ : „);                  // oddelit
    Serial.println(cas % 60000 / 1000);   // sekundy
    delay(1);                             // pockat pres konec ms
  }                                       // konec vypisu
}                                         // konec programu
```

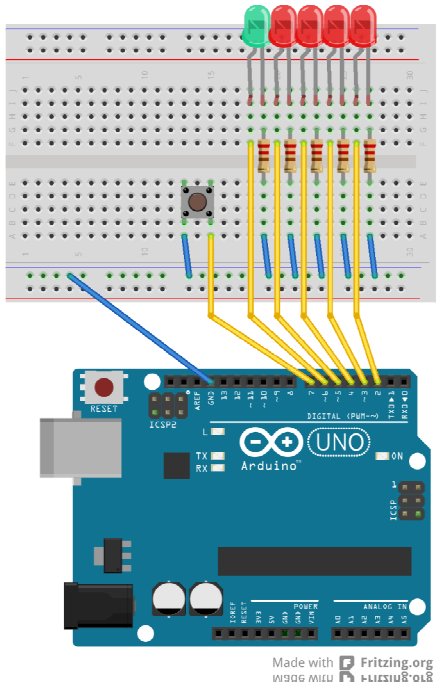
Na tomto programu si všimněme čekání delay 1 ms na konci. Je tam nutné, protože i když v „nulté“ milisekundě každé sekundy děláme výpočty a šestkrát voláme funkci výpisu, stále se to vše ještě stine v průběhu té jedné milisekundy nejméně dvakrát, takže řádky by byly zdvojené. Čekání zajistí, že když se program vrátí ke kontrole času, bude už „nultá“ milisekunda pryč. Podobnou rychlost umožní to, že příkaz bleskově zapíše údaje do výstupního bufferu a běží dál, nemusí čekat, až se je podaří fyzicky odeslat rychlostí 9 600 Bd.



Výpisy přes sériovou linku do monitoru v PC využíváme velmi výhodně zejména při ladění programu, když potřebujeme zjistit, jestli danou větví program prošel, jakou hodnotu vstupu opravdu čte a podobně.

Naprosto stejným způsobem lze pracovat se zařízením připojeným na piny 1 (Rx) a 2 (Tx), které má sériový vstup/výstup. V tomto případě ovšem nemůže být Arduino připojeno v průběhu chodu programu k PC.

Výpis po stisku tlačítka



Zkusíme reagovat výpisem na vnější událost, na stisk tlačítka. Pokud bude tlačítko stisknuto, vypíše se oznámení (jen jednou), pokud bude puštěno, vypíše se oznámení (také jen jednou). Zapojení využijeme stejné jako jsme připravili ke zkouškám časování programu bez delay.

Aby se výpis udělal jen jednou při změně, poznamenáme si minulý stav tlačítka do proměnné a budeme porovnávat, zda došlo ke změně. I když je tlačítko kvalitní, mechanické kontakty někdy zakmitávají a Arduino je tak rychlé, že může tyto zákmity nasnímat a považovat za záměrné vypnutí a zapnutí.

Použijeme nejjednodušší způsob „odkmitání“ tlačítka, po změně počkáme určitou dobu, během níž zákmity odezní. Zpomalíme tím sice program, ale v tomto ukázkovém příkladě to vadit nebude. LED diody zůstanou zatím nevyužity.

```
// PRENOSY3 - DO PC - HLASENI OD TLACITKA NA PINU 7

boolean tlac = false;                     // promenna pro stav tlacitka

void setup() {                            // inicializace
  Serial.begin(9600);                     // seriovy port 9600Bd
  pinMode(7,INPUT_PULLUP);                // tlačitko jako vstup
}

void loop() {                             // smycka programu
  if ((digitalRead(7) == HIGH) && tlac) { // neni stisknuto a bylo
    Serial.println(„Tlacitko vypnuto“);   // vypsat text
    tlac = false;                         // nastavit tlacitko vypnuto
    delay(100);                           // pockat na ustaleni
  }
  if (digitalRead(7) == LOW && !tlac) {   // je stisknuto a nebylo
    Serial.println(„Tlacitko zapnuto“);   // vypsat text
    tlac = true;                          // nastavit tlacitko zapnuto
    delay(100);                           // pockat na ustaleni
  }
}                                         // konec programu
```

Ovládáme Arduino z PC – switch … case

Následující příklad ukazuje, jak lze ovládat činnost programu v Arduinu z PC. Využijeme stále stejné zapojení, tantokrát ale budeme pracovat s červenými LED. Chceme zajistit, aby se vysláním znaku „A“ rozsvítila první LED, následně znakem „a“ zhasla. Analogicky další LED budou ovládány znaky S/s, D/d a F/f. Tyto znaky byly vybrány proto, že jejich klávesy na PC klávesnici leží vedle sebe a dají se rychle ovládat.

V programu nejprve musíme zjistit, jestli je v bufferu nějaký přijatý znak. Pokud je, přečteme ho do proměnné znak a pro kontrolu vyšleme zpět do monitoru v PC. Následující příkaz switch … case jsme zatím nepoužili. Nahrazuje skupinu rozhodovacích příkazů if. Za case je uvedena hodnota, a pokud je shodná s hodnotou v proměnné, provede se další příkaz (blok příkazů) až po break. Za posledním case může být ještě možnost default, která se provede ve všech ostatních případech.

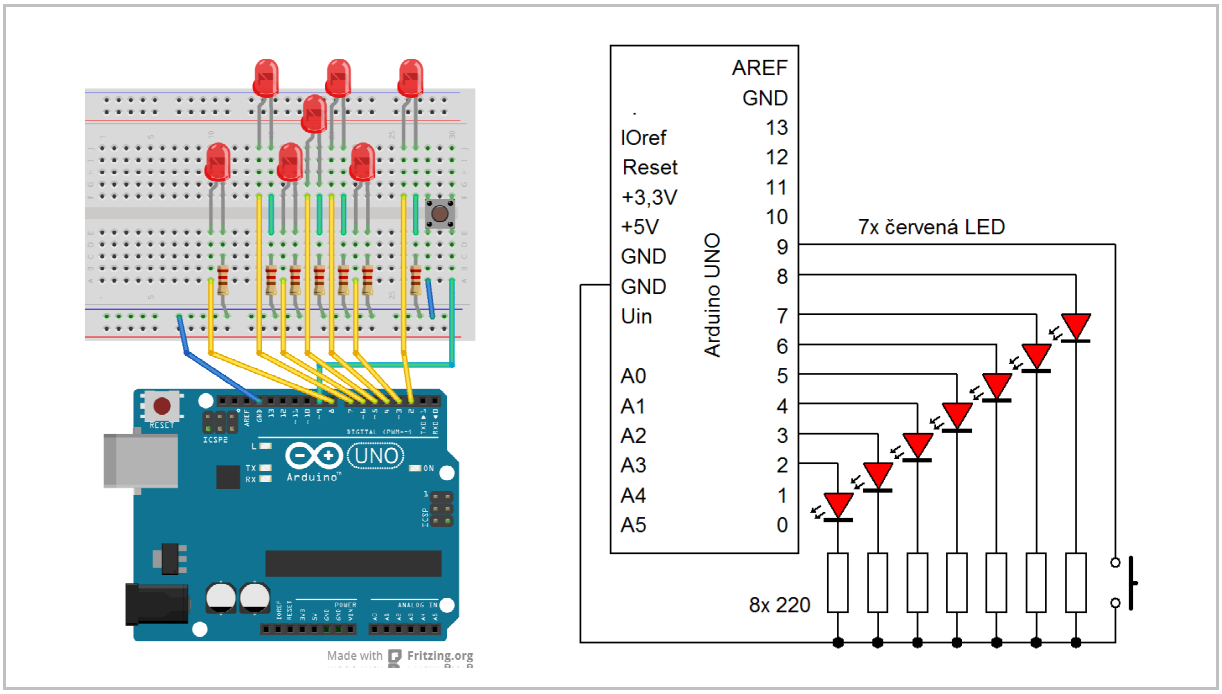
Hodnoty v monitoru zapisujeme do horního pole, odesíláme klávesou Enter nebo tlačítkem. Na vyjmenované znaky LED zareagují, ostatní nevyvolají žádnou akci, jen se odešlou zpět a vypíší jako přijatá data. Znaky můžeme sestavit i do delších řetězců.

```
// PRENOSY4 – OVLADANI LED NA ARDUINU Z PC

char znak;                                    // promenna na cteni znaku

void setup() {                                // inicializace
  Serial.begin(9600);                         // seriovy port 9600Bd
  pinMode(2,OUTPUT);                          // cervene LED jako vystup
  pinMode(3,OUTPUT);                          //
  pinMode(4,OUTPUT);                          //
  pinMode(5,OUTPUT);                          //
}

void loop() {                                 // smycka programu
  if (Serial.available()>0){                  // je cekajici znak
    znak = Serial.read();                     // precist znak
    Serial.println (znak);                    // pro kontrolu zpet
    switch (znak){                            // rozhodovani
      case 'A': digitalWrite(2,HIGH); break; // rozsvitit 1. LED
      case 'a': digitalWrite(2,LOW) ; break; // zhasnout 1. LED
      case 'S': digitalWrite(3,HIGH); break; // rozsvitit 2. LED
      case 's': digitalWrite(3,LOW) ; break; // zhasnout 2. LED
      case 'D': digitalWrite(4,HIGH); break; // rozsvitit 3. LED
      case 'd': digitalWrite(4,LOW) ; break; // zhasnout 3. LED
      case 'F': digitalWrite(5,HIGH); break; // rozsvitit 4. LED
      case 'f': digitalWrite(5,LOW) ; break; // zhasnout 4. LED
    }                                         // konec switch
  }                                           // konec akce kdyz je znak
}                                             // konec programu
```

Náměty:

- Doplňte následující povely: vysláním Q všechny LED zhasnou, vysláním W všechny LED rozsvítí.
- Využijte tlačítko v zapojení a doplňte původní program tak, aby po jeho stisknutí všechny LED zhasly.
- Nahraďte tlačítko spínačem. Bude-li sepnut, budou LED ovládány z PC, při rozepnutém spínači zůstanou LED beze změny, ovládání z PC nebude mít účinek. Vycházíme z původního programu.

Hrací kostka



Pokusíme se Arduinem nasimulovat funkci klasické hrací kostky a následně vyhodnotit, jak kvalitní je. Kvalitou se rozumí především to, aby všech šest možných výsledků hodu bylo stejně pravděpodobných. Budeme potřebovat celkem 7 LED připojených k Arduinu (opravdu 7, ne 6) a tlačítko, kterým se bude hod spouštět.

Generátor náhodných čísel – random()

Funkce random vrací (pseudo)náhodné číslo typu long. Rozsah čísel můžeme určit parametry, je-li použit jeden, bere se jako maximum, jsou-li použity dva, berou se jako minimum a maximum pro generování (pseudo)náhodného čísla. Minimum se do intervalu zahrnuje, maximum ne.

Mikrokontrolér generuje „náhodná“ čísla podle určitého algoritmu a je poměrně logické, že stejný algoritmus dojde pokaždé ke stejné posloupnosti čísel. Každé nově generované číslo určitým způsobem vychází z předchozího. Abychom zamezili stavu, kdy vždy dostaneme stejná „náhodná“ čísla (někdy je to naopak žádoucí), „podstrčíme“ generátoru na začátku svoje číslo a on pak vychází z něj. K tomu slouží funkce randomSeed. Nevrací nic, předá naše číslo z argumentu (typu long nebo int) generátoru. Když se nám podaří zajistit, aby se toto číslo odvodilo ze skutečnosti, kterou nejsme schopni oblivnit, získáme skutečný generátor náhodných čísel. Jak se to dělá? Existuje mnoho způsobů, například měříme tepplotu s přesností na setiny stupně a čísla vyjadřující teplotu v obráceném pořadí jsou oním „startovacím“ číslem generátoru. Asi nejjednodušší způsob je nasnímat čas do stisku tlačítka nebo délku stisku tlačítka. Vzhledem k tomu, že snímání času funguje s velkým rozlišením, o mnoho řádů vyšším, než je možná reakční doba člověka, lze získané číslo považovat za náhodné. Můžeme použít funkci millis s rozlišením 0,001 s.

Příklady:

randomSeed(micros()); // nastaví náhodně generátor náhodných čísel (dle času)
random(200,300); // generuje náhodné číslo mezi 200 a 299

Cyklus while

Chceme-li, aby blok příkazů byl cyklicky vykonáván tak dlouho, dokud platí uvedená podmínky, použijeme cyklus while. Jakmile je podmínka nepravdivá, program bude pokračovat. Na rozdíl od for cyklu, který používáme v případě, že je předem známo, kolikrát se má provést, while cyklus používáme, když to známo není. Podmínka se vyhodnocuje před vstupem do bloku příkazů, takže se může stát, že (je-li podmínka nesplněna už při prvním cyklu) se blok příkazů neprovede ani jednou.

Příklad:

int pocet = 0;
while(pocet < 200){ …blok příkazů. ; pocet++; } // v tomto případě se blok projde 200x

```
// KOSTKA1 - SIMULACE HRACI KOSTKY

int cislo;                      // promenna pro stav kostky

void setup() {                  // inicializace
  pinMode(2,OUTPUT);            // piny 2-8 na vystup
  pinMode(3,OUTPUT);            //
  pinMode(4,OUTPUT);            //
  pinMode(5,OUTPUT);            //
  pinMode(6,OUTPUT);            //
  pinMode(7,OUTPUT);            //
  pinMode(8,OUTPUT);            //
  pinMode(9,INPUT_PULLUP);      // pin 9 na vstup TL
  while(digitalRead(9)==HIGH){} // cekej na stisk tlacitka
  randomSeed(millis());         // inicializace random
}

void loop() {                   // smycka programu
  cislo = random(1,7);          // generovani cisla
  for (int i=2; i<9; i++){      // vypnout vsechny LED
    digitalWrite(i,LOW);        //
  }
  switch(cislo){                // zobrazeni cisla 1 - 6
  case 1: digitalWrite(5,HIGH);break;
  case 2: digitalWrite(3,HIGH);
    digitalWrite(7,HIGH);break;
  case 3: digitalWrite(2,HIGH);digitalWrite(5,HIGH);
    digitalWrite(8,HIGH);break;
  case 4: digitalWrite(2,HIGH);digitalWrite(3,HIGH);
    digitalWrite(7,HIGH);digitalWrite(8,HIGH);break;
  case 5: digitalWrite(2,HIGH);digitalWrite(3,HIGH);
    digitalWrite(7,HIGH);digitalWrite(8,HIGH);
    digitalWrite(5,HIGH);break;
  case 6: digitalWrite(2,HIGH);digitalWrite(3,HIGH);
    digitalWrite(4,HIGH);digitalWrite(6,HIGH);
    digitalWrite(7,HIGH);digitalWrite(8,HIGH);break;
```

```
  }
  delay(500);                    // zobrazit nejmene 0,5s
  while(digitalRead(9)==HIGH){} // cekat na stisk tlacitka
}                                // konec programu
```

Program po spuštění nechá všechny LED zhasnuté a čeká na stisk tlačítka, v nastavení (void setup) zůstává proti předchozím příkladům velmi dlouho. První stisk načte čas uplynulý od spuštění programu, použije ho k inicializaci generátoru náhodných čísel a rovnou přejde k vygenerování prvního čísla.

Chceme-li si ověřit, že opravdu může generátor vytvořit vícekrát po sobě stejnou posloupnost, stačí trvale držet stisknuté tlačítko a tlačítkem Reset na Arduinu restartovat program. A znovu a znovu. Ukáže se posloupnost stejných čísel, např. 2-2-6-3-5-3-1-3-6- …

Máme tedy funkční simulaci hrací kostky, ale jak ověřit „kvalitu“ náhody? Pro každou ze šesti možností, které mohou „padnout“ si založíme proměnnou. Když padne příslušné číslo, přičteme do této proměnné jedničku, takže v kterémkoli okamžiku budeme mít k dispozici počet jedniček, dvojek, … až šestek, které zatím padly. Budou-li tato čísla po velkém počtu hodů stejná (velmi podobná), je pravděpodobnost všech možností stejná. Aby bylo vyhodnocení přehledně shrnuté do jednoho čísla, vezmeme nejmenší a největší z uvedených počtů a uděláme z nich poměr. Měl by se velmi blížit číslu 1.

Aby zkouška šla přiměřeně rychle, nebudeme ani čísla nechávat delší dobu zobrazovat a čekat na stisk tlačítka, současný program „obestavíme“ cyklem, který vždy proběhne 50 000x a pak výsledek pošle k zobrazení do PC.

Pro sledování počtu „padnutí“ jednotlivých možností si zavedeme proměnnou nového typu, a to pole, v daném případě pole šesti hodnot typu unsigned long (aby bylo možné nechat prověřovat pravděpodobnost opravdu, ale opravdu dlouho).

Proměnná typu pole – array[ ]

Pole je skupina proměnných stejného názvu (a samozřejmě i typu), v níž jsou jednotlivé položky (proměnné) dostupné přes celočíselný index. Při deklaraci se může jen stanovit, kolik položek bude pole mít, nebo současně i zadat počáteční hodnotu každé položky. Zadáváme-li hodnoty, není třeba udávat počet, program si sám spočítá, kolik položek má vytvořit. Indexy píšeme do hranatých závorek.

Je třeba si pamatovat, že první položka v poli má index 0, ne 1. Potřebujeme-li kvůli zjednodušení indexovat od 1, tak jako v tomto případě, necháme proměnnou s indexem 0 nevyužitou.

Příklady:

```
int pole[6] ;                    // založí pole šesti hodnot typu int, hodnoty nezadá
                                 // proměnné jsou pole[0] až pole[5]
int pole[5] = {2, 4, -8, 3, 2}; // založí pole šesti hodnot typu int a obsadí ho počátečními
                                // hodnotami, těch je ale jen 5, což je častou chybou.
```

V následujícím programu v nulté položce pole budeme uchovávat celkový počet hodů, který se dosud vyhodnotil. Program má stejný základ jako program pro kostku, ale vlastně je úplně jiný. Tím, že vynecháme vše nepodstatné (např. zobrazení na kostce) z původního programu zbude doslova pár řádků.

```
// KOSTKA2 - SIMULACE HRACI KOSTKY S KONTROLOU PRAVDEPODOBNOSTI

unsigned long pole[] = {0,0,0,0,0,0,0};// pocitani moznosti
unsigned long nejmensi;                // hod s nejmensim zastoupenim
unsigned long nejvetsi;                // hod s nejvetsim zastoupenim
float pomer;                           // pomer minima a maxima
int cislo;                             // promenna pro stav kostky

void setup() {                         // inicializace
  Serial.begin(9600);                  // seriovy port 9600Bd
  pinMode(9,INPUT_PULLUP);             // pin 9 na vstup TL
  while(digitalRead(9)==HIGH){}        // cekej na stisk tlacitka
  randomSeed(millis());                // inicializace random
}

void loop() {                          // smycka programu
  for(unsigned int j=0; j<50000;j++){  // 50000 hodu v serii
    pole[0]++;                         // pocet hodu celkem
    cislo = random(1,7);               // generovani cisla
    pole[cislo]++;                     // pocet konkretniho hodu
  }
  Serial.print(„Pocet hodu : „);       // vypis poctu hodu celkem
  Serial.println(pole[0]);             //
  for(int k=1; k<7;k++){               // vypis konkretnic vysledku
    Serial.print(k);                   //
    Serial.print(„ padla : „);         //
    Serial.println(pole[k]);           //
  }
  nejmensi = 4294967295;               // max. mozne cislo
  nejvetsi = 0;                        // min. mozne cislo
  for(int k=1; k<7;k++){               // hledani min a max
    if (pole[k]<nejmensi) nejmensi = pole[k];
    if (pole[k]>nejvetsi) nejvetsi = pole[k];
  }
  pomer = float(nejmensi)/float(nejvetsi);  // vypocet pomeru
  Serial.print(„Pomer : „);                 // vypis pomeru
  Serial.println(pomer,6);                  // na 6 des. míst
  Serial.println(); Serial.println();       // mezera odradkovanim
}                                           // konec programu
```

Vypisovaný poměr největšího a nejmenšího zastoupení se po určité době chodu (řádově milionech hodů) ustálí na čísle vyšším než 0,99 a dál už se téměř nemění. Je to tím, že i generátor náhodných čísel pracuje s určitou periodou a jednou se začne opakovat. Pokud bychom se chtěli této vlastnosti zbavit, pak generátoru po řádově stovkách tisíc použití dodáme nové výchozí číslo (Seed) a tak vlastně použijeme nový, jiný generátor.

Funkce float použité v závěru programu ve výpočtu poměru minima a maxima převádějí hodnotu proměnné nejmensi respektive nejvetsi z libovolného typu (zde unsigned long) na typ float.

Náměty:

- Upravte program pro kostku tak, aby byla falešná. Šestka by měla padala 2x častěji než ostatní čísla.
- Upravte stejným způsobem jako v předchozím námětu program pro kontrolu kostky a ověřte výsledek na sérii nejméně 500000 hodů.
- Upravte program pro kostku respektive kontrolní tak, aby kostka byla falešná, ale ne tak nápadně jako v předchozím případě. Největší pravděpopodobnost budou mít čísla 3 a 4, o něco menší 2 a 5, nejmenší pravděpodobnost čísla 1 a 6. Nápověda: zkuste například, co se stane, když vygenerujete dvě náhodná čísla, jedno v rozsahu 1 až 3, druhé v rozsahu 1 až 4, sečte je a od výsledku odečtete 1.
- Zkuste vymyslet způsob, jak změnit pravděpodobnost „padnutí“ čísel na kostce tak, aby čím větší číslo, tím větší mělo pravděpodobnost. Ověřte výsledek na sérii nejméně 500000 hodů.
- Do zapojení doplňte generátor kmitů s obvodem 555 stejný jako jsme již dělali, použijte v něm kondenzátor 100 nF. Výstup bude připojen k pinu 13. Napište vlastní program pro (poctivou) hrací kostku, který ale bude pracovat na zcela jiném principu bez použití genrátoru (pseudo)náhodného čísla. Kostka ukáže číslo vždy po stisku tlačítka. Od zobrazení do dalšího stisku tlačítka Arduino počítá pulzy přicházející z generátoru, ke zbytku po dělení šesti přičte jedničku a tak získá výsledné náhodné číslo. Vzhledem k tomu jak jsou pulzy rychlé, výsledek nejde ovlivnit.
- Ověřte kvalitu tohoto způsobu generování náhody a výsledek zobrazte stejně jako v programu kostka2. Ručně řízených „hodů“ pro zkoušku bude méně, stačí asi 100.

Píšeme vlastní podprogramy a funkce

Dostali jsme se už k takové složitosti programů, že začínají ztrácet přehlednost a vyplatilo by se některé jejich části „osamostatnit“ a mít možnost je volat jediným příkazem vícekrát z různých míst programu. Pokud zavolaný úsek programu nevrací žádnou hodnotu, například jen rozsvítí určitou kombinaci LED podle parametru, budeme mluvit o podprogramu. Pokud se vrací hodnota daného typu, budeme mluvit o funkci.

Typickým příkladem, kdy by bylo vhodné použít podprogram, je úsek pro zobrazení stavu na kostce se sedmi LED. Tak jak je napsán se projde jen jednou při zobrazení čísla. Pokud bychom chtěli třeba vytvořit efekt, který rychle střídá zobrazená čísla až se nakonec zastaví to, které má zobrazit, musel by se tento rozsáhlý úsek dát do programu vícekrát. My z něj uděláme podprogram:

```
void neco() {…příkazy…}    // tento podprogram s názvem neco nemá žádný parametr
                           // a takto se volá … neco();
void neco(byte cas, long cas) { …příkazy…}  // podprogram s názvem neco má dva
          // parametry, první je typu byte a znamená číslo pinu, druhý typu long a bude
          // mít asi nějakou spojitost s časovým úsekem
          // a takto se volá … neco(1,5624) … provede se na pinu 1, čas bude 5624
```

Funkce se dělají úplně stejně, pokud si uvědomíme že slovo void nemá jiný význam než pustota, prázdnota, prostě nic. Znamená to v tomto případě, že nic se nevrací. Chceme-li, aby podprogram vracel hodnotu, musíme určit její typ … a máme funkci. Jak podprogramy tak funkce by měly být zapsány v programu dřív (výš), než jsou použity.

```
int soucet(int prvni, int druhy) {soucet = prvni + druhy;} // tato kompletní funkce vrací
          // hodnotu typu integer, jmenuje se soucet a dávají se jí dva parametry, oba
          // typu integer. Výsledkem je součet obou parametrů.
```

A teď konkrétní ukázka na předchozím programu pro kostku. V něm dvakrát potřebujeme, aby program čekal až do stisknutí tlačítka na pinu 9. Uděláme si tedy podprogram, který pak stačí dvakrát zavolat.

```
void cekej(){while(digitalRead(9)==HIGH){}}
```

nebo obecněji jako podprogram, který čeká na stisk tlačítka na zadaném pinu:

```
void cekej(int pin){while(digitalRead(pin)==HIGH){}}
```

Druhým a rozsáhlejším příkladem je část programu, která se stará o rozsvícení LED podle toho, jaké číslo padlo (číslo je v proměnné cislo a je typu integer).

Původní úsek vypadá takto:

```
switch(cislo){                    // zobrazeni cisla 1 – 6
  case 1: digitalWrite(5,HIGH);break;
  case 2: digitalWrite(3,HIGH);
```

```
    digitalWrite(7,HIGH);break;
  case 3: digitalWrite(2,HIGH);digitalWrite(5,HIGH);
    digitalWrite(8,HIGH);break;
  case 4: digitalWrite(2,HIGH);digitalWrite(3,HIGH);
    digitalWrite(7,HIGH);digitalWrite(8,HIGH);break;
  case 5: digitalWrite(2,HIGH);digitalWrite(3,HIGH);
    digitalWrite(7,HIGH);digitalWrite(8,HIGH);
    digitalWrite(5,HIGH);break;
  case 6: digitalWrite(2,HIGH);digitalWrite(3,HIGH);
    digitalWrite(4,HIGH);digitalWrite(6,HIGH);
    digitalWrite(7,HIGH);digitalWrite(8,HIGH);break;
 }
```

My z něj uděláme podprogram a parametrem:

```
void zobraz(int kolik){
  switch(cislo){                    // zobrazeni cisla 1 – 6
    case 1: digitalWrite(5,HIGH);break;
    // … tady je vše stejné jako bylo pro hodnoty 2 – 5
    case 6: digitalWrite(2,HIGH);digitalWrite(3,HIGH);
      digitalWrite(4,HIGH);digitalWrite(6,HIGH);
      digitalWrite(7,HIGH);digitalWrite(8,HIGH);break;
  }                                 // konec switch
}                                   // konec podprogramu
```

To ale nemusí být konec. I podprogram může mít svoje podprogramy a my si všimneme, že některá čísla se dají udělat z nižších jednoduchým doplněním. Takže lze napsat třeba rozsvícení čísla 6 jako rozsvícení čísla 4 a k tomu pak doplnit dvě LED. Když odladěné části se samostatnou činností přeměníme na podprogramy a funkce, „vyčistí“ a zpřehlední se program pro další úpravy. Jak bude vypadat program pro kostku zapsaný s využitím podprogramů?

```
// KOSTKA3 - SIMULACE HRACI KOSTKY S PODPROGRAMY

int cislo;                              // promenna pro stav kostky

void cekejTL() {                        // cekani na stisk tlacitka
  while(digitalRead(9)==HIGH){}         //
}

void setup() {                          // inicializace
  pinMode(2,OUTPUT);                    // piny 2-8 na vystup
  pinMode(3,OUTPUT);                    //
  pinMode(4,OUTPUT);                    //
  pinMode(5,OUTPUT);                    //
  pinMode(6,OUTPUT);                    //
  pinMode(7,OUTPUT);                    //
  pinMode(8,OUTPUT);                    //
  pinMode(9,INPUT_PULLUP);              // pin 9 na vstup TL
  cekejTL();                            // cekej na stisk tlacitka
  randomSeed(millis());                 // inicializace random
}

void zobraz(int kolik){                 // zobrazeni cisla 1 - 6
  for (int i=2; i<9; i++){              // vypnout vsechny LED
    digitalWrite(i,LOW);}               //
  switch(cislo){                        // rozdeleni na 6 moznosti
    case 1: digitalWrite(5,HIGH);break;
    case 2: digitalWrite(3,HIGH);
      digitalWrite(7,HIGH);break;
    case 3: digitalWrite(2,HIGH);digitalWrite(5,HIGH);
      digitalWrite(8,HIGH);break;
    case 4: digitalWrite(2,HIGH);digitalWrite(3,HIGH);
      digitalWrite(7,HIGH);digitalWrite(8,HIGH);break;
    case 5: digitalWrite(2,HIGH);digitalWrite(3,HIGH);
      digitalWrite(7,HIGH);digitalWrite(8,HIGH);
      digitalWrite(5,HIGH);break;
    case 6: digitalWrite(2,HIGH);digitalWrite(3,HIGH);
      digitalWrite(4,HIGH);digitalWrite(6,HIGH);
      digitalWrite(7,HIGH);digitalWrite(8,HIGH);break;
  }                                     // konec pro switch
}                                       // konec podprogramu zobraz

void loop() {                           // smycka programu
  cislo = random(1,7);                  // generovani cisla
  zobraz(cislo);                        // zobraz cislo na LED
  delay(500);                           // udrzet nejmene 0,5s
  cekejTL();                            // cekat na stisk tlacitka
}                                       // konec programu
```

Náměty:

- Doplňte do programu efekt – při zobrazení rychle proběhnou všechna čísla až se zastaví na tom, které „padlo“ podle generátoru.

Práce s napětím – AD převodníky

Zatím jsme používali pouze digitální vstupy a výstupy Arduina. V praxi v mnoha případech nepracujeme jen s digitálními signály, ale také se signály analogovými, tedy spojitými, kde je informace nesena například velikostí napětí. Aby mohl mikrokontrolér Arduina se spojitou veličinou (napětím) pracovat, musí si ji nejdřív převést na číslo; k tomu slouží analogově-digitální (AD) převodníky.

AD převodníky Arduina pracují s rozlišením 10 bitů, to znamená, že poskytují číslo v rozsahu 0 až 1 023. Nulovému napětí odpovídá vždy hodnota 0, ale jaké napětí odpovídá maximu, to záleží na tom, jaké referenční napětí se vezme. V nejjednodušších a na přesnost nejméně náročných případech se bere jako referenční napájecí napětí Arduina a považuje se za napětí +5,00 V (což nikdy s takovou přesností splněno není). Kromě této možnosti lze u Arduina Uno využít vnitřní referenční zdroj napětí 1,1 V nebo přepojit na vnější referenční napětí, které dodáme zvenčí na pin AREF (musí být menší než napájecí napětí +5 V).

Pokud bereme jako referenci napětí +5 V, potom jednomu bitu v rozlišení AD převodníku odpovídá 5/1023 = 0,004 887 585 V. To není číslo, s nímž by se dobře pracovalo. V přesnějších aplikacích je lepší dát přednost externímu referenčnímu napětí, které může mít menší závislot na teplotě, a nastavit jej třeba na 2,56 V, takže jednomu bitu odpovídá přesně 2,5 mV.

Pro jednoduché zkoušení budeme potřebovat nastavitelný zdroj napětí. Poslouží nám odporový trimr, jehož krajní vývody zapojíme na potenciál 0 V (GND) a napětí 3,3 V Arduina. Nepoužijeme +5 V, napětí 3,3 V je při napájení z USB přesnější a umožní mám lépe vyzkoušet různé reference. Nastavitelné napětí budeme snímat z třetího, prostředního vývodu trimru.

Nastavení zdroje referenčního napětí – analogReference()

Příkaz analogReference má jeden parametr. Pro Arduino UNO má smysl použít tyto možnosti:

DEFAULT – jako referenční se použije napájecí napětí +5 V

INTERNAL – jako referenční se použije napětí 1,1 V z vnitřního referenčního zdroje

EXTERNAL – jako referenční se použije napětí z pinu AREF (musí být <5 V)

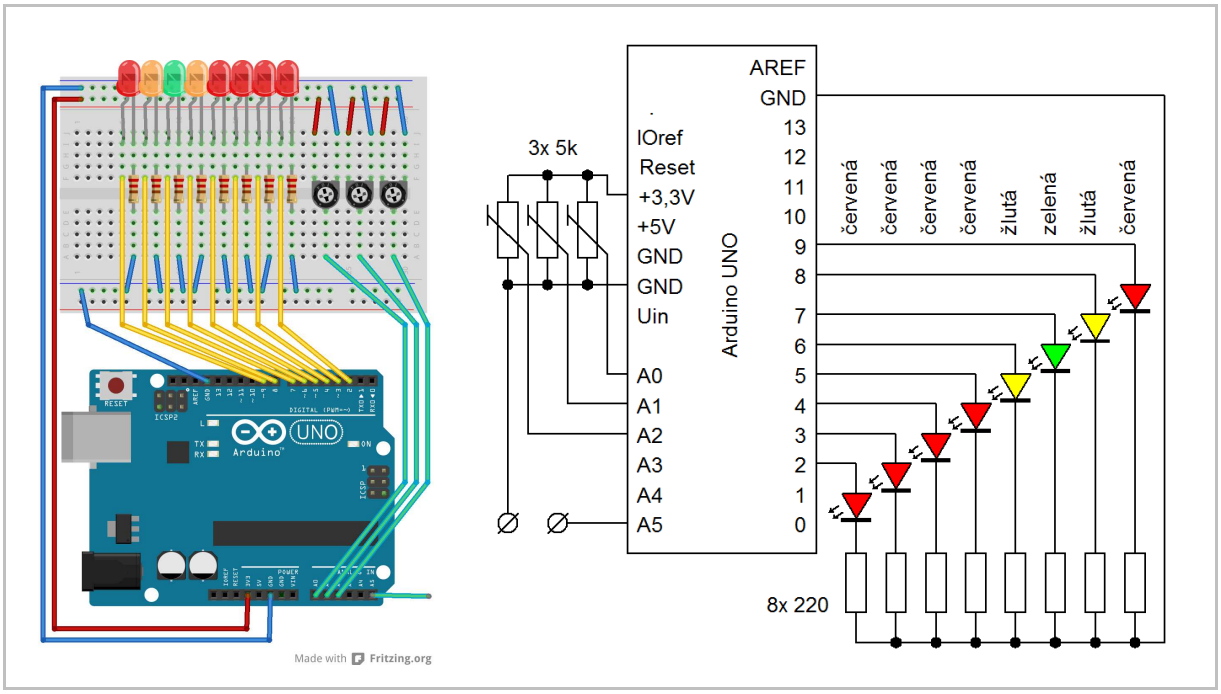
Čtení údaje z AD převodníku – analogRead()

Funkce analogRead potřebuje jako parametr číslo pinu, na který se má obrátit (0 až 5) a vrací číslo 0 až 1 023 typu integer, úměrné měřenému napětí a referenci.

Příklad:

```
napeti = analogRead(2);  // přečte napětí z analogového pinu A2 a uloží do proměnné
                         // napěti
```

Připravíme si zapojení podle obrázku, bude sloužit pro několik následujících úloh. Zatím využijeme jen jeden odporový trimr na pinu A0 a budeme změřené napětí odesílat do PC.




```
// ADC1 MERENI NAPETI S RUZNOU REFERENCI

void setup(){                       // nastaveni
  Serial.begin(9600);               // seriova linka 9600Bd
  analogReference(DEFAULT);         // reference napajeci napeti
}

void loop(){                        // program
  Serial.println(analogRead(0));    // vypis napeti v cisle
  delay(500);                       // pockat 0,5 s
}                                   // konec programu
```

V jedné krajní poloze trimru bychom měli naměřit 0, v druhé něco kolem 710, protože trimr je připojen na napájecí napětí 3,3 V a jako reference je bráno napájecí napětí přibližně 5 V. Když změníme regerenční napětí na INTERNAL (1,1 V), bude stále na jednom konci trimru 0, ale na druhém 1 023. Hodnota se mění jen asi v jedné třetině výchylky trimru. Nastavíme EXTERNAL referenční zdroj a spojíme pin AREF s napětím 3,3 V. Získávaná hodnota napětí by měla zahrnovat celý rozsah od 0 do 1 023, ale tentokrát by se měla měnit v celém rozsahu výchylky. Tuto referenci ponecháme.

Když se budeme snažit nastavit libovolnou hodnotu, brzy zjistíme, že u okrajů výchylek je to obtížné nebo nemožné. Není to chyba Arduina nebo převodníku, ale odporové dráhy trimrů, které přibližně v krajních 10% dráhy nemusí mít plynulou změnu odporu.

Zkusíme přepočítat údaj, který vrací převodník, na reálné napětí. Jeden bit změny odpovídá napětí 3,3 / 1 023 = 0,003 226 V. Tuto hodnotu zavedeme do programu. Když nezadáme formátování výpisu, budou výsledná čísla na dvě desetinná místa. Z výpisu do PC vytvoříme samostatný podprogram.

```
// ADC2 MERENI NAPETI S PREPOCTEM NA V

float krok = 3.3/1023;                    // krok pri referenci 3,3V
float napeti0;                            // promenna pro napeti0

void setup(){                             // nastaveni
  Serial.begin(9600);                     // seriova linka 9600Bd
  analogReference(EXTERNAL);              // reference napajeci napeti
}

void vypis(float cislo){                  // vypis napeti v monitoru
  Serial.print(„Napeti na vstupu 0 : „); // uvodni text
  Serial.print(napeti0);                  // vypis napeti ve V
  Serial.println(„ V“);                   // vypis jednotek
}

void loop(){                              // program
  napeti0 = krok * analogRead(0);         // nacteni a vypocet napeti ve V
  vypis(napeti0);                         // vypis do PC
  delay(500);                             // pockat 0,5 s
}                                          // konec programu
```

Náměty:

- Upravte program tak, aby se napětí vypisovalo na 3 desetinná místa. Všimněte si, že vzhledem k tomu, že tento výpis má už lepší rozlišení než snímání napětí, nejsou na posledních desetinných místech libovolné hodnoty, ale střídají se tam jen určité kombinace. Zaokrouhlení na 2 desetinná místa pro tento případ bylo přiměřené.
- Vytvořte podprogram graf, který dostane stejně jako podprogram vypis hodnotu napětí, ale zobrazí ji jinak. Vypíše do řádku tolik znaků x, kolik odpovídá napětí, jednotkou je 0,05 V. Při pohybu trimrem by se měl vytvořit pomalu rolující graf se záznamem napětí (na vodorovné ose je napětí, na svislé čas).
- Doplňte do původního zapojení generátor kmitů s obvodem 555 a kondenzátorem 470 µF, který jsme už používali. K analogovému vstupu A3 připojte vývod 2 obvodu 555 (ne vývod 3, který je výstupem a který jsme zatím používali). Nechte vypisovat napětí na tomto vývodu, mělo by se cyklicky plynule měnit přibližně mezi 1/3 a 2/3 napájecího napětí, takže mezi 1,7 a 3,3 V.
- Zkombinujte předchozí dva náměty, napětí z doplněného generátoru s obvodem 555 přivedené na A3 zobrazujte graficky pomocí znaků x. Ověřte rozsah, v němž se napětí, pohybuje.

V dalším kroku využijeme řadu LED. Vytvoříme podprogram zobraz, který rozsvítí sloupeček LED v délce odpovídající hodnotě napětí. Každá LED by měla odpovídat intervalu 0,5 V (měří se od nuly).

K této úloze můžeme přistoupit dvěma způsoby. První možnost je, že si řekneme: první LED svítí v intervalu 0 až 0,5 V, druhá v intervalu 0,5 až 1,0 V, třetí pro 1,0 až 1,5 V atd., přičemž mez patří vždy jen do jednoho z intervalů (nikdy nesvítí dvě LED současně). Takto program vytvoříme, protože je to jednodušší. Druhou možností je brát meze tak, že první LED ukazuje napětí 0,5 V s určitou přesností, v našem případě je to ±0,25 V, druhá LED napětí 1,0 V ±0,25 V atd. To znamená, že první LED má svítit od 0,25 do 0,75 V, druhá od 0,75 do 1,25 V, atd. Protože ale tyto intervaly nezahrnují nulu, při napětí menším než 0,25 V by nesvítilo nic a nebylo by vidět, že indikace napětí funguje. V praxi se obvykle první LED bere jako „napětí je menší než“. Nás program bude zatím podle první, jednodušeji programovatelené možnosti. Aby LED indikace pracovala bez zpoždění, budeme muset udělat časování výpisu do PC bez použití delay.

```
// ADC3 MERENI NAPETI S PREPOCTEM NA V A ZOBRAZENIM NA LED SLOUPCI

float krok = 3.3/1023;                  // krok pri referenci 3,3V
float napeti0;                          // promenna pro napeti A0

void setup(){                           // nastaveni
  Serial.begin(9600);                   // seriova linka 9600Bd
  analogReference(EXTERNAL);            // reference napajeci napeti
  for (int i = 2; i < 10; i++){         // piny 2 - 9 na vystup
    pinMode(i,OUTPUT);}                 //
}

void vypis(float cislo){                // vypis napeti v monitoru
  Serial.print(„Napeti na vstupu 0 : „);// uvodni text
  Serial.print(napeti0);                // vypis napeti ve V
  Serial.println(„ V“);                 // vypis jednotek
}

void zobraz(float cislo){               // zobrazeni na pěti na LED
  for (int i = 2; i < 10; i++){         // zhasnout vsechny LED
    digitalWrite(i,LOW);}               //
    digitalWrite(int(cislo*2+2),HIGH);  // rozsvit LED dle cislo
}

void loop(){                            // program
  napeti0 = krok * analogRead(0);       // nacteni a vypocet napeti ve V
  zobraz(napeti0);                      // zobrazeni na LED
  if (millis() % 500 == 0){             // jednou za 0,5s
    vypis(napeti0);                     // vypis napěti
    delay(1);}                          // pockat na dalsi 0,001s
}                                       // konec programu
```

Náměty:

- Upravte program tak, aby pracoval podle druhého způsobu, aby hodnoty 0,5, 1,0, 1,5 V atd. byly uprostřed intervalu, kdy příslušná LED svítí
- Přepojte příkazem referenční napětí na 5 V a upravte program tak, aby zelená LED svítila při napětí 4 V
- Nastavte referenční napětí 5 V. Upravte program tak, aby LED stupnice nezobrazovala napětí od nuly, ale každá LED odpovídala stupni 0,1 V. Zelená LED by měla svítit při napětí 3,3 V, to máme na zdroji Arduina. Máte-li k dispozici dostatečně přesný voltmetr, ověřte meze přepínání mezi svitem jednotlivých LED.
- Doplňte do zapojení generátor kmitů s obvodem 555 a kondenzátorem 470 µF, který jsme už používali. K analogovému vstupu A3 připojte vývod 2 obvodu 555. Nechte zobrazit na řadě LED napětí na tomto vývodu – bude vidět, že rozsah není plně využit k zobrazení změn.

V následující úloze této série budeme stále používat trimr připojený k A0 jako simulaci vstupního napětí indikátoru, ale meze pro spodní i horní LED z řady nebudou určeny pevně programem. Spodní mez napětí určí nastavení trimru připojeného k A1, horní mez trimr na A2. Nastal čas seznámit se s dalším chytrým příkazem.

Transformace rozsahu – map()

Případy, kdy potřebujeme „změnit rozsah“ hodnoty nějaké proměnné, jsou velmi časté. Arduino má připravenou funkci, která to dělá přímo. Funkce má pět parametrů v daném pořadí, všechny jsou typu long stejně jako vracený výsledek:

1. vstupní hodnota
2. první mez vstupu
3. druhá mez vstupu
4. první mez výstupu
5. druhá mez výstupu

Parametry se záměrně nenazývají spodní a horní meze, protože to není podmínkou. Funkce transformuje první mez vstupu na první mez výstupu, druhou mez vstupu na druhou mez výstupu, vše mezi nimi dostane úměrnou hodnotu. Funkce nekontroluje a neomezuje to, jestli je vstupní hodnota ve stanovených mezích a může se potom stát, že výstupní hodnota nebude ve výstupních mezích. V podstatě dělá funkce map() toto:

```
return (x – in_min) * (out_max – out_min) / (in_max – in_min) + out_min;
```

Příklady:

map(x,0,100,100,0)	„obrátí“ hodnotu x v intervalu 0-100, z 2 bude 98 apod.
map(x,0,1023,0,100)	„stlačí“ interval 0-1 023 na 0-100 (typické pro výstup z AD převodníku transformovaný na procenta napětí)
map(x,452,958,5,-380)	„otočí, posune a stlačí“ interval 452-958 na rozsah 5 až -380
map(x, 1,10,51,60)	pro x ze zadaného intervalu hodnoty jen posune, ale např. pro x=12 vrátí hodnotu 62 (mimo výstupní interval) a není to chyba

Kontrola mezí – constrain()

Pokud požadujeme, aby nějaká hodnota (typicky v proměnné) byla určitě v zadaném rozsahu, pak si ti můžeme zajistit napsaným programem třeba takto:

if (x< minimum) {x=minimum};

if (x> maximum) {x=maximum};

Funkce constrain dělá pro všechny datové typy totéž a zapisuje se takto:

constrain(x, minimum, maximum);

Příklad:

constrain(5,10,20)	vrátí hodnotu 10 (minimum intervalu)
constrain(99,10,20)	vrátí hodnotu 20 (maximum intervalu)
constrain(12,10,20)	vrátí hodnotu 12 (je z intervalu)

Budeme-li řešit znovu zobrazení napětí na řadě LED, nemusíme už vlastně ani programem přepočítávat údaje z AD převodníku na napětí, stejně tak nemusíme přepočítávat meze snímané z dalších dvou převodníků. Omezením intervalu se nemusíme zabývat, údaj z AD převodníku bude vždy v intervalu 0 až 1 023, to plyne z podstaty jeho funkce, ale určitě budeme muset ošetřit rozsah ovládání LED. Přenosem údaje do PC už se nebudeme zabývat a všechny žluté a zelené LED v zapojení můžeme vyměnit za červené.

```
// ADC4 MERENI NAPETI V MEZICH ZADANYCH TRIMRY

void setup(){                       // nastaveni
  analogReference(DEFAULT);         // reference napajeci napeti
  for (int i = 2; i < 10; i++){     // piny 2 - 9 na vystup
    pinMode(i,OUTPUT);}             //
}

void loop(){                        // program
  for (int i = 2; i < 10; i++){     // LED 2 - 9 zhasnout
    digitalWrite(i,LOW);}           //

digitalWrite(constrain(map(analogRead(0),analogRead(1),analogRead(2),2,9),2,9),HIGH
);
//                            +--------------------------------------+
//                                          cteni hodnot
//                         +-----------------------------------------------+
//                                          zmena rozsahu
//             +-------------------------------------------------------------+
//                                          omezeni hodnot pro LED
//+-----------------------------------------------------------------------------+
//                                          rozsviceni jedne LED
//
delay(100);                         // pockej 0,1s
}                                   // konec programu
```

Podívejme se na výkonnou část programu. Nejdříve se v cyklu zhasnou všechny LED, což je jednoduchá a dobře čitelná činnost, a pak se celý zbytek programu vtěsná do jediné řádky respektive do jediného příkazu digitalWrite. Nepotřebovali jsme ani jednu pracovní proměnnou, program je velmi efektivní, ale také krajně nepřehledný. Pokud chceme podobně napsanou část programu přečíst a pochopit, musíme začít uvnitř v nejhlouběji zanořené úrovni závorek a pak se postupně „prokousat“ až ven. V tomto případě ukazují komentáře níže, jaký je rozsah jednotlivých do sebe zanořených příkazů a funkcí. Snad to alespoň trochu pomůže při prvním pokusu o rozklíčování podobného způsobu zápisu.

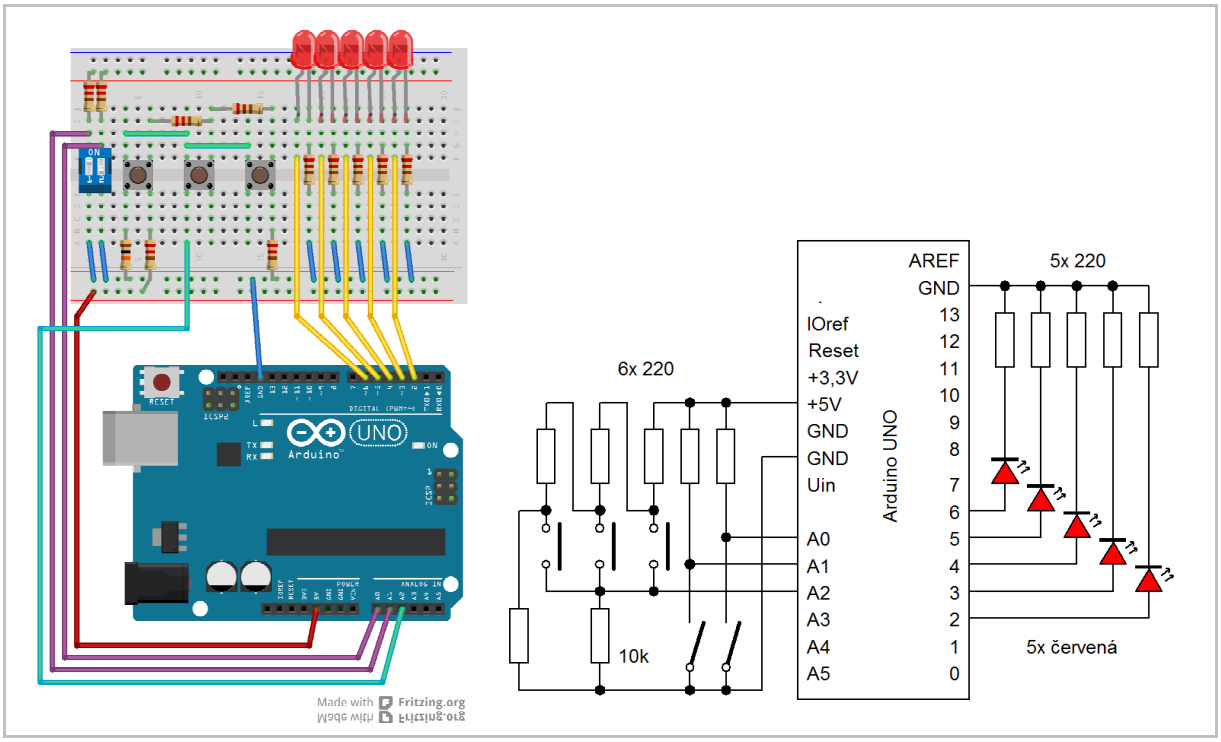
Nastavením trimrů řídících meze můžeme dosáhnout prakticky libovolné funkce včetně obráceného „pohybu“ signalizace nebo změny rozsahy co do šířky i posunutí, potřebujeme-li však nastavit indikátor na přesné napětí, bez voltmetru se už neobejdeme.

Náměty:

- Upravte předchozí program tak, aby indikace napětí pracovala v rozsahu 2 až 3,4 V. Místo nastavení mezí trimry použijte konstanty (hodnoty zadané přímo do výrazu). Předpokládáme, že napájecí napětí je přesně 5 V.
- Upravte předchozí program tak, aby indikace pracovala v pevném rozsahu 0 až 5 V (rozsah napájecího napětí), vstupem je stále napětí snímané vstupem A0 z prvního trimru. Svítící LED bude ukazovat aktuální hodnotu napětí a maximální dosaženou hodnotu od spuštění programu bude ukazovat rychle blikající LED. Indikátor tedy zobrazí současně jak okamžité napětí tak maximální. Pokud by stejná LED měla současně blikat i trvale svítit, bude blikat.
- Napište nový program, který využije ze stávajícího zapojení jen LED na výstupech 2 až 5. LED na výstupu 2 bude blikat 1x za sekundu, rychlost blikání ostatních LED bude vzájemně nezávisle nastavitelná jednotlivými trimry v rozsahu 10x za sekundu až 1x za deset sekund.
- Doplňte do zapojení generátor kmitů s obvodem 555 a kondenzátorem 470 µF, který jsme už používali. K analogovému vstupu A3 připojte vývod 2 obvodu 555. Nechte na řadě LED zobrazit změny napětí na vývodu A3 tak, aby se plně využil rozsah všech LED. Nápověda: Využijte funkci map a počítejte s tím, že napětí na vstupu se mění od 1/3 do 2/3 napájecího napětí, takže přibližně mezi 1,7 a 3,3 V.

Analogová simulace tlačítka

Tlačítko (nebo spínač) je typické vstupní zařízení připojované na digitální vstupy. Piny 0 až 13 můžeme používat volně jako vstupy nebo výstupy, i když určité omezení v tom je, když budeme chtít zachovat funkci sériové linky (spojení s PC), musíme se vyhnout pinům 0 a 1. Není nijak vyjímečná situace, kdy budeme pociťovat nedostatek vstupů, ale nebudeme potřebovat 6 analogových vstupů A0 až A5. Co v takovém případě můžeme dělat?



Připravíme si nové zapojení, v němž uplatníme tři tlačítka i dvojitý spínač, vše připojené na analogové vstupy. Na výstupech 2 až 6 ponecháme červené LED. Máme pět ovládacích prvků a pět LED, úkol je jednoduchý, zajistit, aby každý ze spínačů i tlačítek ovládal svou LED.

V prvním kroku použijeme analogový vstup tak jak je jako digitální, aniž bychom měnili způsob jeho obsluhy. Spínač spojuje analogový vstup ze zemí a současně je připojen na +5 V pull-up rezistorem. V našem případě jsme použili rezistory s odporem 220 Ω, nevadí to, jen zbytečně zvyšuje spotřebu proudu. V praxi je obvyklejší větší odpor třeba 2k2 nebo 4k7.

Ze vstupu přečteme napětí respektive stav AD převodníku, je-li spínač sepnut, bude výsledek nulový (velmi blízký nule), je-li rozpojen, bude výsledek blízký 1 023.

Jeden analogový vstup ale může posloužit pro několik tlačítek, v našem případě vyzkoušíme (jen) tři. Ze čtyř rezistorů 220 Ω je složený dělič napětí a tlačítka jedním pólem spojená připojují na analogový vstup tato napětí. Aby napětí na vstupu bylo jasně definované i když žádné z tlačítek není sepnuto, musí být vstup spojený s jedním nebo druhým koncem děliče přes odpor podstatně větší, než mají rezistory v děliči. V našem případě je spojen se zemí.

Jaké napětí detekujeme v jednotlivých situacích? Bez stisknutého tlačítka to bude (téměř) nula, to zajistí rezistor 10 k. Se stisknutým prvním tlačítkem (nejblíž nule) to bude přibližně ¼ napájecího napětí, tedy asi 1,25 V, s druhým tlačítkem ½ napájení, tedy 2,5 V, s třetím tlačítkem ¾ napájení, kolem 3,75 V. Omezením je, že nesmí být stisknuto současně víc než jedno tlačítko. Žádná kombinace stisku tlačítek nezpůsobí zkrat nebo jiný důvod k poškození Arduina.

Zkusme porušit podmínku jednoho stisknutého tlačítka. Když stiskmene dvě spodní, bude na výstupu napětí 1/3 napájení (1,66V), když dvě horní, tak 2/3 napájení (3,33V). Stisk krajní dvojice nebo všech tří dá na vstup AD převodníku přibližně ½ napájecího napětí, to odpovídá prostřednímu tlačítku a nové možnosti to nepřinese. Ukazuje se tedy, že tímto způsobem je možné detekovat s jedním AD vstupem nejen tři připojená tlačítka, ale navíc ještě další dvě kombinace dvou tlačítek. Má-li rezistor, který se stará o úroveň vstupu, když není stisknuto žádné tlačítko, dostatečně vysokou hodnotu odporu (u nás 10k), jeho vliv na posunutí napětí snímaného při stisku tlačítek je malý a lze zanedbat.

Kolik tlačítek je možné jedním AD vstupem obsloužit? Teoreticky třeba i 20, ale aby byla detekce spolehlivá i při určité toleranci součástek a změnách (rušení) na napájení, nebývá rozumné dávat na jeden vstup víc než 4 tlačítka a jako maximum lze uvést 10 (číslicová klávesnice). Pro 4 tlačítka vychází výsledky velmi pěkně, jednotlivá tlačítka dají napětí 1 – 2 – 3 – 4 V, kombinace dvou sousedních pak 1,25 – 1,33 – 2,66 – 3,75 V. Nejprve připravíme program pro jen jeden spínač a jednu LED tak, aby byl přehledný a dobře čitelný.

```
// ADC5 NEJJEDNODUSSI OBSLUHA SPINACE NA AD

void setup(){                       // nastaveni
  analogReference(DEFAULT);         // reference napajeci napeti
  for (int i = 2; i < 7; i++){      // piny 2 - 6 na vystup
    pinMode(i,OUTPUT);}             //
}

void loop(){                        // program
  if (analogRead(0)>512)            // porovnani AD s polovinou rozsahu
    digitalWrite(2,HIGH);           // kdyz je vice zapni LED
    else {digitalWrite(2,LOW);      // kdyz je mene vypni LED
  }                                 // konec if
}                                   // konec programu
```

A pak už plný program pro dva spínače a tři tlačítka, ale jiným způsobem. Stisky kombinací tlačítek neuvažujeme.

```
// ADC6 OBSLUHA 2 SPINACU A 3 TLACITEK NA AD

void setup(){                     // nastaveni
  analogReference(DEFAULT);       // reference napajeci napeti
  for (int i = 2; i < 7; i++){    // piny 2 - 6 na vystup
    pinMode(i,OUTPUT);}           //
}

void LED (int pin, int vstupAD, int mez, int tolerance){  // obsluha LED
  if (abs(mez-analogRead(vstupAD))<tolerance)     // porovnani tolerance
    digitalWrite(pin,HIGH);       // zapnout LED - je v toleranci
    else digitalWrite(pin,LOW); // vypnout LED - je mimo
}

void loop(){                      // program
  LED(2,0,1023,300);              // LED na pin 2 dle A0
  LED(3,1,1023,300);              // LED na pin 3 dle A1
  LED(4,2,256,20);                // LED na pin 4 dle A2 a TL1
  LED(5,2,512,20);                // LED na pin 5 dle A2 a TL2
```

```
  LED(6,2,768,20);              // LED na pin 6 dle A2 a TL3
}                               // konec programu
```

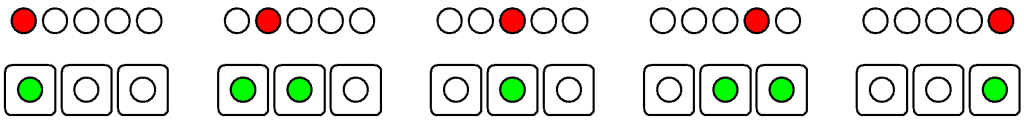
U posledního programu má smysl se zastavit. Podprogram „LED“ má čtyři parametry, prvním je pin, na kterém je příslušná LED (v našem případě 2 až 6), druhým je pin vstupu AD, na němž měříme napětí (v našem případě 0 až 2), třetí je hodnota z AD převodníku, která je očekávaná pro danou LED (jinak řešeno vyhodnocení stisku tlačítka nebo kombinace tlačítek) a posledním parametrem je tolerance, v níž budeme brát hodnotu z AD převodníku za správnou.

Podprogram „LED“ se volá i pro spínače, v jejich případě lze očekávat z AD převodníku jen hodnoty 0 a 1023, proto je stanovena velká tolerance 300, takže pro údaj >723 bude vyhodnocen jako rozpojení spínače a rozsvícení LED, v ostatních případech LED zhasne. U tlačítek přiravuje podprogram možnost použít i jiný počet než tři. AD převodních je ve všech případech stejný, liší se očekávaná hodnota napětí a tolerance je poměrně malá, dokonce mezi hodnotami poskytovanými tlačítky jsou „pásma nikoho“, kdy nesvítí žádná LED. Můžeme se i tom jednoduše přesvědčit, když stiskneme současně krajní a prostřední tlačítko

Analogově snímané spínače a tlačítka vyžadují komplikovanější zapojení, ale výrazně šetří počet nutných vstupních digitálních pinů, které pak mohou být využity třeba jako piny výstupní.

Náměty:

- Vyzkoušejte, jak malé tolerance stačí na to, aby program správně a spolehlivě pracoval s tlačítky. Zkoušíte tím vlastně, jak velký vliv má reálný odpor 10 k na posunutí napětí.
- Nastavte pomocí tolerance (případně jemnou úpravou mezí) vyhodnocení tak, aby se při stisku prostředního tlačítka rozsvítily všechny tři LED ovládané tlačítky. Funkce krajních tlačítek jen na svoje LED musí zůstat zachovány.
- Upravte původní program. Spínače nebudou mít žádnou funkci, všech pět LED bude ovládáno tlačítky. První se rozsvítí při stisku 1. tlačítka samostatně, druhá při stisku 1. a současně 2. tlačítka, třetí při stisku jen 2. tlačítka, čtvrtá při stisku 2. a 3. tlačítka současně a pátá při samostatném stisku 3. tlačítka. (viz obrázek)



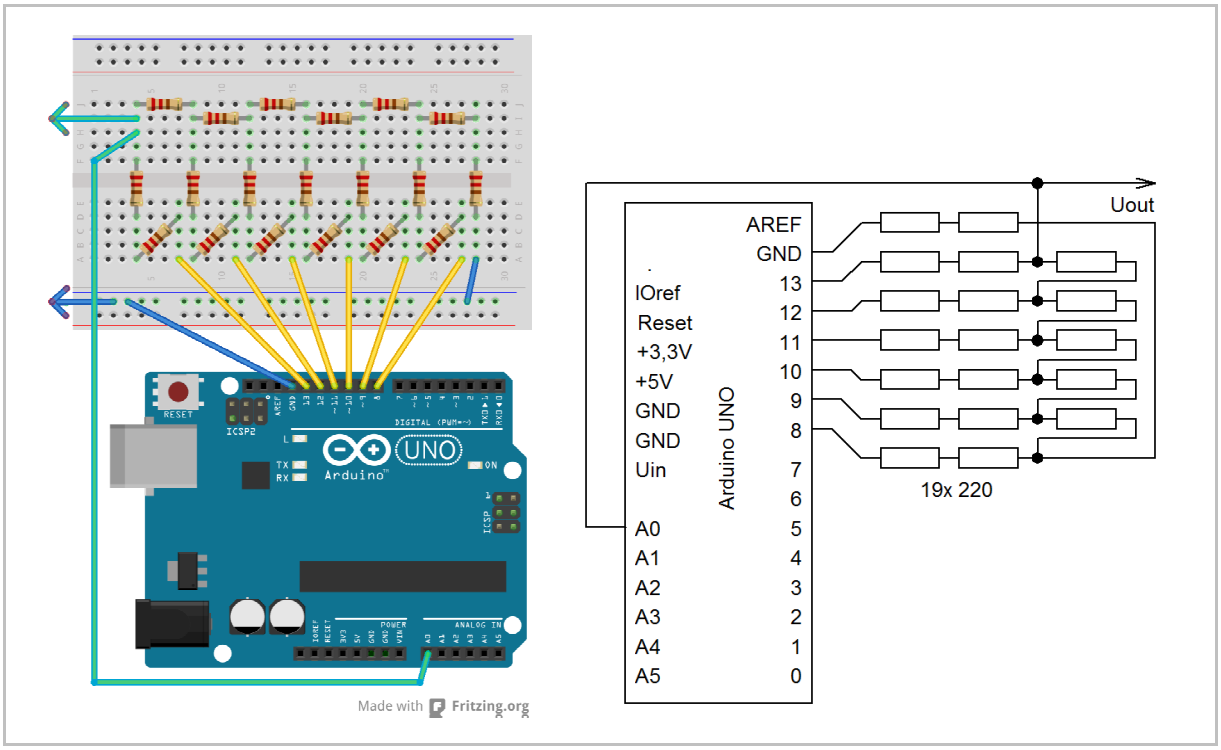
- Doplňte do zapojení další LED, na výstupech 2 až 9 budou červené, na 10 až 13 zelené. Sestavte vlastní program, který využije analogové snímání tří tlačítek i dva připravené spínače. Spínače určují trojici LED, s níž se bude pracovat (oba vypnuté – první tři LED, na A0 zapnutý a A1 vypnutý – druhá trojice LED, na A0 vypnutý a A1 zapnutý – třetí trojice LED, na A0 i A1 zapnutý – čtvrtá trojice LED). Nejprve se navolí trojice LED pomocí spínačů a pak stiskne příslušné tlačítko určující LED ve vybrané trojici, ta změní svůj stav (rozsvítí se, byla-li zhasnuta, znasne, byla-li rozsvícena). Tímto způsobem zkuste střídavě ovládat libovolnou LED.

- Využijte úpravu zapojení z předchozího námětu, program bude jiný. Budeme ovládat jen 10 LED rozdělených na dvě pětice. Spínače určují vybranou pětici, tlačítka a jejich kombinace stejně jako v předminulém námětu výběr LED z pětice. Aktivací příslušné LED se změní její stav. Problém je v tom, že při stisku dvou tlačítek nikdy nestiskneme obě naprosto přesně současně, takže je potřeba měřit délku sepnutí každého tlačítka a teprve pokud trvá určitou dobu, třeba 0,3 s, brát to opravdu jako platný stisk a kontrolovat, zda bylo současně stisknuto dostatečně dlouho jiné tlačítko. Snažte se ovládat střídavě různé LED. Pětice LED je vybrána jediným spínačem na A0, je-li sepnut, týká se činnost první pětice, je-li vypnut, druhé pětice.
- Totéž jako v předchozím námětu, ale mění se způsob výběru pětic LED. Je-li sepnut spínač na A0, je vybrána první pětice, je-li sepnut spínač na A1, je vybrána druhá pětice. Zadání povoluje, aby byly vybrány obě pětice současně a změny se tedy prováděly v obou pěticích jediným stiskem tlačítka.
- Zapojení i program zůstává stejný jako v předchozím námětu. Rozsvítíme náhodně vybraných 5 LED z 10 ovládaných, nicméně z každé pětice musí svítit nejméně jedna. Nyní s co nejmenším počtem manipulací zhasněte všechny LED. Manipulací se rozumí stisk tlačítka nebo kombinace tlačítek či přepnutí vypínače. Dokážete slovně popsat algoritmus, který vede ke zhasnutí všech LED s co nejmenším počtem manipulací?

Práce s napětím – DA převodníky

Arduino nabízí šest vstupů s přiměřeně rychlými a relativně dobrými (desetibitovými) AD převodníky i možnost nastavení jejich společného referenčního napětí, ale pro opačný převod digitálního údaje na napětí nic podobného nemá. Existuje více možností, jak absenci DA převodníku obejít, dvě z nich si nyní ukážeme.

Paralelní DA převodník s váhovými rezistory R-2R



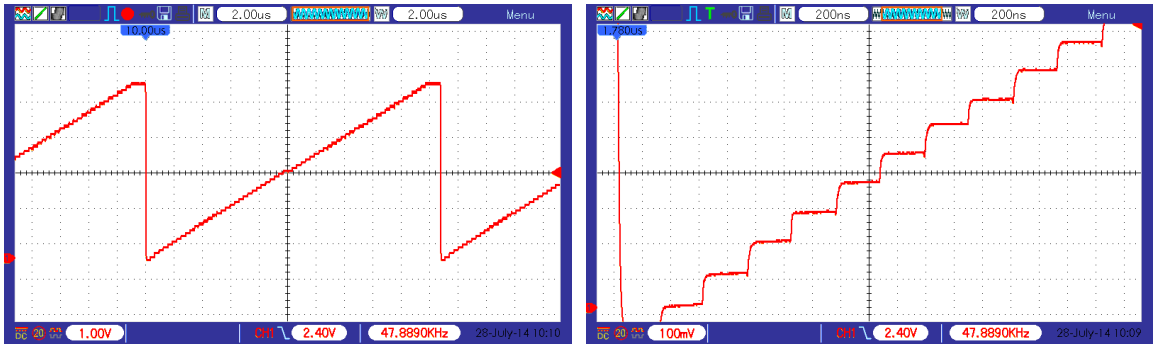
Na ukázku si s rezistory 220 Ω, kterých máme k dispozici větší množství, postavíme šestibitový převodník se sítí R-2R. Použijeme digitální piny 8 až 13 a vytvoříme zapojení podle předchozího obrázku. Pin 8 je v šestibitovém údaji nejméně významný, pin 13 nejvíce, rozlišení je na 64 hodnot, nejnižší odpovídá napětí 0 V, nejvyšší odpovídá napájecímu napětí +5 V. Vzhledem k tomu, že výstupy mikrokontroléru nejsou ideální spínače bez úbytku napětí, má podobný převodník určité chyby, ale funguje poměrně dobře.

Ověřit činnost DA převodníku můžeme více způsoby. Tím nejjednodušším je připojit na jeho výstup voltmetr (s velkým vstupním odporem), poslat na výstupy 8 – 13 nějaké číslo. Pak porovnat změřené napětí s teoretickým. Jednomu bitu by mělo odpovídat napětí 5/63 = 79,365 mV. Když nemáme po ruce voltmetr, můžeme si pomoci AD převodníkem Arduina. Při referenčním napětí +5 V by mělo jednomu bitu poslanému DA převodníku odpovídat číslo 16 z desetibitového AD převodníku.

Takto postavený DA převodník je jednoduchý a levný, ale není malý, má hodně součástek a zabírá hodně vývodů Arduina. Pokud bychom posílali digitální údaj na výstup bit po bitu, výstupní hodnota by „poskakovala“ a bylo by to pomalé; toto je přesně ten okamžik, kdy je třeba na úkor čitelnosti a přehlednosti programu dát přednost rychlosti (synchronizaci) a využít práce s registry. To je také největší výhodou podobného typu převodníku, je rychlý.

Připravíme si program, který bude cyklicky posílat bez prodlev do převodníku čísla 0 až 63 a podíváme se na výsledné napětí, které by mělo mít průběh vzestupné pily. K předání čísla na výstupy je použit registr PORTB, jinak je program krajně jednoduchý.

```
// DAC1 6-BIT DA PREVODNIK SE SITI R-2R

void setup(){                              // nastaveni
  DDRB = B00111111;                        // piny 8 - 13 vystup
}

void loop(){                               // program
  for (int i = 0; i<64;i++){PORTB = i;}    // poslat 0 - 63
}                                          // konec programu
```



K zobrazení činnosti programu a průběhu napětí na výstupu nejlépe poslouží osciloskop. Výsledek ukazují obrázky. Na prvním obrázku je celý průběh, vzestupná pila s periodou téměř 48 kHz, což je velmi dobrý výsledek. Už na tomto obrázku je vidět, jak je výstupní napětí schodovité a kdybychom přepočítali úrovně, bylo by jich 64. „Uříznutá“ špička respektive prodleva při dosažení maximální úrovně je způsobená časem, který potřebuje cyklicky se opakující void loop k novému „nastartování“. Detail už jen názorně ukazuje schodovitost napětí. Takto rychlý převodník by byl schopen přenést s dostatečnou mírou srozumitelnosti třeba řeč, třeba slovo „nahrané“ do digitálního záznamu a uložené do paměti.

I když náš program nikde nečeká, není nejrychlejší možnou obsluhou převodníku. Pokud by byla čísla odesílána řadou za sebou následujících příkazů PORTB = xxx, byl by převod rychlejší, ovšem za cenu velmi dlouhého a nehospodárného prohramu.

Když nemáme k dispozici osciloskop, vložíme do cyklu čekání 2 s a na voltmetru pozorujeme, jak napětí schod za schodem stoupá. Když není k dispozici ani voltmetr, pomůžeme si následujícím programem, který z Arduina udělá jak generátor pilovitého průběhu s DA převodníkem, tak současně voltmetr (vlastně jednoduchý osciloskop s časem na svislé ose) s grafickým zobrazením vstupního napětí pomocí znaků na sériovém monitoru v PC. Už není nutné (ani možné) spěchat, takže tentokrát už nebudeme používat registry a vše obsloužíme standardním (pomalým) způsobem.

```
// DAC2 6-BIT DA PREVODNIK SE SITI R-2R A ZOBRAZENIM V PC

void setup(){                          // nastaveni
  analogReference(DEFAULT);            // reference napajeci napeti
  for (int i = 8; i < 14; i++){        // piny 8-13 na vystup
    pinMode(i,OUTPUT);}                //
  Serial.begin(9600);                  // seriova linka 9600Bd
}
void DAC (int cislo){                  // obsluha DA prevodniku
  for (int i = 8; i < 14; i++){        // pro piny 8-13
    if (cislo % 2 != 0){               // cislo je liche?
      digitalWrite(i,HIGH);}           //   ano - bit HIGH
      else {digitalWrite(i,LOW);       //   ne  - bit LOW
    }
    cislo = cislo/2;                   // deleni 2 (bit posun vlevo)
  }
}
void loop(){                           // program
  for (int i = 0; i<64;i++){           // cyklus pro 64 hodnot
    DAC(i);                            // generuj napeti
    delay(500);                        // pockej 0,5s
    for (int j = 0; j < analogRead(0)/8;j++){ // vypis rady x dle napeti
      Serial.print(„x“);               //
    }
    Serial.println();                  // dalsi radek
  }                                    // konec cyklu
}                                      // konec programu
```

Náměty:

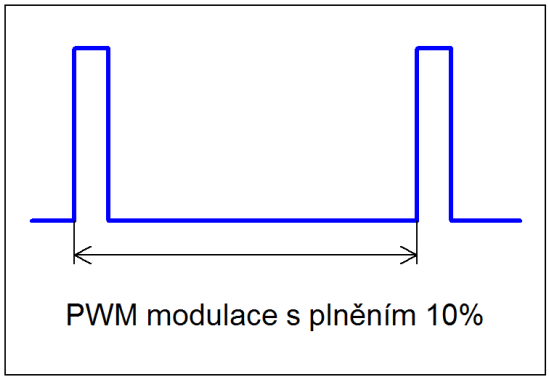
- Upravte poslední program tak, aby generoval trojúhelníkovitý signál (plynulý růst i pokles napětí). Zkontrolujte průběh napětí podle zobrazení v sériovém monitoru PC.
- Upravte program tak, aby generoval sinusový signál v rozsahu 0 až 5V. Funkci sin() Arduino zná, parametrem je údaj (úhel) v radiánech typu float, vrací hodnotu typu double v rozsahu -1 až +1. Zobrazte nasnímaný průběh napětí v sériovém monitoru PC.
- Doplňte do zapojení generátor kmitů s obvodem 555 a kondenzátorem 470 µF, který jsme už používali. Trimrem nastavte nejpomalejší kmity. K analogovému vstupu A5 připojte vývod 2 obvodu 555. Napětí nasnímané z generátoru přes vstup A5 převeďte na napětí vytvořené DA převodníkem, to následně kontrolujte voltmetrem nebo zaveďte zpět na vstup A0 a měřte Arduinem. Využijte funkci map a počítejte s tím, že napětí na vstupu A5 se mění od 1/3 do 2/3 napájecího napětí, takže přibližně mezi 1,7 a 3,3 V, napětí na výstupu DA převodníku se musí měnit s plným využitím rozsahu.

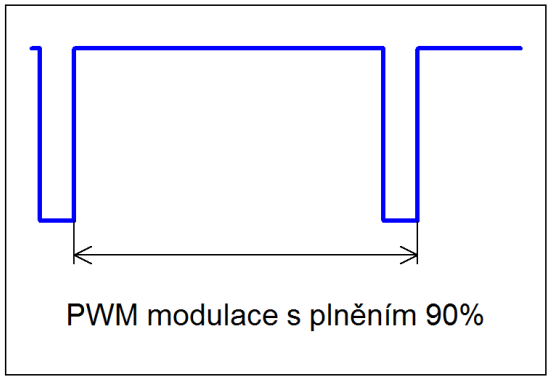
Převod DA pomocí PWM modulace

Co je PWM – analogWrite()

V úplně první úloze jsme rozblikali LED, stačil na to (pochopitelně) jeden výstupní pin. Pokud budeme blikání postupně zrychlovat od určité meze dál už naše oko neuvidí blikání, ale jen svit s nižším jasem. Oko nestihne jednotlivá zapnutí a vypnutí za normální situace vyhodnotit, ale když třeba podobně blikající světlo kolem nás rychle projede ve tmě, trvá to jen mžik, ale nezanechá v nás dojem svítící čáry, ale přerušovaně svítící čáry.

Když necháme LED rychle blikat a budeme plynule měnit poměr rozsvícení a zhasnutí, bude se nám zdát, že se plynule mění jas LED. Na stejném principu pracuje PWM (Pulse-Width Modulation), česky pulzně šířková modulace. Používá se nejen k regulaci svitu LED, ale také třeba k řízení výkonu stejnosměrných motorů, a může posloužit i k velmi jednoduchému převodu čísla na úroveň napětí. Poměru aktivní části periody (stavu H) a celé periody se také říká činitel plnění nebo zkráceně plnění. Pokud bude mít signál činitel plnění 25%, znamená to, že je v H čtvrtinu periody a v L tři čtvrtiny periody. Signál s činitelem plnění 0% znamená, že je trvale v L a už nekmitá, signál s činitelem plnění 100% znamená, že je trvale v H a už nekmitá.





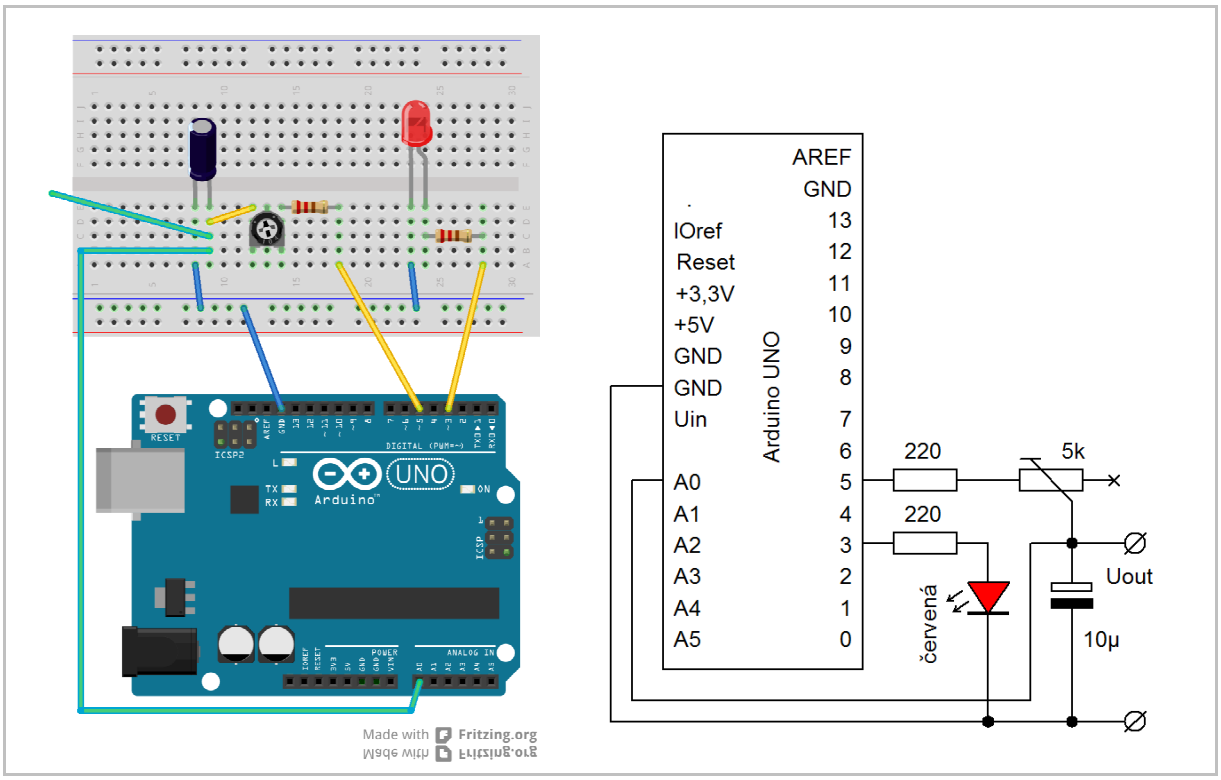
Zapínání a vypínání výstupu kvůli regulaci PWM signálem bychom mohli dělat programově, ale pro více výstupů současně by to bylo dost složité a zaměstnávalo by to mikrokontrolér. Příkaz analogWrite to dělá za nás. Má dva parametry, prvním je pin, na kterém má PWM modulaci vytvořit, ale nemůže to být každý pin, jen piny 3, 5, 6, 9, 10 a 11. Celkem tedy můžeme obsloužit šest PWM modulací současně. Druhým parametrem je činitel plnění. Tentokrát jej nezadáváme v procentech (0 až 100), ale jako hodnotu bytu v rozsahu 0 až 255. Perioda PWM signálu je pevně určená a nejde měnit, kolem 2 ms, přesněji frekvence je 490 Hz. Jakmile jednou zadáme příkazem analogWrite hodnoty, už se nemusíme dál o signál starat a další chod programu to nezatěžuje.

Příklad:

```
analogWrite(3,128);     // spustí PWM signál se střídou 128/256 = 50% na pinu 3
analogWrite(6,0);       // vypne PWM signál na pinu 6 (trvale L)
analogWrite(10,255);    // vypne PWM signál na pinu 10 (trvale H)
```

Převod DA pomocí PWM

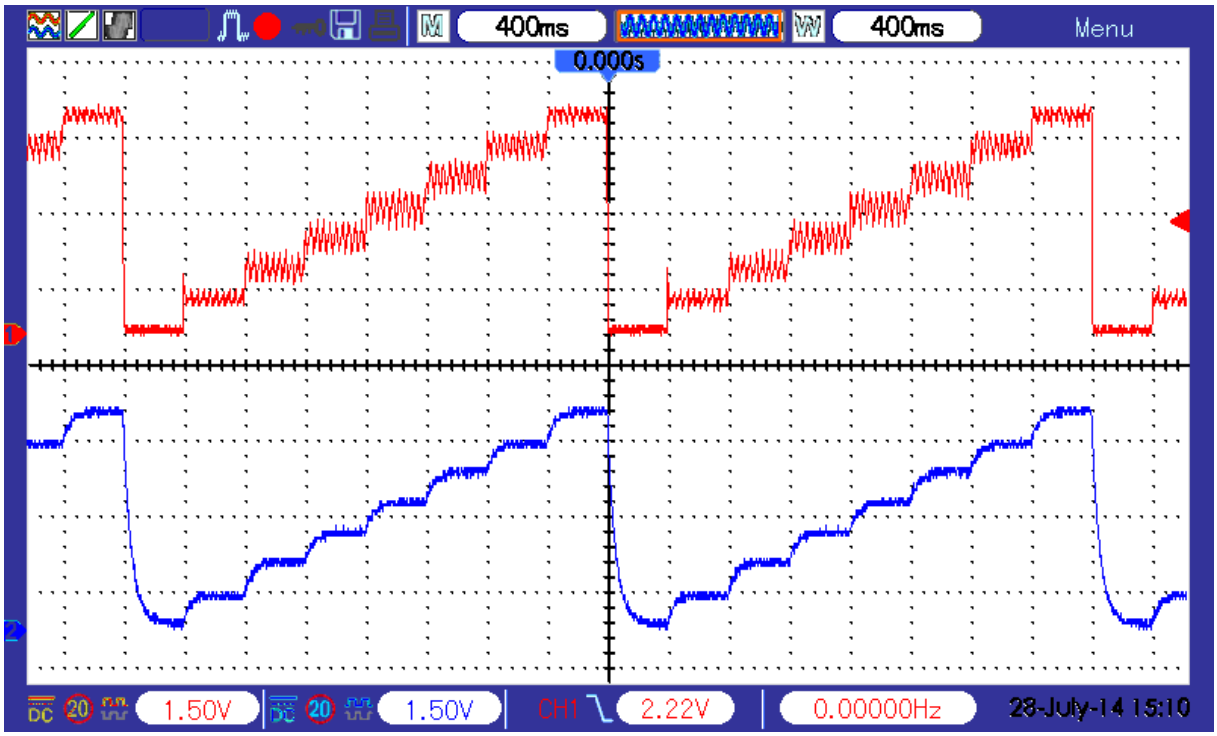
Nejprve si vyzkoušíme řídit pomocí PWM jas jedné LED a hned potom se podíváme na vytvoření převodníku na napětí v rozsahu 0 až 5 V. Obsluha bude naprosto stejná, necháme LED pomalu rozsvítit (napětí plynule růst) v osmi krocích a pak rychle zhasnout (napětí skokem klesnout). Opět si průběh napětí zkontrolujeme osciloskopem, ale tentokrát už můžeme Arduinem ověřit jen jeho (střední) hodnotu, drobné odchylky nezjistíme.




```
// DAC3 DA PREVOD A RIZENI JASU LED S PWM

void setup(){                       // nastaveni
  pinMode(3,OUTPUT);                // pin 3 na vystup PWM LED
  pinMode(5,OUTPUT);                // pin 5 na vystup PWM DA
}

void loop(){                        // program
  for (int i = 0; i<256;i=i+32){    // cyklus pro 8 hodnot
    analogWrite(3,i);               // zadat PWM pro LED
    analogWrite(5,i);               // zadat PWM pro DA
    delay(400);                     // pockat 0,4s
  }
}                                   // konec programu
```

LED opravdu po krocích zvětšuje jas, perioda trvá přibližně 3,2 sekundy (8x0,4 s). S generováním napětí je to složitější. Abychom získali vyhlazené napětí, musíme na PWM výstup připojit RC článek. V našem případě je odpor složený jednak z ochranného rezistoru 220 Ω, jednak z trimru 5 k, kondenzátor má kapacitu 10 µF. Čím menší bude nastavený odpor (případně kapacita kondenzátoru), tím hůře se vyhladí jednotlivé pulzy PWM modulace, ale při velké změně napětí se tato změna bude sledovat rychleji (lépe), čím větší je nastavený odpor (případně kapacita kondenzátoru), tím lépe budou vyhlazeny pulzy PWM modulace, ale na skokové změny bude převodník reagovat pomaleji (hůře). Výsledný průběh napětí je na obrázku. Horní červená stopa s výraznými zbytky PWM je pro trimrem nastavený nejmenší odpor, spodní modrá stopa je pro největší nastavený odpor. PWM je sice téměř zmizela, ale na poklesu napětí na konci periody je vidět, jak převodník reaguje pomalu.

Převod pomocí PWM se hodí v případě, že není potřeba rychle reagovat na změny, napětí se mění velmi pomalu vzhledem k použitému kmitočtu PWM, v našem případě je k ustálení potřeba přibližně 0,5 s. Výhodou je, že stačí jediný pin a dvě další součástky (není-li odpor nastavitelný), zapojení je velmi jednoduché a při dostatečně dlouhé době na ustálení má i osmibitové rozlišení stejně jako povel analogWrite.



Jak zkontrolovat činnost převodu pomocí PWM modulace bez osciloskopu? Opět použijeme vstup s AD převodníkem Arduina. Na konci každého stupně změříme napětí a délkou řádky znaků znázorníme jeho hodnotu. LED už obsluhovat nebudeme. Při nastaveném minimálním odporu trimru budou rozdíly mezi naměřenými hodnotami (délkou řádků znaků) jasně vidět, protože výstup je „zarušený“ zbytky PWM a AD převodník snímá napětí velmi rychle v náhodném bodě rušení. Při maximálním odporu budou rozdíly jen malé, signál je dobře vyhlazený a je téměř jedno, kdy AD převodník odebere měřený vzorek.

```
// DAC4 DA PREVOD A RIZENI JASU LED S PWM – ZOBRAZENI

void setup(){                                    // nastaveni
  pinMode(5,OUTPUT);                             // pin 5 na vystup PWM DA
  Serial.begin(9600);                            // seriova linka 9600Bd
}

void loop(){                                     // program
  for (int i = 0; i<256;i=i+32){                 // cyklus pro 8 hodnot
    analogWrite(5,i);                            // zadat PWM pro DA
    delay(400);                                  // pockat na ustaleni 0,4s
    for (int j = 0; j < analogRead(0)/8;j++){    // vypis rady x
      Serial.print(„x“);                         //
    }
    Serial.println();                            // dalsi radek
  }
}                                                // konec programu
```

Náměty:

- V zapojení použijte místo kondenzátoru 10 µF kondenzátor 470 µF. Program upravte tak, aby napětí na výstupu odpovídalo 10% a 90% maxima (přibližně 0,5 V a 4,5 V) a tyto dvě hodnoty se střídaly po 4 s. Ověřte si měřením napětí AD převodníkem a číselným výpisem hodnot po 0,2 s, že při skokových změnách nastavení nereaguje výstup napětí hned, ale ustaluje se určitou dobu.
- Program z ukázkového příkladu upravte tak, aby bylo výstupní napětí sinusové s periodou kolem půl minuty (bude sa pracovat ne s 8 kroky na periodu, ale se 64 kroky). Zkontrolujte průběh pomocí výpisu do sériového terminálu v PC.
- Sestavte vlastní zapojení i program pro plynulé rozsvícení i zhasínání LED. Po stisku tlačítka se LED plynule rozsvítí během 2 s a zůstane svítit plným jasem, po dalším stisku tlačítka plynule zhasne během 2 s a zůstane zhasnutá.
- Sestavte vlastní zapojení i program pro plynulé rozsvícení a zhasnutí LED pomocí dvou tlačítek. Přidržením jednoho tlačítka se bude LED pomalu rozsvěcet v 16 krocích (plného jasu z vypnutí dosáhne za 4 s), po puštění tlačítka zůstane svítit nastaveným jasem. Přidržením druhého tlačítka bude LED pomalu zhasínat v 16 krocích (vypnutí z plného jasu dosáhne za 4 s), po puštění tlačítka zůstane LED svítit nastaveným jasem. Stisknutím obou tlačítek současně LED okamžitě skokem zhasne. Nepoužívejte vstupy A0 až A5.
- Totéž jako v předchozí úloze, ale tlačítka budou zapojena na analogový vstup A5.
- Sestavte vlastní zapojení a program, v němž se bude snímat napětí z trimru a podle něj řídit jas LED od minima (zhasnutí) do maxima.
- Doplňte do původního zapojení generátor kmitů s obvodem 555 a kondenzátorem 470 µF. K analogovému vstupu A5 připojte vývod 2 obvodu 555. Trimrem nastavte nejpomalejší kmity. Pomocí PWM řiďte jas LED na pinu 3 podle toho, jak se mění napětí z generátoru na vstupu A5. Vzhledem k tomu, že napětí z generátou nedosahuje pod 1,7 V ani nad 3,3 V, LED se nikdy plně nerozsvítí ani nezhasne, ale změny jasu by měly být jasně vidět.
- Totéž jako v předchozím námětu, ale upravte rozsah regulace funkcí map tak, aby LED využila celého rozsahu svitu od nuly do maxima.
- Jak se musí upravit fukce map, aby přibližně jednu čtvrtinu periody byla LED zhasnutá, jednu čtvrtinu se plynule rozsvěcela, další čtvrtinu plně svítila a poslední čtvrtinu plynule zhasínala? Vyzkoušejte.

Řídíme stejnosměrný motor

Pro řízení stejnosměrného motoru nebudeme potřebovat žádné nové příkazy, poprvé však budeme muset proudově posílit výstupy Arduina. Digitální výstupy je možné zatěžovat proudem typicky 20 mA, maximálně 40 mA. Motorek s převodovkou, který je přiložen v sadě, odebírá bez zatížení kolem 60 mA, při zatížení přes 300 mA. Převodovka a malý kotouček se značkou na jejím výstupu dovolují snadno pozorovat dostatečně nízké otáčky a účinek jejich regulace.

Připravíme si zapojení podle obrázku.

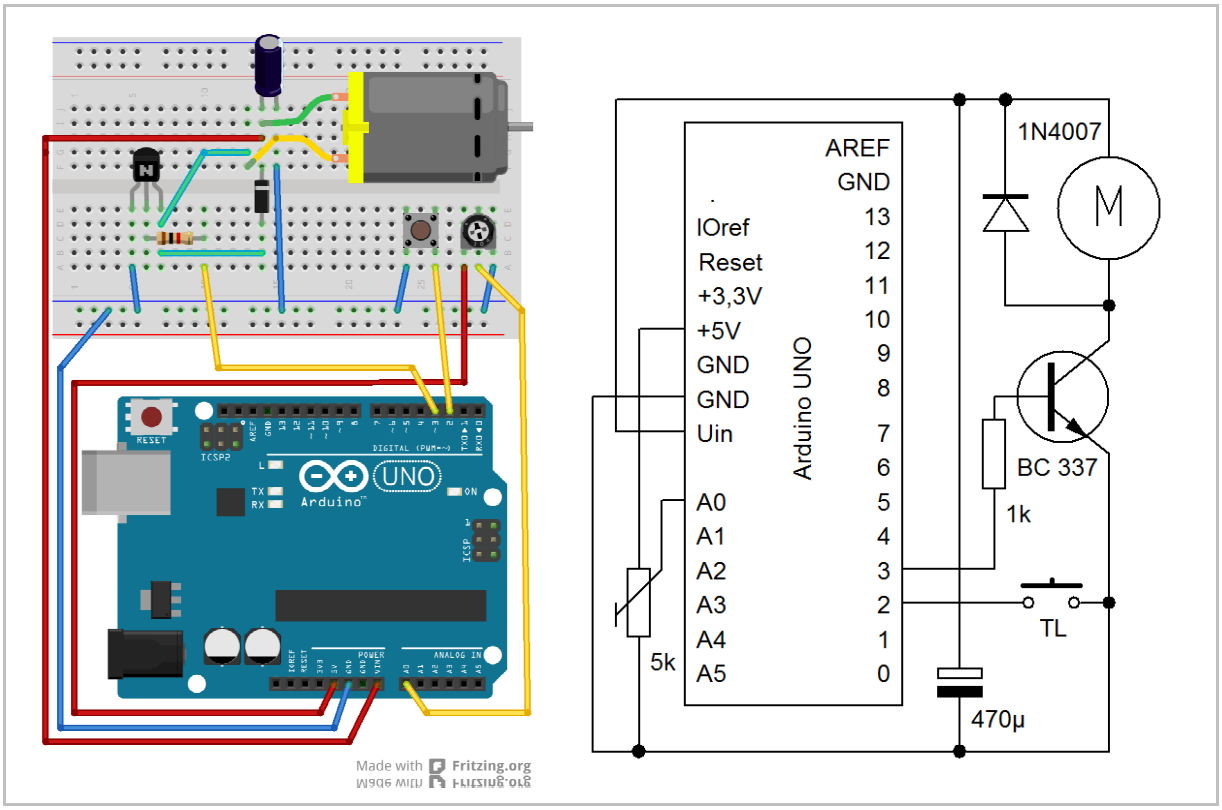
POZOR! Tentokrát je třeba napájet motor z vývodu Uin, ne ze stabilizátoru Arduina a jeho vývodu +5 V! Současně je nutné při pokusech s motorem napájet celé Arduino výhradně USB kabelem z počítače, ne přes souosý konektor pro napětí 7 – 12 V!

Motor vyžaduje napětí do 5 V, ale jeho odběr je příliš velký, tímto způsobem jej napájíme z dostatečně silného zdroje (USB z počítače) a stabilizátor není zatížen.

Přes motor musí být zapojena ochranná dioda, bez ní by mohlo dojít k poškození PC špičkami napětí při brzdění motoru!

Věnujeme pozornost správnému zapojení tranzistoru. Je-li ploška na pouzdru směrem k Arduinu tak jako na nákresu, je vlevo emitor, ten se musí spojit se zemí, prostřední báze musí být spojena přes odpor 1 k s řídícím pinem (3) a k pravému vývodu, kolektoru, připojíme motor. Elektrolytický kondenzátor 470 µF přes napájení motoru v jeho blízkosti je nezbytný, jinak by se mohlo rušení od kartáčků motoru přenést jak do Arduina, tak do PC.

Tranzistor snese trvalou zátěž 800 mA, takže bez potíží zvládne spínat přiložený motor, jiný silnější motor však k němu nelze připojit!



Spínání motoru

Jako první si vyzkoušíme jednoduchý program, který při stisku tlačítka roztočí motor.

```
// MOTOR1 - SPINANI MOTORU TLACITKEM

void setup(){                 // nastaveni
  pinMode(3,OUTPUT);          // pin 3 vystup motor
  pinMode(2,INPUT_PULLUP);    // pin 2 vstup TL
}

void loop(){                  // program
  if (digitalRead(2)==LOW){   // je TL stisknuto?
    digitalWrite(3,HIGH);}    //   ano - motor zap
    else {digitalWrite(3,LOW);//   ne  - motor vyp
  }                           // konec else
}                             // konec programu
```

Řízení otáček motoru pomocí PWM

V další úloze využijeme toho, že řízení motoru je „náhodou“ na pinu 3, který může být také výstupem PWM modulace. Motor budeme ovládat trimrem od zastavení do plného výkonu.

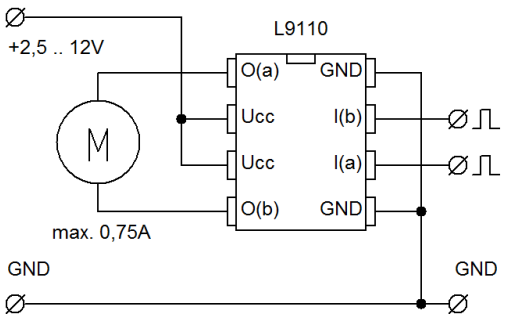
```
// MOTOR2 - RIZENI OTACEK MOTORU POMOCI PWM

void setup(){       // nastaveni
pinMode(3,OUTPUT);  // pin 3 vystup motor
}

void loop(){
analogWrite(3,map(analogRead(0),0,1023,0,255));
}                   // konec programu
```

Obousměrné řízení stejnosměrného motoru

Zatím jsme se spokojili s tím, že motor se točil jen jedním směrem daným tím, jak byl zapojený. Dál budeme chtít, aby se motor podle nastavení trimru točil oběma směry, uprostřed dráhy bude v klidu, na krajích pojede maximální rychlostí. K regulaci rychlosti využijeme opět samozřejmě PWM, dokonce na dvou pinech. Aby se dal jednoduše motor ovládat i včetně přepólovaní (změny směru), využijeme budič motoru – můstek L9110, který je přímo k tomuto účelu navržen.

Driver stejnosměrného motoru L9110

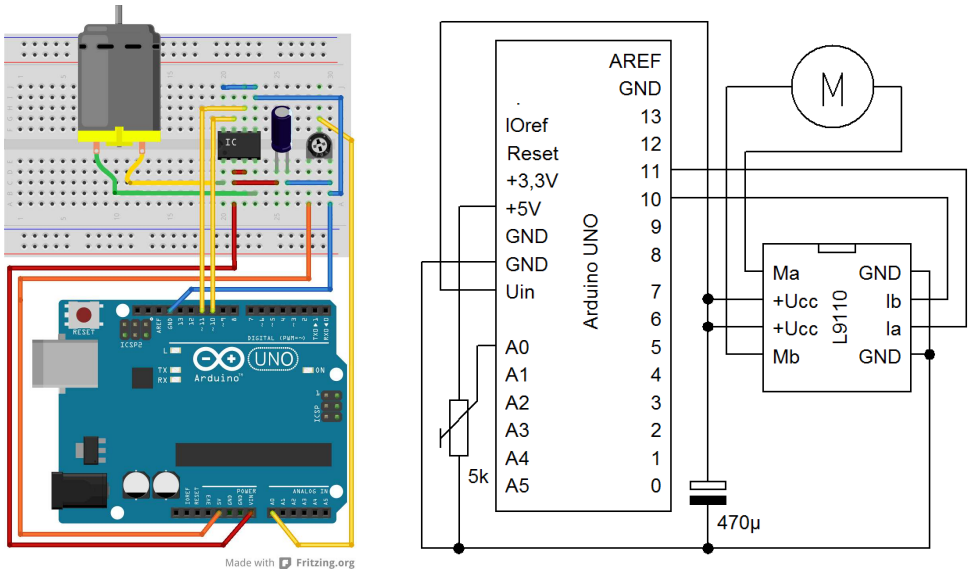
Napájecí napětí můstku a tedy i motoru se může pohybovat od 2,5 do 12 V. Obvod má dva vstupy odděleně pro chod motoru vpřed a vzad. Motor se zapojuje přímo mezi výstupy obvodu, ochranné diody proti špičkám vznikajícím při přepínání komutátoru jsou součástí obvodu.



Nezapojené vstupy drží spolehlivě úroveň L, v úrovni H teče vstupem proud kolem 1 mA. Úroveň napětí pro L je nejvýše 0,7 V, pro úroveň H typicky kolem poloviny napájecího napětí. Proud motorem smí být trvale 0,75 A, ve špičce 1,5 A.

Je-li právě jeden vstup v úrovni H, je příslušný výstup v úrovni H a druhý v úrovni L, motor se točí jedním nebo druhým směrem. Pokud jsou oba vstupy v H nebo v L, jsou oba výstupy v podstatě ve třetím stavu, motor je odpojen.

Řadič v typickém zapojení nevyžaduje žádné další součástky. Vstupy mohou být připojeny přímo k Arduinu. Budeme muset vytvořit nové zapojení, opět nezapomeňte na kondenzátor 470 µF.



Aby šlo spolehlivě motor ve střední poloze trimru zastavit, necháme tam pásmo určité šířky (500 až 524), v němž bude motor zcela bez napájení.

```
// MOTOR3 - OBOUSMERNE RIZENI OTACEK MOTORU POMOCI PWM

int rizeni;                                        // napeti z trimru
void setup(){                                      // nastaveni
  pinMode(10,OUTPUT);                              // pin 10 PWM motor
  pinMode(11,OUTPUT);                              // pin 11 PWM motor
}

void loop(){                                       // program
  rizeni = analogRead(0);                          // nacteni napeti
  if (abs(rizeni - 512) <= 12){                    // 500 az 524
    analogWrite(10,0);                             // vypnout motor
    analogWrite(11,0);}                            //
  if (rizeni < 500){                               // 499 až 0
    analogWrite(10,map(rizeni,0,499,255,0));       //   regulace smer 1
    analogWrite(11,0);}                            //   vypnout druhy
  if (rizeni > 524){                               // 525 az 1023
    analogWrite(10,0);                             //   vypnout prvni
    analogWrite(11,map(rizeni,525,1023,0,255));}//   regulace smer 2
}                                                  // konec programu
```

Náměty:

- V daném zapojení upravte program tak, aby se otáčky motoru regulovaly trimrem (stačí v jednom libovolném směru), ale aby byly extrémně pomalé. Rozsah regulace bude nejvýše 1 až 10 otáček za minutu.Nápověda: PWM modulací s frekvencí téměř 500 Hz generovanou Arduinem se tohoto dosáhnout nedá. Motor buď stojí a píská, nebo se rozjede vyšší rychlostí, než potřebujeme. Je třeba si napsat vlastní a podstatně pomalejší modifikaci PWM. V první řadě si vyzkoušet, jaká nejkratší délka pulzu pohne spolehlivě motorem. Pak pulzy této zjištěné délky posílat na motor s takovým odstupem, aby se pohyb jevil jako (pokud možno) plynulý, i když je vlastně trhaný, řízení motoru „krokuje“. Je slyšet, jako motor „tiká“. Trimrem se pak nastavuje délka prodlev mezi vyzkoušenými pulzy konstantní délky.
- Totéž co předchozí úloha, ale trimrem se řídí pomalý pohyb v obou směrech.
- Sestavte vlastní zapojení i program. Motor bude řízen v pěti stupních rychlosti pro otáčení vlevo a v pěti stupních rychlosti pro otáčení vpravo. K ovládání použijte tři tlačítka. Levé tlačítko bude každým stiskem přidávat rychlost o jeden stupeň vlevo (pravé bude ubírat jeden stupeň), prostřední tlačítko jediným stisknutím motor zastaví, pravé tlačítko bude přidávat rychlost vpravo (levé bude ubírat jeden stupeň). Jedno stisknutí tlačítka způsobí změnu rychlosti o jeden stupeň bez ohledu na to, jak dlouho stisk trvá (při držení tlačítka se rychlost motoru nemění).
- Pro původní zapojení upravte program tak, aby otáčky motoru sledovaly nastavení trimru (stejně, jak to bylo), ale současně aby program nedovolil rychlou změnu otáček. Zpomalení změny bude přibližně odpovídat době 5 s od zastavení do plných otáček. Toto není samoúčelné a používá se to velmi často, protože zpomalení rozběhů a doběhů výrazně přispívá k delší životnosti motorů i převodovek a také odstraňuje špičky odběru proudu při rychlých změnách otáček.

Generování zvuku

Jako první úlohu při pokusech s Arduinem jsme nechali blikat LED. Když „blikání“ dostatečně zrychlíme, dostaneme se do oblasti frekvencí zvuku a můžeme generovat tón. Stejný postup jako při blikání, kdy se mikrokontrolér mohl věnovat jen této činnosti a nic jiného už nestihl, ale rozhodně není ten nejlepší. Chtělo by to příkaz, který by dokázal generovat tón „na pozadí“ a nezdržoval další běh programu podobně, jako třeba příkaz analogWrite generuje PWM modulaci. Takový příkaz opravdu Arduino má, může však tímto způsobem generovat v jednom okamžiku jen jeden tón.

Než se pustíme dál, je potřeba se zamyslet nad účelem generování zvuku Arduinem. Rozhodně má smysl mít možnost vydat nějaký nápadný varovný signál, potvrdit kliknutím stisk tlačítka, zahrát několik tónů melodie jako oznámení, že požadovaná činnost skončila. Arduino samo o sobě není určeno a nemá předpoklady k tomu generovat (syntetizovat) hudbu. Samozřejmě, třeba se specializovaným shieldem s midi generátorem jako „řídící počítač“ to zvládnout může, a dokonce velmi dobře.

Příkazy tone() a notone()

Tone má tři parametry. První určuje pin, na němž má být zvuk generován, druhý zadává frekvenci tónu v Hz (typ unsigned int). Třetí parametr je nepovinný a zadává délku trvání tónu v milisekundách (typ unsigned long). Použití tone znemožňuje současné generování PWM modulace na pinech 3 a 11. Pokud není třetí parametr zadán, je tón generován trvale, ukončí se příkazem notone(), jehož jediným parametrem je pin, kde má tón vypnout.

Pokud jde o různé zvukové efekty, lze pracovat přímo s frekvencemi v Hz, při zadávání melodie ale potřebujeme znát přesné frekvence jednotlivých tónů.

Může posloužit následující tabulka:

```
#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
```

```
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
```

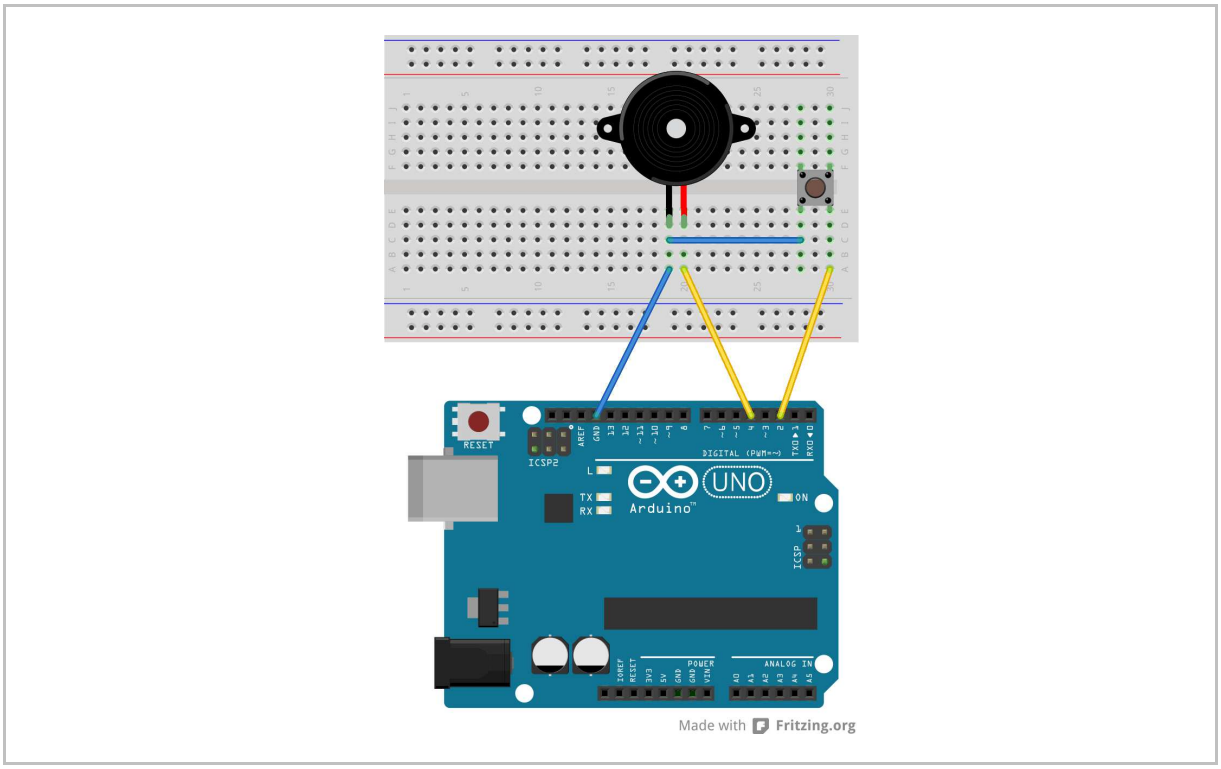
```
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
```

#define NOTE_B2 123 #define NOTE_C3 131 #define NOTE_CS3 139 #define NOTE_D3 147 #define NOTE_DS3 156 #define NOTE_E3 165 #define NOTE_F3 175	#define NOTE_FS5 740 #define NOTE_G5 784 #define NOTE_GS5 831 #define NOTE_A5 880 #define NOTE_AS5 932 #define NOTE_B5 988 #define NOTE_C6 1047	#define NOTE_CS8 4435 #define NOTE_D8 4699 #define NOTE_DS8 4978

Potíž při generování melodií je v tom, že tón A4 má mít frekvenci přesně 440 Hz, to souhlasí. Tóny o oktávu vyšší mají vždy dvojnásobnou frekvenci, to také není problém, o oktávu nižší mají vždy poloviční frekvenci, to až do A1 (55 Hz) jde také. Jenže poměr dvou sousedních půltónů by měl být přesně dvanáctá odmocnina z čísla dvě, to je 1,059 463 094… Bohužel, příkaz tone umí generovat jen frekvence s krokem 1 Hz, takže všechny ostatní tóny jsou nepřesné, čím nižší tón, tím je většinou odchylka větší a melodie víc „tahá za uši“. Nechtějme tedy po Arduinu nějaký umělecký projev, na jednoduchých pár tónů (čím vyšších, tím lépe) ale stačí.

Zvukové efekty

Připravíme si velmi jednoduché zapojení podle obrázku, schéma snad ani není třeba uvádět. Piezoměnič ze sady můžeme připojit přímo mezi pin 4 a zem. Ale pozor, pokud by nešlo o piezoměnič, ale o klasický reproduktor, přímé připojení by přetížilo a pravděpodobně zničilo výstup Arduina. V takovém případě je nezbytné buď zařadit do obvodu kvůli ochraně odpor 110 Ω (dva rezistory 220 Ω paralelně) nebo nějaký nf zesilovač. Tlačítko na pinu 2 nám později dovolí zvuk spouštět.



Nejprve naučíme Arduino trvale přerušovaně pípat tónem 880 Hz, sekunda tónu, sekunda prodlevy.

```
// ZVUK1 - PIPANI

void setup(){                 // nastaveni
  pinMode(4,OUTPUT);          // pin 4 vystup
}

void loop(){                  // program
  tone(4,880,1000);           // ton na pin 4,A5,1s
  delay(2000);                // pockat 2s
}                             // konec programu
```

V praxi bývá výhodnější si jednoduché zvuky připravit jako podprogramy a ty volat, ukážeme si to na dalším zapojení s tlačítkem. Po stisknutí tlačítka připojeného k pinu 2 se ozve kliknutí, pak „něco“ program dvě sekundy dělá a oznámí konec trojitým klouzavým vzestupným zvukem.

```
// ZVUK2 KLIKNUTI A PAK 3X KLOUZAVY ZVUK

#define piezo 4                     // pin piezorepro
#define TL 2                        // pin tlacitka

void setup(){                       // nastaveni
  pinMode(piezo,OUTPUT);            // piezo - vystup
  pinMode(TL,INPUT_PULLUP);         // tlacitko - vstup
}

void klik(){                        // kliknuti
  tone(piezo,300,10);               // 300Hz/10ms
}

void klouz(){                       // tri klouznuti
  for (int i=1;i<4;i++) {           // cyklus pro 3
    for (int j=200;j<600;j++){      // cyklus pro tony
      tone(piezo,j);                // kazdy 4ms
      delay(4);                     //
    }
    noTone(piezo);                  // vypnout zvuk
    delay(500);                     // pauza 0,5s
  }
}

void loop(){                        // program
  if (digitalRead(2)==LOW){         // je-li stik tlacitka
    klik();                         // klikni
    delay(2000);                    // cekej 2s
    klouz();                        // zvuk 3x
  }                                 // konec if
}                                   // konec programu
```

Náměty:

- Sestavte vlastní program, který po stisknuti tlačítka zahraje trylek (velmi rychlé střídání dvou tónů) po dobu 3 s. Doba hraní je přesná (+/- tolerance trvání jednoho tónu) a nezávislá na tom, jak dlouhé tóny se střídají
- Sestavte vlastní program napodobující po dobu stisku tlačítka souvislý kolísavý zvuk policejní sirény
- Sestavte vlastní program generující zvuk původního klasického výstražného signálu hasičů (čistá kvarta s prodlevami 2 s).
- Sestavte vlastní program, který bude nepřetržitě hrát náhodné tóny (frekvence 100 až 2000 Hz) náhodné délky (délka 0,1 až 2 s). Zkuste to vydržet poslouchat alespoň minutu.
- Do zapojení přidejte trimr s výstupem napětí na pin A0. Sestavte vlastní program, který bude při stisku tlačítka generovat tón v rozsahu 50 až 1 000 Hz podle nastavení trimru.
- Vytvořte vlastní zapojení a program pro tříklávesový „hudební nástroj“. Tři tlačítka budou zapojena na piny 8, 9 a 10, výstupy tří trimrů na piny A0 až A2, výstup na piezorepro na pinu 4. Tón pro každé tlačítko se bude nastavovat samostatným trimrem.
- Totéž jako v přechozí úloze, ale navíc stisk libovolných dvou tlačítek současně vyvolá rychlé střídání dvou odpovídajících tónů.

Hrajeme melodii

Pokusíme se nechat Arduino zahrát jednoduchou melodii, motiv z asi nejznáměnší písničky pro děti. Využijeme frekvence z tabulky tónů a melodii zapíšeme do proměnné typu pole přímo v označení tónů, stačí nám jich pět. Podobně jako výšku tónu zapíšeme i jeho délku. Využijeme stejné zapojení jako měla předchozí řešená úloha s přehráváním klouzavého zvuku.

```
// ZVUK3 OVCACI, CTVERACI

#define piezo 4                                   // pin piezorepro
#define TL 2                                      // pin tlacitka
#define C4 262                                    // tony C4
#define D4 294                                    //      D4
#define E4 330                                    //      E4
#define F4 349                                    //      F4
#define G4 392                                    //      G4
#define PA 0                                      // pauza
#define tempo 400                                 // tempo ms/osminu

int noty[]={C4,E4,G4,PA,C4,E4,G4,PA,E4,E4,D4,E4,F4,
  D4,E4,E4,D4,E4,F4,D4,E4,D4,C4};                 // zapis vysky tonu
int delk[]={2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2,
  2, 1, 1, 1, 1, 2, 2, 2, 2, 2};                  // zapis delky tonu

void setup(){                                     // nastaveni
  pinMode(piezo,OUTPUT);                          // piezo - vystup
  pinMode(TL,INPUT_PULLUP);                       // tlacitko - vstup
}

void loop(){                                      // program
  if (digitalRead(TL)==LOW){                      // po stisku tlacitka
    for (int i=0;i<23;i++){                       // prehraj 23 tonu
```

```
      tone(piezo,noty[i],delk[i]*tempo);          // zvuk nastavit
      delay(delk[i]*tempo);                       // nechat znit
      noTone(piezo);                              // ukoncit
} } }                                             // konec for/if/programu
```

Náměty:

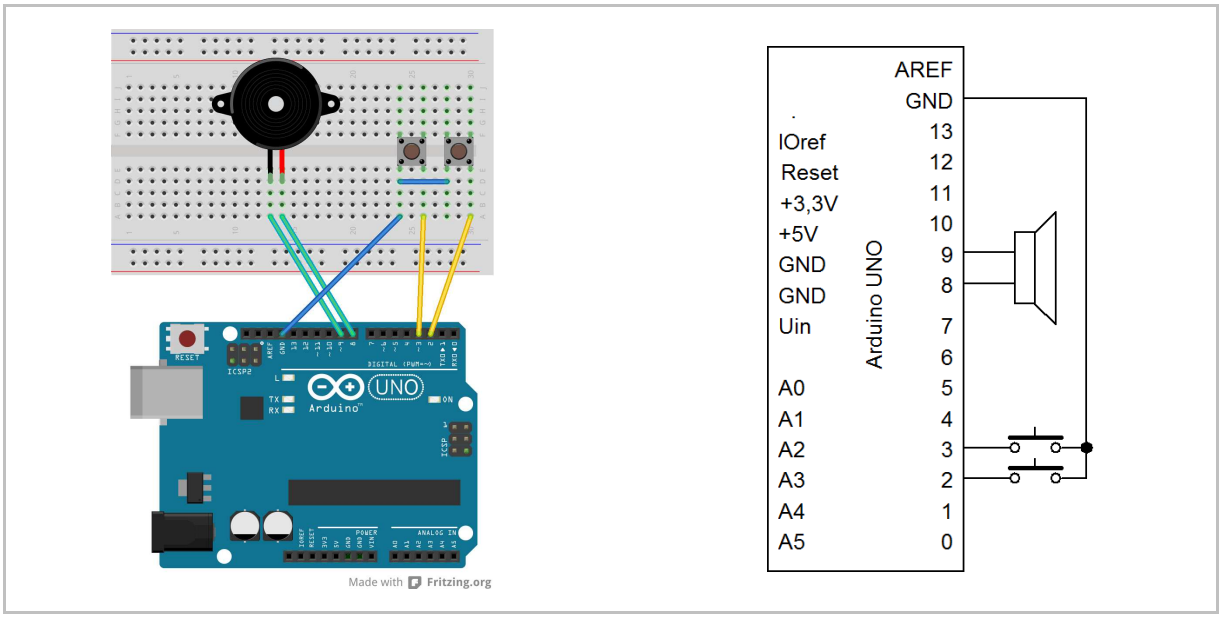
- Upravte program tak, aby se přehrávání skladby zrychlilo (zpomalilo)
- Upravte program tak, aby se celá skladba transponovala (posunula) o oktávu výš nebo níž. Nápověda: všechny frekvence tónů budou 2x vyšší nebo nižší
- Najděte jinou jednoduchou a krátkou písničku v notovém zápisu a upravte program tak, aby ji přehrál (stačí jednoduchý jednoznačně rozpoznatelný motiv melodie, nemusí být celá písnička).

Hlasitější zvuk

Piezoreproduktor je z hlediska připojení k mikrokontroléru nejjednodušším měničem zvuku, nicméně aby měl lepší hlasitost, potřeboval by pracovat s vyšším napětím než je 5 V z výstupu. Uvedeme si dvě možnosti, jak toho docílit.

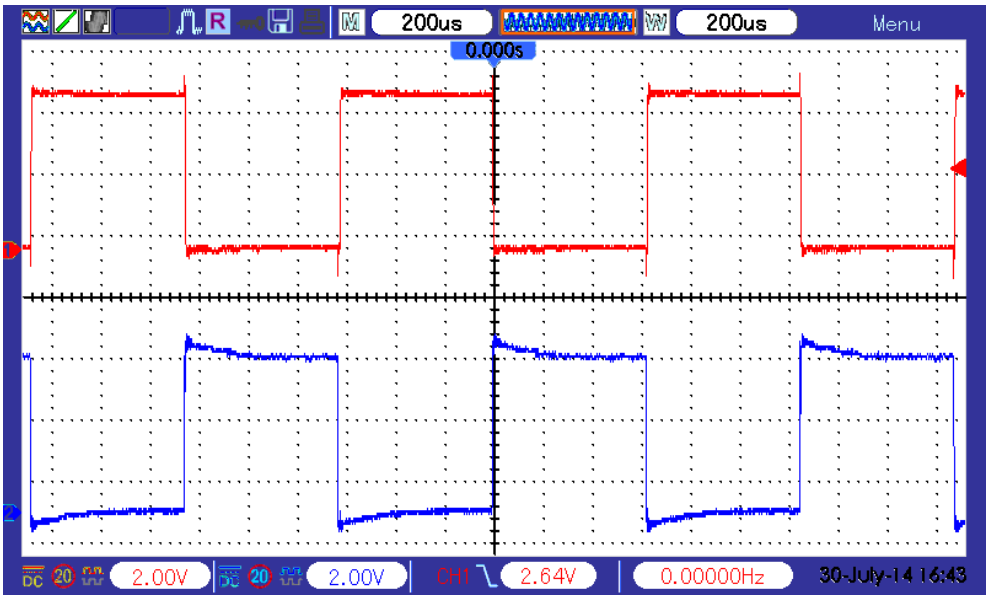
Máme-li dostatek výstupních pinů, můžeme piezoreproduktor zapojit ne mezi pin a zem, ale mezi dva výstupní piny. Na zkoušku využijeme piny 8 a 9. Když jeden z pinů necháme trvale v úrovni L a na druhém budeme generovat tón, dostaneme (téměř) stejnou situaci jako dosud, amplituda signálů je 5 V. Když ale zajistíme, aby se úroveň pinů prohazovala (je-li na jednom H, bylo na druhém L), amplituda signálu se vlastně zvýší 2x a tím podstatně vzroste i hlasitost.

Pro porovnání si upravíme zapojení na dvě tlačítka na pinech 2 a 3, abychom mohli tóny libovolně střídat, a nastavíme si v programu stejný tón, ale pokaždé generovaný jiným způsobem. Poprvé (tišeji) příkazem tone, který průběžně nezaměstnává mikrokontrolér, podruhé (hlasitěji) čistě programově s plným vytížením mikrokontroléru po celou dobu znění tónu.

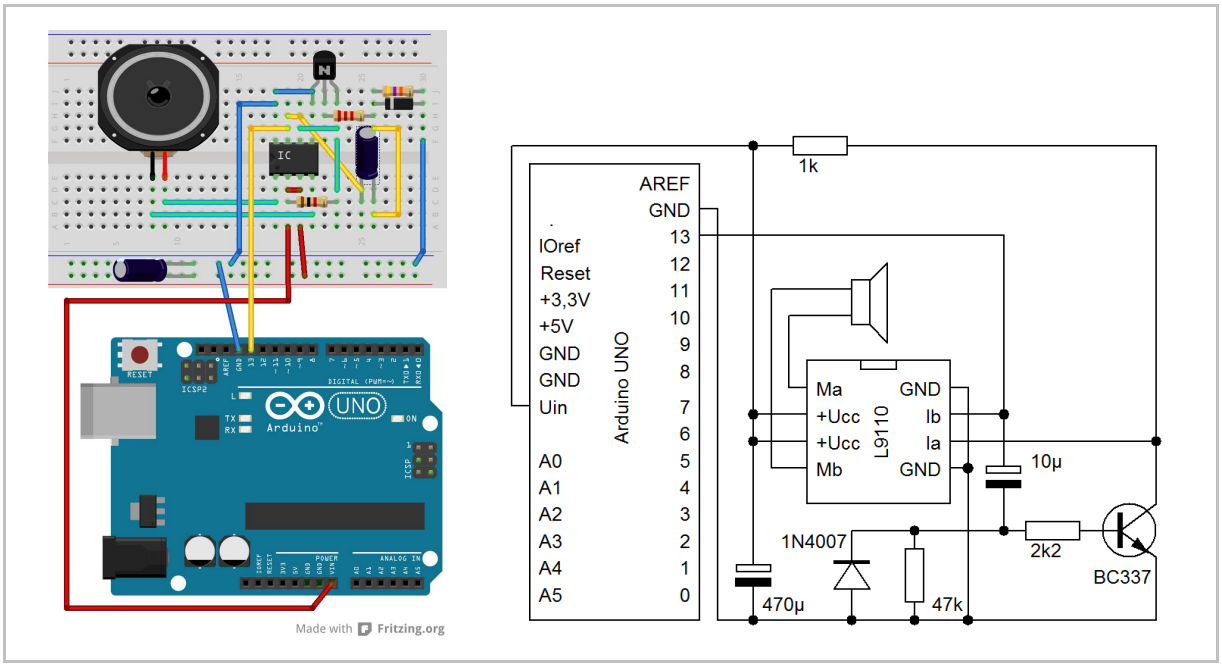


```
// ZVUK4 ZVYSENI HLASITOSTI - DVA STEJNE TONY

void setup(){                    // nastaveni
  pinMode(8,OUTPUT);             // piezo - vystup 1
  pinMode(9,OUTPUT);             // piezo - vystup 2
  pinMode(2,INPUT_PULLUP);       // tlacitko 1 - vstup
  pinMode(3,INPUT_PULLUP);       // tlacitko 2 - vstup
}

void loop(){                     // program
  if (digitalRead(2)==LOW){      // po stisku tlacitka 1
    tone(9,800,1000);            // pin 9,800Hz,1s
    digitalWrite(8,LOW);         // L na pinu 8
    delay(1000);                 // nechat doznit 1s
  }
  if (digitalRead(3)==LOW){      // po stisku tlacitka 2
    for (int i=1;i<800;i++){     // pocet period
      digitalWrite(8,HIGH);      // jeden stav
      digitalWrite(9,LOW);       //
      delayMicroseconds(614);    // pockat 0,614ms
      digitalWrite(8,LOW);       // druhy stav
      digitalWrite(9,HIGH);      //
      delayMicroseconds(614);    // pockat 0,614ms
} } }                            // konec for/tonu/programu
```

V programu je použit příkaz delayMicroseconds(). Pracuje zcela obvobně jako delay, jen doba se udává v µs a typ parametru je unsigned int a rozsah 0 až 16 383. Že to opravdu funguje se lze přesvědčit jak porovnáním hlasitosti sluchem tak technicky nasnímáním průběhů na pinech 8 a 9 – jsou v opačné fázi.



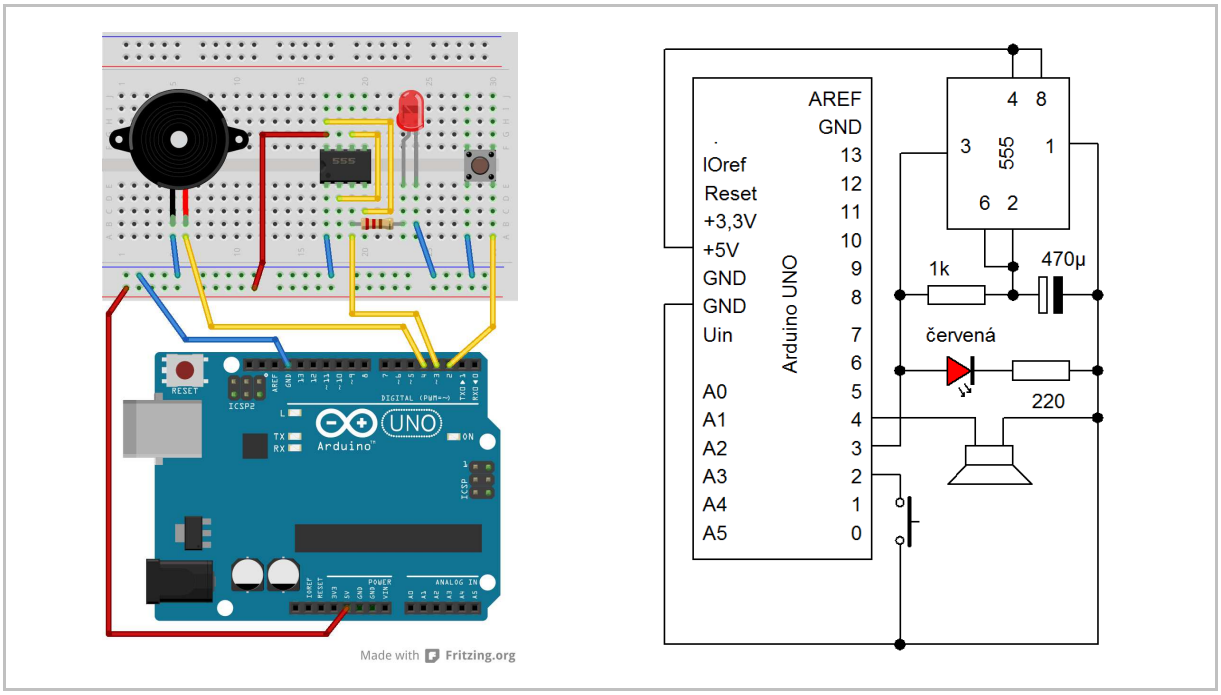
Jiný způsob zesílení netradičním způsobem využije můstku L9110, který jsme používali pro řízení stejnosměrného motoru. Problém je totiž v podstatě stejný a můstku je jedno, jestli obrací směr otáčení motoru změnou polarity nebo obracením polarity zvyšuje výkon piezoreproduktoru. Můstek má navíc tu výhodu, že výrazně posiluje i proudově a můžeme k němu připojit dokonce i standardní reproduktor s impedancí 8 Ω, v tom případě by ale bylo nutné (stejně jako u motoru) napájet můstek přímo ze zdroje (USB), ne ze stabilizátoru 5 V, který je součástí Arduina. My k vyzkoušení použije piezoměnič.



Nový program už nebudeme uvádět, vzhledem k tomu, že stačí, aby generoval tóny na jednom pinu a to libovolným způsobem, lze použít k vyzkoušení některý z předchozích a přesměrovat výstup na pin 13. Rozdíl je jen jediný, je třeba zajistit, aby v klidu byl výstupní pin zvuku v úrovni H. S připojeným piezoreprem se to víceméně jedno, ale pokud by byl připojen standardní reproduktor 8 Ω a vstup by byl v L, procházel by reproduktorem trvale poměrně velký proud.

Přerušení

Vrátíme se k zapojení podobnému tomu, které jsme používali na začátku pokusů se zvukem. Přidáme generátor s obvodem 555, později se bude hodit.



Myšlenka přerušení je poměrně jednoduchá. Když je potřeba hlídat, jestli už nastala nějaká událost, změnil se logický stav na určitém pinu, třeba bylo stisknuté tlačítko, musíme cyklicky a velmi často stav tohoto pinu testovat. To jednak zdržuje chod našeho programu, jednak stejně můžeme zareagovat se zpožděním nebo dokonce v extrémním případě celou událost „prošvihnout“, protože celý pulz (stisk tlačítka) proběhl mezi dvěma testováními.

Arduino UNO má dva použitelné interrupty čili přerušení, a to pro piny 2 a 3. Nastavíme-li například pro pin 2 interrupt tak, aby reagoval na stisk tlačítka (přechod z H do L), nemusíme už změnu testovat programem, mikrokontrolér ji svými prostředky průběžně sleduje sám a jakmile nastane očekávaná změna, okamžitě odskočí na podprogram, který jsme předem stanovili. Hned jak se podprogram vykoná (měl by být pokud možno velmi krátký), program se automaticky vrátí na místo, na němž jej vyrušil odskok do obsluhy přerušení.

Protože přerušení může narušit přesné časování chodu programu, je k dispozici i možnost pro choulostivé části programu přerušení zakázat a následně jej o kousek dál zase povolit. Přerušení může například narušit sériovou komunikaci. Současně s tím je třeba dávat pozor, jestli nějaká část našeho programu již přerušení nepoužívá, aniž by to bylo zjevně vidět. To se týká především použití knihoven, k nimž se zanedlouho dostaneme.

Po dobu vykonávání přerušení nelze používat delay a také funkce millis nebo micros budou číst stále totéž číslo, protože čítače času po tuto dobu nepracují!

Příkaz attachInterrupt()

Tento příkaz nevrací žádnou hodnotu. Jako parametry potřebuje tři údaje, první je číslo interruptu (pro Arduino UNO jen hodnota 0 znamená pin 2 nebo hodnota 1 znamená pin 3), dále pak název podprogramu, na který se při vyvolání interruptu má odskočit. Podprogram nesmí mít žádné parametry a nevrací žádné hodnoty. Posledním z údajů je mód interruptu, čili podmínka, která má být splněna pro vyvolání interruptu:

- LOW – vyvolá interrupt kdykoli je pin v úrovni L
- CHANGE – vyvolá interrupt jak při změně stavu z L na H tak z H na L
- RISING – vyvolá interrupt při změně stavu z L na H
- FALLING – vyvolá interrupt při změně stavu z H na L

Pozor! Arduino UNO nezná parametr HIGH a vyvolání kdykoli při pinu v úrovni H !

Příkaz detachInterrupt()

Nevrací žádnou hodnotu. Má jeden parametr a to číslo interruptu, pro Arduino UNO jsou možnosti jen 0 = pin 2 nebo 1 = pin 3. Odpojí přerušení, platí až do okamžiku, kdy je přerušení opět nastaveno příkazem attachInterrupt().

Příkaz noInterrupts()

Nemá parametr a nevrací žádnou hodnotu. Zakáže všechny interrupty.

Příkaz interrupts()

Nemá parametr a nevrací žádnou hodnotu. Opět povolí provedení nastavených interruptů.

Činnost přerušení si předvedeme na programu, který příkazy (pomocí delay) generuje souvislý tón, současně s tím chceme, aby kdykoli při puštění tlačítka LED změnila svůj stav. K indikaci použijeme žlutou LED označenou L na Arduinu. Po připojení napájení by měla blikat červená LED připojená na výstup 555, ale činnost tohoto obvodu zatím nic neovlivňuje. Trvale zní tón generovaný programem.

```
// INTERRUPT1 PREDVEDENI INTERRUPTU

boolean stav = false;             // promenna pro stav LED

void setup(){                     // nastaveni
  pinMode(4,OUTPUT);              // pin 4 vystup zvuku
  pinMode(13,OUTPUT);             // pin 13 vystup LED
  pinMode(2,INPUT_PULLUP);        // pin 2 vstup tlacitka
  attachInterrupt(0,blik,RISING); // nastaveni interruptu
}

void blik(){                      // obsluha interruptu0
  stav = !stav;                   // zmena logickeho stavu LED
}

void loop(){                      // program (ton cca 500 Hz)
  digitalWrite(4,HIGH);           // repro nastavit H
  delay(1);                       // pockat 1 ms
  digitalWrite(4,LOW);            // repro nastavit L
  delay(1);                       // pockat 1 ms
  if (stav){digitalWrite(13,HIGH);// dle promenne nastavit LED
```

```
    }else{digitalWrite(13,LOW);
  }
}                                 // konec programu
```

Při každém puštění tlačítka by se mělo aktivovat přerušení a přepnout žlutá LED. Na tónu není stisky tlačítka nijak poznat, obsluha přerušení je tak krátká, že se neprojeví. LED se pravděpodobně občas rozsvítí, občas zhasne, ale asi nepracuje zrovna tak, jak bychom předpokládali. Někdy třeba zdánlivě ani vůbec nezareaguje. Příčina je v tom, že obsluha přerušení je opravdu velmi rychlá a stíhá reagovat i na zákmity mechanického tlačítka, takže se může stát, že dvě změny rychle po sobě se jeví jako žádná změna. Tato zkušenost ukazuje, jak důležité je potlačení zákmitů mechanických spínačů, ať už zapojením, nebo programem.

Pokusíme se ověřit, jak bude interrupt pracovat s čistým signálem bez zákmitů, k tomu je připravený generátor s obvodem 555 a jeho výstup přivedený na pin 3, který může být napojen na interrupt 1. Tento program, který se vlastně liší jen v části nastavení, by již měl pracovat přesně podle předpokladů, při každém rozsvícení červené LED připojené na výstup 555 se změní stav žluté LED na Arduinu.

```
// INTERRUPT2 PREDVEDENI INTERRUPTU SIGNALEM 555

boolean stav = false;             // promenna pro stav LED

void setup(){                     // nastaveni
  pinMode(4,OUTPUT);              // pin 4 vystup zvuku
  pinMode(13,OUTPUT);             // pin 13 vystup LED
  pinMode(3,INPUT);               // pin 3 vstup z gen.
  attachInterrupt(1,blik,RISING); // nastaveni interruptu
}

void blik(){                       // obsluha interruptu 1
  stav = !stav;                    // zmena logickeho stavu LED
}

void loop(){                       // program (ton cca 500 Hz)
  digitalWrite(4,HIGH);            // repro nastavit H
  delay(1);                        // pockat 1 ms
  digitalWrite(4,LOW);             // repro nastavit L
  delay(1);                        // pockat 1 ms
  if (stav){digitalWrite(13,HIGH); // dle promenne nastavit LED
    }else{digitalWrite(13,LOW);
  }
}                                  // konec programu
```

Náměty:

- Upravte program tak, aby se stav žluté LED měnil při sestupné hraně pulzu z generátoru.
- Sestavte vlastní program pro připravené zapojení. Pravidelně přesně jednou za pět sekund přehraje klouzavý zvuk podobný tomu, který byl v programu „zvuk2“. Pokud je v průběhu generování zvuku stisknuto tlačítko, zvuk okamžitě ztichne a ozve se znovu až nastane doba na opětovné zaznění. Pro přesné časování zvuků využijte funkci millis, pro umlčení přerušení.
- Sestavte vlastní zapojení i program, který by ověřil, zda je možné při vykonávání obsluhy přerušení vyvolat znovu to samé přerušení. Nápověda: stačí zapojit jedno tlačítko a jednu LED. Stisk tlačítka vyvolá přerušení a obslužný program bude delší než jen mžikový, jeho úkolem bude rozsvítit LED a nechat ji asi dvě sekundy svítit. Doba svícení musí být načasována programem, ale nikoli pomocí delay, například for cyklem s dosti velkým počtem opakování. Pokud během svitu LED vyvolaného prvním přerušením stisknete znovu tlačítko, co se stane? Bude LED svítit jen původní dvě sekundy (druhý pokus o přerušení se ignoruje) nebo bude svítit dvojnásobně dlouho (i druhé přerušení se vykoná)? Jak to bude při vícenásobném pokusu o přerušení?
- Totéž co v předchozím námětu, ale použijeme dvě tlačítka a dvě LED, jedno přerušení rozsvěcí na 2 s první LED, druhé přerušení druhou. Lze během výkonu přerušení 0 (na pinu 2) aktivovat i přerušení 1 (na pinu 3)? A co opačně?

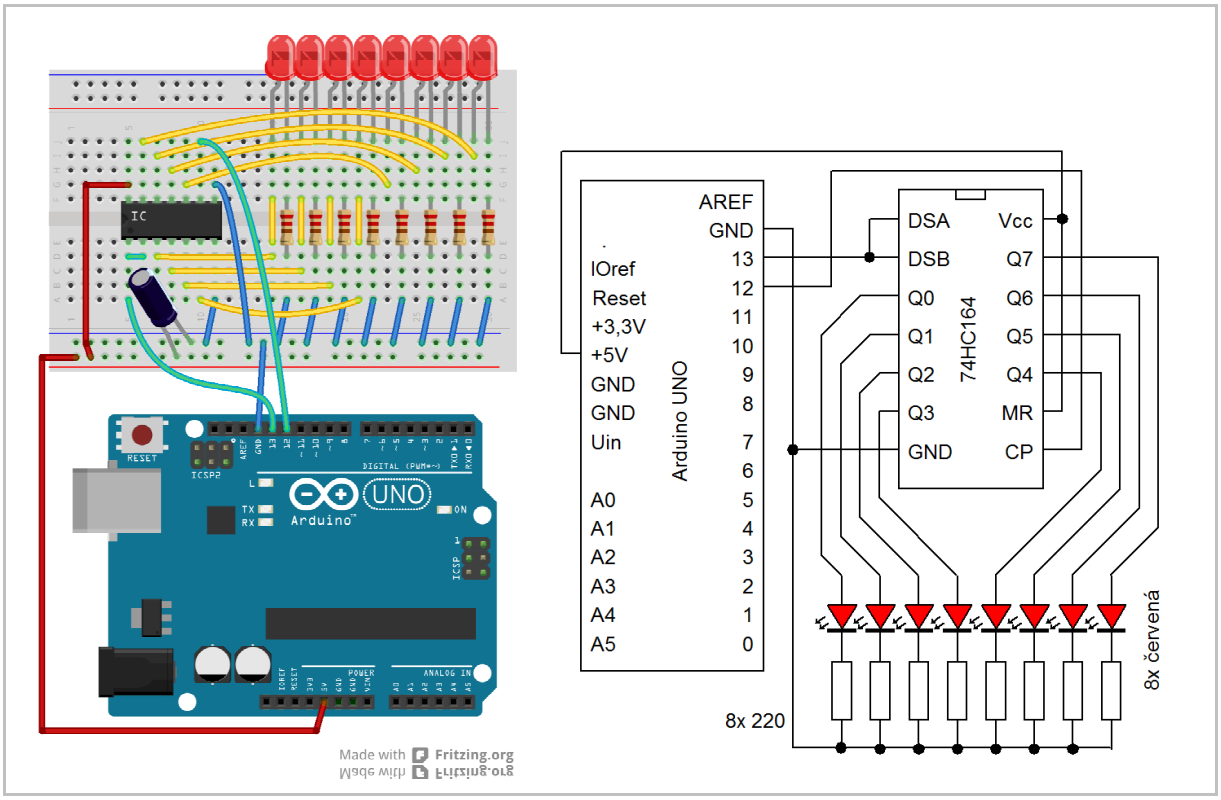
Rozšíření počtu výstupů

Vstupů a výstupů nikdy není nadbytek. Tento problém se může vyřešit zakoupením specializovaného shieldu, který rozšíří počet ovladatelných pinů, vždy ale bude takové rozšíření vyžadovat programovou obsluhu. Jeden z možných principů řešení si předvedeme v následující úloze.

Chceme ovládat 8 LED a máme na to jen dva výstupní piny. Využijeme toho, že velmi krátké (řádově mikrosekundové) bliknutí LED okem nevnímáme a na výstupy připojíme posuvný registr, v daném případě typu 74HC164. Z našeho hlediska má posuvný registr dva podstatné vstupy. Na první vstup přivedeme data (logickou úroveň) a náběžnou hranou signálu (změnou z L na H) na druhém vstupu (vstupu hodinového signálu) tato data zapíšeme do obvodu na jeho první výstup. Každá další náběžná hrana hodinového signálu zapíše další data a současně posune stávající stav na další výstup, takže stav ze vstupu 1 přejde na vstup 2, ze vstupu 2 na vstup 3 atd. Obvod má celkem 8 výstupů, stav z posledního se posunutím ztrácí. Pokud by bylo potřeba vytvořit delší posuvný registr než pro 8 bitů, dají se obvody libovolně řetězit, nám ale bude stačit jeden.

LED připojíme stejným způsobem jako k Arduinu přes rezistory 220 Ω, ale tentokrát na osm výstupů posuvného registru. Jedním pinem (13) budeme předávat do registru bit po bitu data z proměnné, jejíž obsah chceme na LED odeslat, druhým bitem (12) tato data bit po bitu odesíláme (signál hodin). Celé odeslání bude samostatným podprogramem, který dostane jako parametr odesílanou hodnotu.

Zapojení je na následujícím obrázku:



Program cyklicky posílá na LED čísla 0 až 255, takže se zobrazí na LED v binárním kódu. Aby bylo názorné, jak zobrazení funguje, je posouvání záměrně hodně zpomaleno. Při maximální rychlosti je možné zařadit za sebe 3 až 4 posuvné registry (16 až 24 bitů) a změna je znát jen jako nepatrné bliknutí, přitom nároky jsou pořád stejné, stačí dva piny.

Tento způsob rozšíření je velmi vhodný a efektivní v případě, že se stav na výstupech nemění příliš často, nevadí zpomalení a nevadí zákmity způsobené posouváním bitů při změně, typicky tedy pro zobrazení na LED včetně sedmisegmentových nebo maticových displejů. Naopak třeba pro rychlé DA převodníky se uvedený způsob naprosto nehodí.

```
// POSUV1 8 LED A POSUVNY REGISTR 74HC164 - PREDVEDENI

#define data 13                  // pin pro data
#define clk 12                   // pin pro hodiny

void setup(){                    // nastaveni
  pinMode(clk,OUTPUT);           // hodiny pro vystup
  pinMode(data,OUTPUT);          // data pro vystup
}

void posli(int hodnota){         // vyslani hodnoty na LED
  for (int i=1;i<9;i++){         // cyklus pro 8 bitů
    if (hodnota % 2 > 0){        // cislo je liche?
      digitalWrite(data,HIGH);   // ano - priprav H
    } else {                     //
      digitalWrite(data,LOW);    // ne - priprav L
    }
    digitalWrite(clk,HIGH);      // pulz hodin
    delay(50);                   // demonstracni zpomaleni
    digitalWrite(clk,LOW);       //
    hodnota = hodnota /2;        // bitove posunuti vpravo
  }
}

void loop(){                     // program
  for (int i=0;i<256;i++){       // cyklicky 0 - 255
    posli(i);                    // zobraz hodnotu
    delay(3000);                 // cekej
  }                              // konec for
}                                // konec programu
```

Náměty:

- Odstraňte zpomalení při přenosu hodnoty a ověřte, jak je vidět posunování bitů při plné rychlosti.
- Sestavte vlastní program, který demonstruje činnost posuvného registru 74HC164 při ručním řízení. Arduino bude ovládáno z PC dvěma klávesami, jedna bude dělat signál data, druhá hodiny. Arduino jen přijme povely a pošle logické urovně do obvodu.
- Upravte zapojení tak, aby 7 LED tvořilo znak H jako tečky na kostce na házení, jednu odstraňte. Sestavte vlastní program pro náhodné generování čísla na kostce s využitím zobrazení přes 74HC164.

Práce s knihovnami – ovládáme modelářské servo

Velkou výhodou Arduina je dostupnost velkého množství knihoven funkcí, které je možné přičlenit k vlastnímu programu a využívat bez detailní znalosti jejich funkce, v podstatě se musíme seznámit jen s tím, jaké parametry dané funkci předat a jaké hodnoty nám vrátí nebo jakou akci provede. Podíváme-li se do menu – Sketch – Import Library, nabídne se nám několik standardních knihoven.

Jsou to:

- **EEPROM** – čtení a zápis do „trvalé“ paměti EEPROM
- **Ethernet** – pro připojení k internetu s využitím shieldu „Arduino Ethernet Shield“
- **Firmata** – pro komunikaci s aplikacemi běžícími v PC s využitím standardního sériového protokolu
- **GSM** – pro spojení s GSM/GRPS sítí s využitím shieldu „GSM shield“
- **LiquidCrystal** – pro řízení LCD displejů
- **SD** – pro čtení a zápis dat na SD karty
- **Servo** – pro řízení modelářských serv
- **SPI** – pro komunikaci se zařízeními využívajícími sběrnici SPI
- **SoftwareSerial** – pro sériovou komunikaci přes kterékoli digitální piny
- **Stepper** – pro řízení krokových motorů
- **TFT** – pro výpis textů, vykreslení obrázků a obrazců na TFT obrazovce
- **WiFi** – pro připojení k internetu s využitím shieldu „Arduino WiFi shield“
- **Wire** – pro obsluhu dvouvodičové sběrnice (TWI/I2C) k vysílání a příjmu dat v síti zařízení a senzorů

Už ze stručného popisu je vidět, že některá omezení Arduina, která jsme si v průběhu práce uváděli, vlastně neplatí. Například bez knihoven pracuje Arduino jen s jednou sériovou linkou s HW podporou propojující jej s PC. Volně ji může využít pro jiné zařízení, jen když se vzdáme komunikace s PC. S knihovnou SoftwareSerial dostaneme další programově obsluhované sériové linky využitelné pro libovolné účely.

Práci se standardními knihovnami si vyzkoušíme na obsluze modelářského serva. Knihovnu přičleníme ke svému programu výběrem v menu, na začátek programu se okamžitě přidá řádek s příkazem #include, v daném případě #include <Servo.h> . Tím jsou zpřístupněny funkce knihovny, jejich seznam a popis lze najít na domácích internetových stránkách Arduina *http://arduino.cc/en/Reference/Servo* pod položkou Reference – Libraries. Pro tuto knihovnu si jednotlivé příkazy probereme podrobně a využijeme je v několika úlohách.

Nejdůležitější vlastnosti modelářského serva

Miniaturní modelářské servo se připojuje třížilovým kabelem. Krajní černý nebo hnědý vodič je zem (GND). Prostřední červený nebo oranžový vodič slouží k napájení napětím +5 V (servo většinou pracuje asi od napětí 3,3 V a neuvádí-li výrobce výslovně něco jiného, může být napájeno napětím až 6,0 V). Krajní bílý nebo žlutý vodič přenáší signál pro řízení serva. Typickým signálem jsou kladné pulzy o amplitudě nejméně 3 V (nebo v rozsahu napájecího napětí), které se opakují 50x za sekundu (s periodou 20 ms) a jejich šířka se mění standardně od 1,000 do 2,000 ms. Rozsah řízení bývá reálně větší, nejméně od 0,8 do 2,2 ms, Arduino v příkazech povoluje rozsah širší, ten už ale nemusí každé servo akceptovat. Střední poloha (neutrál) odpovídá středu intervalu řídících pulzů tj. 1,500 ms a převod šířky pulzu na úhlovou výchylku by měl být teoreticky lineární.

Výstupní pákou nebo kotoučem serva nikdy neotáčíme násilím, převody malých serv se mohou snadno nevratně poškodit! Nejčastější poškození je stržení plastových zubů v převodovce. Každé servo má od výrobce udávané nejméně dva základní parametry, a to moment a rychlost přeběhu výchylky daného úhlu (obvykle 60º). Oba parametry jsou víceméně v praxi nedosažitelné, protože moment se udává v okamžiku zastavení serva vnější silou respektive zatížením a když se servo už přestalo pohybovat, neplní svou úlohu. Podobně rychlost (doba přeběhu) se udává zcela bez zatížení a v ideálních podmínkách.

Pro nás je důležité, že každé zatížení serva vede k poklesu rychlosti jeho pohybu. Jakmile klesne rychlost pohybu k polovině rychlosti bez zatížení (lze určit i „od oka“), je zatížení serva asi tak na maximu, jaké je schopno opakovaně vydržet. To, že takové zatížení servo vydrží, ještě neznamená, že může být takto zatěžováno v trvalém provozu. Už krátké intenzivní zatížení vede ke vzrůstu teploty serva a s tím ruku v ruce jde snížení jeho využitelného momentu i rychlosti. Čím větší zatížení, tím delší přestávky na vychladnutí by mělo servo dostat. Dobrým vodítkem je teplota, asi do 40 ºC je vše bez problémů, mezi 40º a 60 ºC je servo schopné pracovat s přestávkami a nad 60 ºC se dostáváme do oblasti, kdy je servo ohroženo.

Typická výchylka výstupu při standardním řízení serva (1,0 – 2,0 ms) je kolem ±60º, ale může se hodně lišit typ od typu. Při rozšířeném buzení v rozsahu Arduina by mělo být servo schopné výchylky kolem ±90º. V okrajích rozsahu jsou možné výraznější nelinearity pohybu! Mechanické dorazy serva dovolují rozsah pohybu v úhlu od 210º do 270º podle typu, nicméně neutrál nemusí být v polovině tohoto rozsahu dorazů. Při využívání rozšířeného buzení serva musíme vždy pohlídat, aby výstup serva nikdy nenarážel na mechanické dorazy. Pokud se to stane, zablokovaným motorem jde plný proud a servo se během několika málo desítek sekund zničí! Spálí se vinutí motoru nebo zničí servozesilovač, obě možnosti jsou prakticky neopravitelné (oprava bývá dražší, než nové servo).

Standardní konektor serva má dutinky s roztečí 2,54 mm, stejného rozměru, jako jsou kontakty v kontaktním poli. Pro připojení servokonektoru do pole použijeme „hřebínek“ se třemi kontaktními kolíky, který zasuneme jedním koncem do servokonektoru (tím vlastně vytvoříme ze zásuvky zástrčku), druhým do kontaktního pole.

Knihovna Servo – attach()

Příkaz attach přiřadí proměnnou servo ke zvolenému digitálnímu pinu, typicky se použije v nastavení a nahradí definování pinu jako výstupu. Attach má tři parametry, první je povinný a určuje pin, ostatní dva jsou nepovinné. Druhý parametr udává délku řídícího pulzu v mikrosekundách pro (krajní) výchylku serva, kterou budeme považovat za výchozí (0º = implicitní hodnota 544 µs), třetí parametr podobně délku řídícího pulzu v mikrosekundách pro druhou (krajní) výchylku serva s úhlem 180º (implicitně 2400 µs). Podaří-li se správně pro dané servo nastavit parametry v příkazu attach, bude správně pracovat i rozsah pohybu zadaný úhlem. Ovládání serv využívá stejné prostředky mikrokontroléru jako PWM modulace na pinech 9 a 10, proto tyto dvě věci nelze používat současně.

Příklad:

```
servo.attach(10);          // přiřadí proměnnou servo k pinu 10, ponechá implicitní meze
                           // 544 a 2400 µs
servo.attach(9,620,2340);  // přiřadí servo k pinu 9, rozsahu 180º odpovídá řízení
                           // 620 až 2340 µs
```

Knihovna Servo – write()

Příkaz write zadává polohu serva. Má jeden parametr, a to úhel v rozsahu 0 až 180º, kam má servo nastavit svůj výstup. Pracuje s mezemi pulzů předem udanými příkazem attach.

Příklad:

```
servo.write(90);        // nastaví výstup do středu intervalu
servo.write(132);       // jsou-li správně nastaveny meze pro pohyb v úhlu 180º,
                        // nastaví výstup na polohu 132º
```

Knihovna Servo – writeMicroseconds()

Příkaz writeMicroseconds zadává polohu serva přímo v délce řídícího pulzu udaného v mikrosekundách.

Příklad:

```
servo.writeMicroseconds(1500);      // nastaví servo na standardní střed výchylky
```

Knihovna Servo – read()

Funkce vyžaduje jako parametr proměnnou typu servo. Přečte a vrátí polohu serva ve formě úhlu v rozsahu 0 až 180º. Je nutné si uvědomit, že funkce nemůže znát a přečíst skutečnou polohu serva, vrací jen poslední polohu zapsanou příkazem write. Pokud tedy dojde k poruše serva nebo ještě servo nestihlo po povelu dojet do požadované polohy, funkce read to nerozpozná!

Knihovna Servo – attached()

Funkce attached vrací hodnotu true, je-li dané servo přiřazeno k nějakému pinu, hodnotu false pokud přiřazeno není.

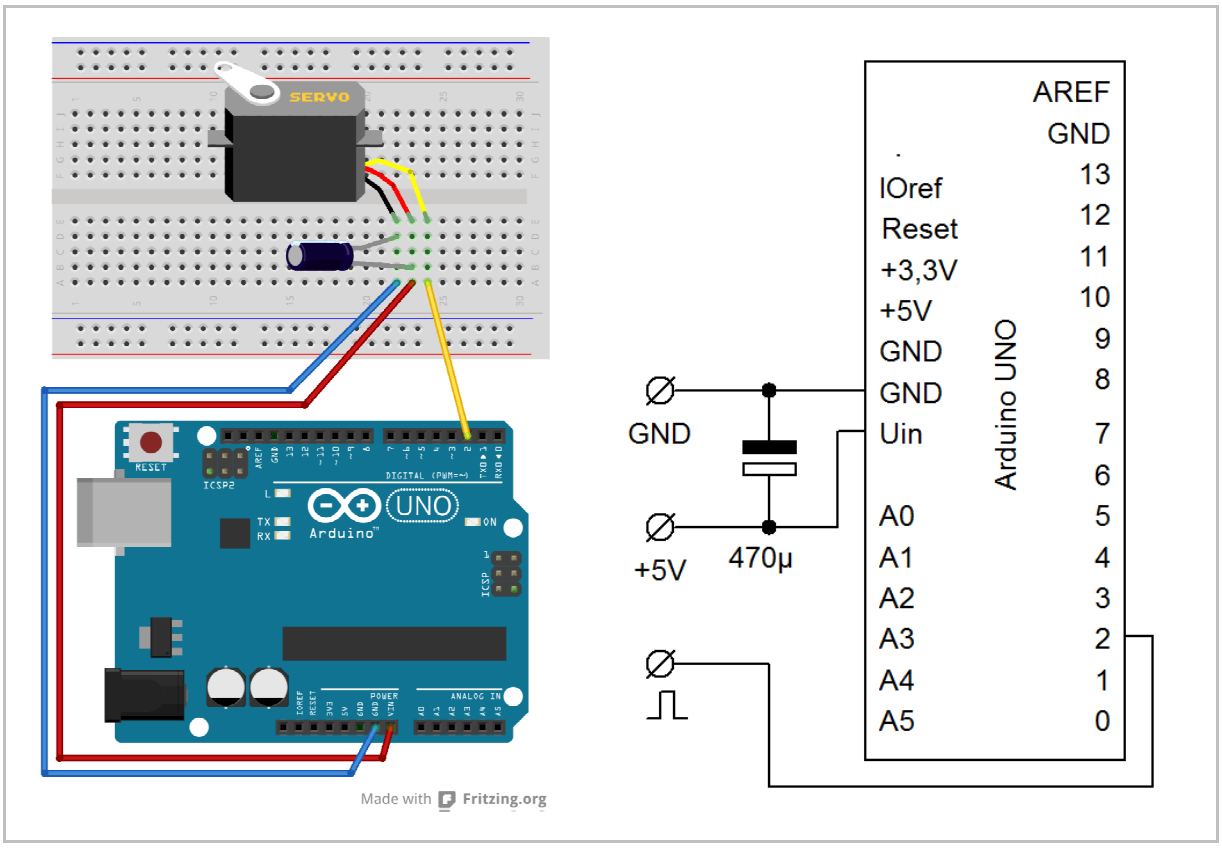
Knihovna Servo – detach()

Odpojí (zruší přiřazení) dané servo od jeho řídícího pinu. Funkce má význam zejména v případě, že potřebujeme nadále použít pin jiným způsobem než k řízení serva. Je-li nějaké (alespoň jedno) servo přiřazeno, nelze používat piny 9 a 10 pro funkci PWM.

Servo – cyklické pohyby

Máme základní popis toho, co nám nabízí knihovna Servo, takže můžeme vyzkoušet první zapojení a program. Na výstup serva nasadíme prodlouženou páku a zatím ji nebudeme zajišťovat šroubkem. Řídící signál serva bude připojen na pin 2, budeme kývat servem mezi krajními polohami ve standardním rozsahu řízení tj. 1 000 až 2 000 µs. Jakmile připojíme servo k Arduinu, je podobně, jako když jsme zkoušeli stejnosměrný motor, důležité na napájecí vodiče připojit elektrolytický kondenzátor 470 µF kvůli pokrytí špiček proudového odběru a také pohlcení přepěťových špiček, které mohou při pohybu serva vznikat.

Miniaturní servo, které je součástí sady, lze bez problémů napájet i ze stabilizátoru 5 V, který je součástí Arduina. My máme Arduino napájené z USB počítače a zapojíme napájení serva přímo na toto napětí, tím se lépe oddělí případné negativní vlivy od Arduina. Pokud bychom chtěli pracovat s většími modelářskými servy, bylo by nutné jim zajistit odpovídající nezávislé napájení, například o něco větší miniserva již mohou mít ve špičkách odběr kolem 2 A, což přesahuje možnosti stabilizátoru napájení USB. Větší rychlá serva s momentem kolem 100 N.m celkem běžně odebírají ve špičkách kolem 10 A, to je zcela mimo možnosti napájení z Arduina nebo z USB.



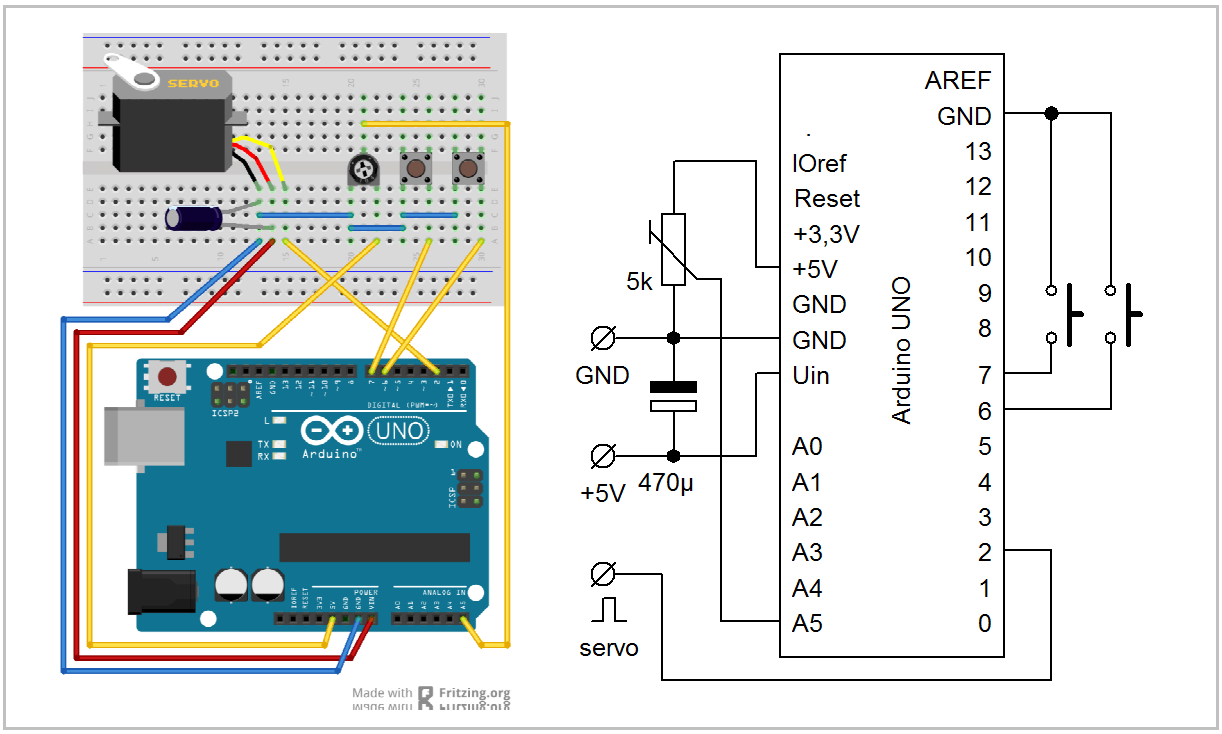

```
// SERVO1 - OVLADANI MODELARSKEHO SERVA - PREJEZDY 1-2 ms

#include <Servo.h>                    // knihovna Servo
Servo vigor1;                         // nazev serva vigor1

void setup(){                         // nastaveni
  vigor1.attach(2);                   // vigor1 priradit pinu 2
}

void loop(){                          // program
  vigor1.writeMicroseconds(1000);     // pulzy 1,00 ms pro vigor1
  delay(2000);                        // pockat 2s na pohyb
  vigor1.writeMicroseconds(2000);     // pulzy 2,00 ms pro vigor1
  delay(2000);                        // pockat 2s na pohyb
}                                     // konec programu
```

Servo by mělo přejíždět mezi dvěma polohami v úhlu něco mezi 90º a 120º. Pokusíme se změřit úhel, který páka serva přejíždí, to je jeho výchylka na standardní budicí signál. Potom program upravíme, obě dvě polohy (délky pulzů) nahradíme hodnotou 1 500 µs, to by mělo odpovídat střední poloze serva. Program přeneseme do Arduina, servo nastaví páku do střední polohy. Sejmeme páku z výstupu a nasadíme ji tak, aby byla kolmo na delší rozměr serva. To mám umožní, abychom měli servo při pokusech položené na stole a páka nikam nenarážela ani při výchylce kolem 180º. Páku zajistíme šroubkem.

Servotester s trimrem a tlačítky

V dalším kroku ověříme, jaké jsou mezní polohy daného serva. Připravíme si rozšířené zapojení s trimrem a dvěma tlačítky pro ovládání polohy. Nejprve využijeme tlačítka. Jedním tlačítkem posunujeme páku v jednom směru, druhým opačně. Do monitoru v PC se vypisuje aktuální délka řídícího impulzu.

```
// SERVO2 – SERVOTESTER S TLACITKY

#include <Servo.h>                    // knihovna Servo
Servo vigor1;                         // nazev serva vigor1
int poloha = 1500;                    // promenna pro zmeny polohy serva

void setup(){                         // nastaveni
  vigor1.attach(2);                   // vigor1 priradit pinu 2
  pinMode(6,INPUT_PULLUP);            // pin 6 pro tlacitko
  pinMode(7,INPUT_PULLUP);            // pin 7 pro tlacitko
  Serial.begin(9600);                 // seriovy port 9600Bd
}

void loop(){                          // program
  vigor1.writeMicroseconds(poloha);   // pulzy 1,00 ms pro vigor1
  delay(100);                         // pockat 0,1 s
  if (digitalRead(6)== LOW){          // tlacitko prodluzovani pulzu
    poloha = poloha + 5;              // polohu zvysit o 5
  }
  if (digitalRead(7)== LOW){          // tlacitko zkracovani pulzu
    poloha = poloha – 5;              // polohu snizit o 5
  }
  Serial.println(poloha);             // vypsat aktualni polohu
}                                     // konec programu
```



Tlačítko přidržíme tak dlouho, dokud se páka nepřestane otáčet v daném směru. Servo buď dál nereaguje a je zcela v klidu, nebo tiše vrčí a chvěje se. Pak druhým tlačítkem necháme udělat malý viditelný pohyb zpět, servo by už v této poloze mělo zůstat v klidu.

POZOR! Nenecháme servo narážet na doraz déle než několik sekund, jinak se může poškodit!

Poznamenáme si hodnotu vypisovanou na sériovém monitoru, to bude mez pro řídící pulz. Stejně to uděláme i pro druhou mez a opět si poznamenáme hodnotu. Je-li rozsah pohybu větší než 180º, změříme rozsah pro 180º, pokud je menší, servo bohužel tak velkou výchylku nezvládne. Řekněme, že jsme naměřili 2 350 a 550 µs a tento rozsah právě odpovídal výchylce 180º. Tyto hodnoty si pro dané servo poznamenáme, budeme je při jeho použití nastavovat.

Jako další úlohu si vyzkoušíme servotester ovládaný trimrem, teď už ale se znalostí možného rozsahu pohybu serva a s ovládáním přes úhly. Výpis do PC už bude zbytečný.

```
// SERVO3 – SERVOTESTER S TRIMREM

#include <Servo.h>                // knihovna Servo
Servo vigor1;                     // nazev serva vigor1

void setup(){                     // nastaveni
  vigor1.attach(2,550,2350);      // vigor1 priradit pinu 2 a nastavit meze
}

void loop(){                      // program
  vigor1.write(map(analogRead(5),0,1023,0,180));  // uhel dle trimr
  delay(100);                     // pockat 0,1s
}                                 // konec programu
```

Náměty:

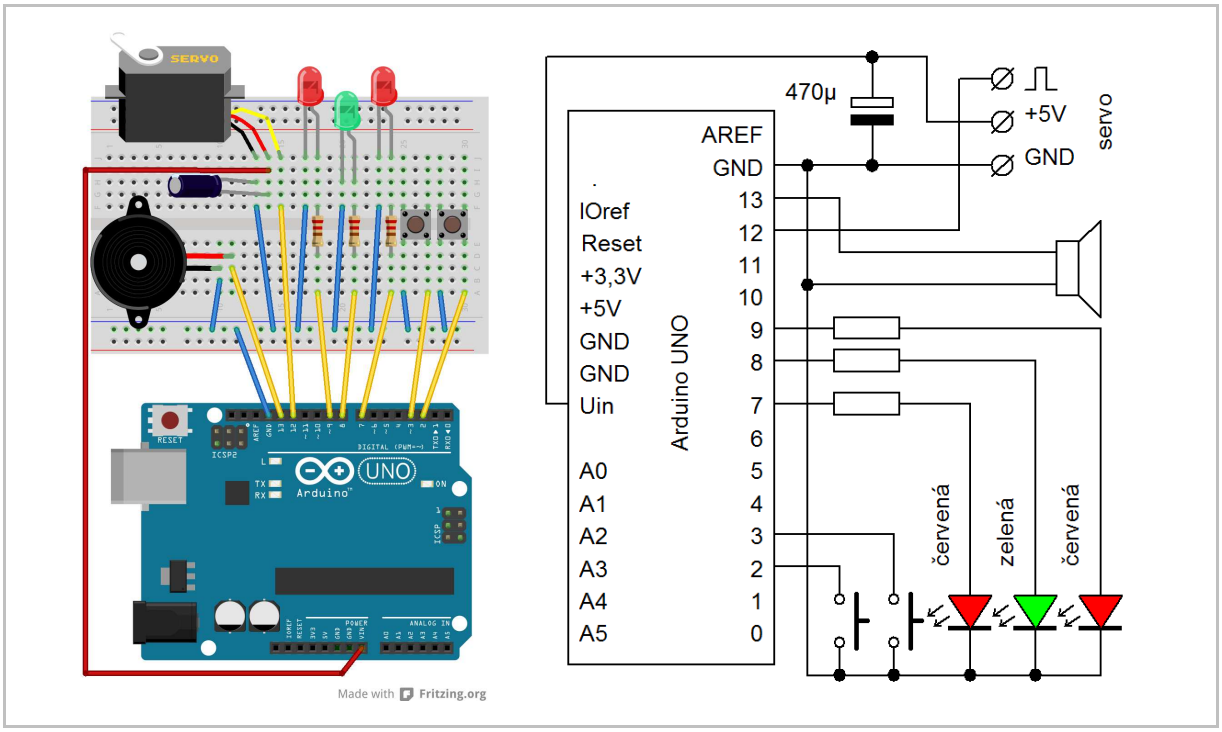
- Rozšiřte funkci programu z poslední řešené úlohy tak, aby po stisku jednoho z tlačítek servo okamžitě přejelo do střední polohy (pulz 1,5 ms). Dokud se nebude měnit poloha trimru, servo ve střední poloze zůstane. Jakmile se poloha trimru změní (o víc než 1%), servo začne okamžite podléhat řízení trimru.
- Sestavte vlastní program, servo bude v dříve zjištěných mezích pro výchylku 180º ovládáno z PC zadáním úhlu, kam se má jeho jeho výstup nastavit.
- Sestavte vlastní program. Výchozí a současně klidová poloha páky serva je v krajní poloze. Po stisku jednoho z tlačítek kývne servo pákou do druhé krajní polohy a hned ji nechá vrátit zpět do klidové polohy. Podmínkou je, že toto kývnutí nebude zdržovat mikrokontrolér od další činnosti (nepoužívejte delay).
- Každé servo má určité omezení citlivosti a nereaguje na jakkoli malou změnu řídídího pulzu. Pokud se pulzy mění velmi pomalu, nezatížené servo začne „krokovat“, poskočí setrvačností dokonce před požadovanou polohu, pak zůstane nějakou dobu v klidu až se zase nahromadí odchylka, znovu poskočí. Sestavte vlastní program, který bude v sekundových intervalech posunovat páku o 1º. Ověřte, jestli dané servo je vždy schopné reagovat na tak malou změnu polohy.
- Totéž jako v předchozí úloze, ale změřte citlivost serva podstatně přesněji tak, že budete zadávat změnu polohy v mikrosekundách. Na jak velkou změnu změnu polohy servo spolehlivě reaguje?

Závora

V následující úloze se budeme snažit vytvořit řízení závory s optickou i zvukovou signalizací, částečně podobnou železničnímu přejezdu. Závora je ovládaná dvěma tlačítky, jedním se spustí zavírání, druhým otevírání. Před tím, než se závora začne pomalu zavírat, a v průběhu zavírání, blikají střídavě dvě červená stvětla a zní přerušovaný zvukový signál. Při zavření závory svítí obě červená světla trvale a to až do úplného otevření závory. Během otevírání zní opět přerušovaný zvukový signál, po otevření závory svítí trvale jedno zelené světlo. Výchozí stav je závora otevřena.

Budeme muset změnit rychlost pohybu serva ovládajícího závodu s pohybem v úhlu 90º. Zrychlit nejde (snad jedině malým zvýšením napájecího napětí), ale zpomalení můžeme zařídit programově. Tato úloha není samoúčelná, naopak v praxi se velmi často musí omezit rychlost pohybu serva nebo zajistit „měkký“ rozběh i doběh na požadovanou polohu. Použije-li se silnější a rychlé servo (schopné například momentu 200 N.cm a rychlosti přeběhu 0,05 s/60º) a to se bude rozbíhat i dobíhat s plným výkonem, často se prudkými rázy poškodí převodovka serva, přetrhne řemínek navazujícího náhonu, opakované nárazy způsobí povolování šroubových spojů a podobně. Prudké rozběhy a zastavení také způsobují několikanásobně vyšší proudový odběr ve špičkách než postupné rozběhy a zpomalování.

Přejezd páky serva rozdělíme na velké množství kroků (změn polohy), které budeme servu posílat v přesných časových úsecích. Využíváme poloviční výchylku serva 90º, přímo se nabízí posílat změny polohy po jednotlivých stupních. Jak často se má změna vyslat? Například pro pohyb v průběhu 5 s potřebujeme interval 5 000 ms/90º, to je přibližně 55 ms. Připravíme si zapojení podle následujícího obrázku.



```
// SERVO4 - ZAVORA SE ZPOMALENYM POHYBEM

#include <Servo.h>                          // knihovna Servo
Servo vigor1;                               // nazev serva vigor1
int poloha = 90;                            // promenna pro polohu zavory

void setup(){                               // nastaveni
  vigor1.attach(12,550,2350);               // vigor1 pin 2 a nastavit meze
  pinMode(2,INPUT_PULLUP);                  // pin 2 pro tlacitko
  pinMode(3,INPUT_PULLUP);                  // pin 3 pro tlacitko
  pinMode(7,OUTPUT);                        // pin 7 cervena LED
  pinMode(8,OUTPUT);                        // pin 8 zelena LED
  pinMode(9,OUTPUT);                        // pin 9 cervena LED
  pinMode(13,OUTPUT);                       // pin 13 repro
  vigor1.write(90);                         // pocatecni otevreni zavory
  digitalWrite(8,HIGH);                     // sviti zelena LED
}

void loop(){                                // program
  if (digitalRead(2)==LOW && poloha == 90){// zavrit zavoru
    digitalWrite(8,LOW);                    // zelena LED zhasnout
    for (int i=0; i<6; i++){                // 6 cyklu blikani
      digitalWrite(7,HIGH);                 // jedna cervena
      digitalWrite(9,LOW);                  //
      tone(13,800,250);                     // vystrazny ton 0,25s
      delay(500);                           // cekani 0,5s
      digitalWrite(9,HIGH);                 // druha cervena
      digitalWrite(7,LOW);                  //
      tone(13,800,250);                     // vystrazny ton 0,25s
      delay(500);                           // cekani 0,5s
    }                                       // bude se zavirat
    digitalWrite(7,HIGH);                   // obe cervene svitit
    for (int i=90; i>-1; i--){              // zpomaleny pohyb serva
      if (i % 10 == 0){                     // kazdych 0,5s
        tone(13,800,250);                   //   vystrazny ton 0,5s
      }
      vigor1.write(i);                      // pohyb serva
      delay(50);                            //   1 stupen za 0,05s
    }
    poloha = 0;                             // zavora zavrena
  }
  if (digitalRead(3)==LOW && poloha == 0){ // otevrit zavoru
    for (int i=0; i<91; i++){               // cyklus pro 90stupnu
      if (i % 10 == 0){                     // kazdych 0,5s
        tone(13,800,250);                   //   vystrazny ton 0,25s
      }
      vigor1.write(i);                      // pohyb serva
      delay(50);                            //   1 stupen za 0,05s
    }
    poloha = 90;                            // zavora otevrena
    digitalWrite(7,LOW);                    // obe cervene zhasnout
    digitalWrite(8,HIGH);                   // zelena LED rozsvitit
    digitalWrite(9,LOW);                    //
  }                                         // konec otevirani
}                                           // konec programu
```

Náměty:

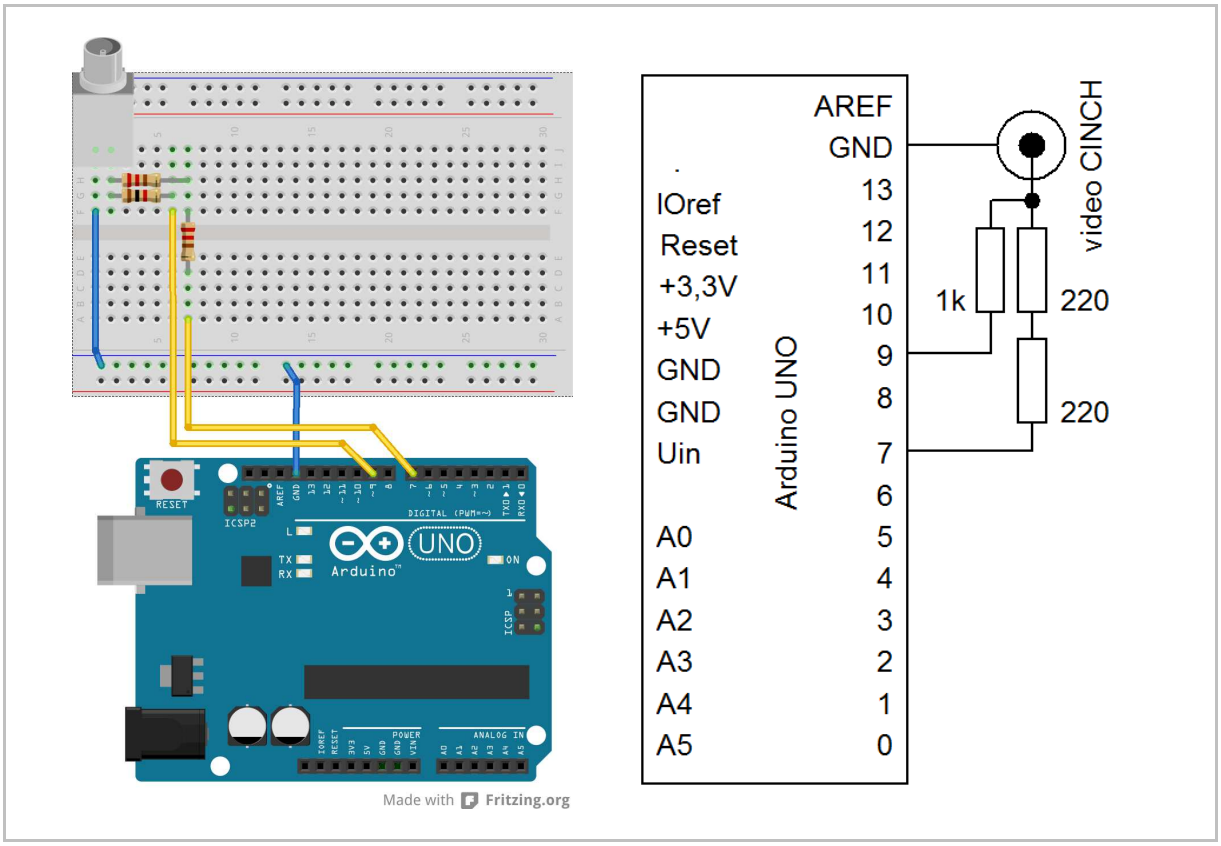
- Upravte program tak, aby při otevřené závoře zelená LED pravidelně pomalu blikala. Blikání nesmí omezovat okamžitou reakci na stisk tlačítka pro zavření závory.
- Doplňte zapojení a upravte program tak, aby se rychlost blikání a s ní současně výstražného zvukového signálu nastavovala pomoci trimru v rozsahu jeden cyklus za 0,5 až 2 s (zatím to bylo 1x za sekundu).
- K předchozímu námětu doplňte do zapojení druhý trimr a upravte program tak, aby se výška tónu výstražného zvukového signálu jím dala nastavit v rozsahu od 400 do 1200 Hz.
- Upravte původní program tak, aby se závora nepohybovala konstantní sníženou rychlostí, ale postupně se rozběhla do plné rychlosti serva a opět postupně zastavila v cíli. Pro jednoduchost po dobu pohybu závory nebudou nadále blikat červená světla (budou jen svítit) a nebude znít přerušovaný zvukový signál.

Když chceme vidět víc…

Jednočipové mikrokontroléry, a Arduino není ve své podstatě nic jiného, než jednočipový mikrokontrolér ATMega umístěný spolu s obvody napájení a nezbytnou komunikací s PC na jednu desku s plošným spojem, jsou určeny především k obsluze jednodušších technických zařízení. Snímají povelová tlačítka i údaje z čidel a podle toho řídí motory, spínají ventily, signalizují obsluze stav a podobně. Mohou toho umět mnohem víc, ale toto je jejich základní poslání. Mezi časté činnosti patří také zobrazení nezbytných údajů na displeji, v dnešní době většinou jde o LCD znakové displeje s jedním až čtyřmi řádky po 8 až 20 znacích.

Komunikaci s PC a výpis hodnot do sériového terminálu spuštěného v něm jsme používali především pro ladění programů a jednoduché zjištění hodnot získaných mikrokontrolérem. Při psaní programu je to způsob, který je vždy k dispozici a nevyžaduje žádné další technické prostředky. Nicméně sériový terminál má podstatnou nevýhodu, cokoli na něm vypsaného už nejde změnit, text může jen rolovat nahoru, grafické možnosti jsou velmi omezené. Kromě toho cílem většinou je, aby Arduino pracovalo samostatně a nezávisle na PC.

V závěrečné sérii úloh si ukážeme jinou možnost velmi efektivního textového i grafického zobrazení, kterou můžeme získat s nepatrnými náklady. Využije toho, co většinou máme v blízkosti k dispozici, televizního přijímače, respektive videomonitoru. Současně si na tomto příkladu vyzkoušíme práci s jinými než standardními knihovnami.



Na adrese *https://code.google.com/p/arduino-tvout/downloads/list* je přehled verzí knihoven pro generování videosignál v normě PAL nebo NTSC, vybereme a stáhneme si soubor TvoutBeta1.zip. Otevřeme složku Dokumenty – Arduino a do této složky stažený soubor rozbalíme, vytvoří se tři složky: pollserial, TVout a Tvoutfonts. V obslužném programu Arduina menu – Sketch – Import Library pak už najdeme pod seznamem standardních knihoven tyto nové knihovny. Uvedený postup je v podstatě „ruční instalací“ externích knihoven.

Videosignál vyžaduje použít dva výstupní piny, jeden přenáší synchronizační pulzy pro obraz (pin 9), druhý vlastní černobílý obraz (pin 7). Můžeme případně použít i třetí pin pro výstup zvuku vedený samostatně do televize. Oba digitální obrazové signály se spojí přes dva rezistory do kompletního videosignálu a jsou vedeny na konektor CINCH. Propojovací kabel k videovstupu buď se stejným typem konektoru nebo s konektorem SCART bývá v příslušenství televize.

Generování videosignálu ve velice náročná úloha, která vyžaduje naprosto přesné časování, proto je využito přerušení, které pak už nemůžeme využívat pro svoje jiné účely. Podobně je několik příkazů, kterých se budeme muset vzdát, protože jejich činnost generování videosignálu naruší, typickým příkladem je programová komunikace po sériové lince (té základní na pinech 0 a 1 podporované přímo obvody se to netýká). Místo příkazu tone budeme muset používat podobný příkaz z knihovny TVout a podobně.

Nebudeme podrobně probírat všechny dostupné příkazy, jejich přehled najdete na adrese *https://code.google.com/p/arduino-tvout/wiki/FDcomplete*. Probereme jen ty, které se nám hodí pro předvedení možností nebo jsou důležitější. Rozlišení černobílého grafického displeje je 128x96 bodů (nebudeme jej měnit), bod 0,0 leží na obrazovce vlevo nahoře. Přičleníme-li si knihovnu TVout, vytvoří se v našem programu na začátku dva řádky:

```
#include <TVout.h>
#include <video_gen.h>
```

a my přidáme třetí s přičleněním různých fontů:

```
#include <fontALL.h>
```

TVout jmeno	vytvoření objektu jmeno třídy TVout (píše se mezi deklarace proměnných)
jmeno.begin(video)	parametr video _PAL nebo _NTSC nastaví normu a spustí generování obrazu
jmeno.clear_screen()	smaže obsah displeje
jmeno.invert()	invertuje celou plochu displeje
jmeno.set_pixel(x,y,barva)	vykreslí bod v absolutních souřadnicích x,y. Parametr barva: 0 – černá,1 – bílá, 2 – inverzně
jmeno.get_pixel(x,y)	funkce vrací barvu bodu v daném bodě x,y: 0–černá, 1-bílá
jmeno.draw_line(x1,y1,x2,y2,barva)	vykreslí čáru v absolutních souřadnicích od bodu x1,y1 do bodu x2,y2. Parametr barva: 0–černá,1–bílá, 2-inverzně
jmeno.draw_row(y,x0,x1,barva)	velmi rychle vykreslí vodorovnou čáru od bodu x0,y do bodu x1,y barvou 0–černá,1–bílá, 2-inverzně
jmeno.draw_column(x,y0,y1,barva)	velmi rychle vykreslí svislou čáru od bodu x,y0 do bodu x,y1 barvou 0–černá,1–bílá, 2-inverzně
jmeno.draw_rect(x,y,sirka,vyska,barva,vypln)	vykreslí obdélník od bodu x,y o šířce sirka, a výšce

	vyska s barvou čáry (0–černá,1–bílá, 2-inverzně) a s výplní (0–černá,1–bílá, 2-inverzně,-1-nechat původní). Parametr vypln není povinný.
jmeno.draw_circle(x,y,polomer,barva,vypln)	vykreslí kružnici se středem v bodě x,y o poloměru polomer, s barvou čáry (0–černá,1–bílá, 2-inverzně) a s výplní (0–černá,1–bílá, 2-inverzně,-1-nechat původní). Parametr vypln není povinný.
jmeno.shift(vzdalenost,smer)	posune obsah displeje o počet bodů udaných parametrem vzdalenost ve smeru 0-nahoru, 1-dolů, 2-vlevo, 3-vpravo. Obsah, který „vyjede“ ze zobrazovaného pole, se ztrácí.
jmeno.print_char(x,y,chr)	od bodu x,y vypíše znak s kódem chr
jmeno.set_cursor(x,y)	nastaví pozici textového kurzoru na bod x,y
jmeno.select_font(font)	nastaví font – parametr font4x6, font6x8, font8x8 nebo font8x8ext. Externí fonty lze předefinovat a vytvořit si tak v podstatě libovolnou grafiku.
jmeno.print(x,y,str)	vypíše na pozici x,y řetězec str, při přesahu přechází na další řádek
jmeno.print(str)	vypíše na pozici textového kurzoru řetězec str
jmeno.print(x,y,int,base)	vypíše na pozici x,y proměnnou typu int (unsigned int, long, unsigned long) udanou v desítkové nebo šestnáctkové soustavě (parametr 10, 16). Parametr base se nemusí uvádět.
jmeno.print(x,y,double,presnost)	vypíše na pozici x,y hodnotu proměnné typu double s přesností na daný počet desetinných míst. Bez udání přesnosti na 2 desetinná místa. Pozice x,y není povinná.
jmeno.delay_frame(pocet)	čeká daný počet snímků podle paramteru počet, slouží k synchronizaci s vykreslováním obrazu
jmeno.tone(frekvence, delka)	jako standardní tone, ale funguje i při generování obrazu a vždy je pro pin 11 (!)

Poznámka: všechny příkazy print existují i ve verzi println s odřádkováním

Ukázka grafiky

Nyní se pokusíme představit některé možnosti grafiky jednoduchým (ale nikoli úplně krátkým) programem. Jakmile se pustíme do programování grafiky, musíme počítat s tím, že bude třeba obsloužit hodně věcí. Programy se protáhnou, i když z hlediska obsažených myšlenek jsou stále stejně jednoduché. Naším cílem je nechat cyklicky střídat tři „obrazovky“. První ukáže základní vykreslené tvary, druhá fonty a třetí předvede náhodný „pohyb“ objektu v reálném čase. Nic se neovládá, zapojení je jen základní videoadaptér k Arduinu s dvěma rezistory (v našem případě je jeden z nich složen ze dvou kusů). Je-li to možné, nastavte televizní přijímač nebo monitor na poměr stran obrazu 4:3, pak jsou body téměř čtvercové.

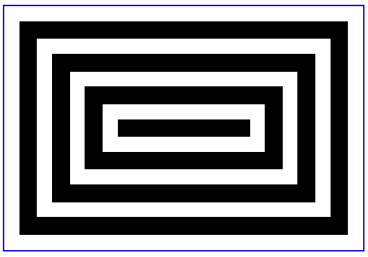
//VIDEO1 - PREDVEDENI GRAFIKY

```
#include <TVout.h>                        // knihovna TVout
#include <video_gen.h>
#include <fontALL.h>                      // vsechny fonty
TVout TV;                                 // objekt TV tridy TVout
int x1,y1,x2,y2;                          // pracovni promenne pro pohyb
float krok,krokx,kroky;                   // pracovni promenne pro pohyb

void setup(){                             // nastaveni
  TV.begin(_PAL);                         // generovani signalu PAL
  TV.clear_screen();                      // smazani obrazovky
}

void loop(){                              // smycka programu
// Cast 1 - Obrazovka Zakladni tvary
  TV.draw_rect(0,0,127,95,1);             // ramecek kolem cele plochy
  for (int i=0; i<7; i++){                // ctverce
    TV.draw_rect(3*i+8,3*i+8,50-6*i,40-6*i,1);
  }
  for (int i=0; i<5; i++){                // kruznice
    TV.draw_circle(100,28,i*5,1);
  }
  delay(500);                             // pockat 0,5s
  for (int i=5; i<123; i++){              // cary invertujici
    TV.draw_line(64,55,i,90,2);
    delay(20);
  }
  delay(6000);                            // pockat 6s
  for (int i=0; i<98; i++){               // odrolovani nahoru
    TV.shift(1,0);
  delay(5);
  }
// Cast 2 - Obrazovka Fonty a vypis hodnot
  TV.select_font(font8x8);                // nastavit font 8x8
  TV.print(0,0,“Font 8x8 - ABCD“);
  TV.select_font(font6x8);                // nastavit font 6x8
  TV.print(0,10,“Font 6x8 - ABCDEFGHIJ“);
  TV.select_font(font4x6);                // nastavit font 4x6
  TV.print(0,18,“Font 4x6 - ABCDEFGHIJKLMNOPQRST“);
  TV.select_font(font6x8);                // dale font 6x8
  for (int j=0; j<6; j++){                // vypis tabulky znaku
    for (int i=0; i<16; i++){
      TV.print_char(i*8,j*8+40,j*16+i+32);
      delay(20);
    }
  }
  delay(6000);                            // pockat 6 s
  for (int i=0; i<129; i++){              // odrolovani vlevo
    TV.shift(1,2);
    delay(5);
  }
// Cast 3 - Obrazovka nahodny pohyb
  TV.print(0,0,“Nahodny pohyb“);          // nadpis
  x1=64; y1=48;                           // pocatecni poloha
```

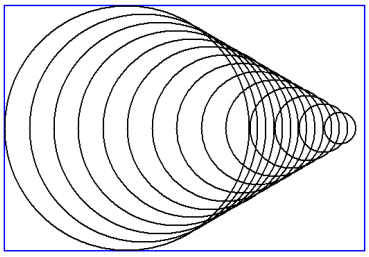
```
  for (int i=0;i<20;i++){                 // 20 pohybu
    x2=random(0,120);                     // nove souradnice
    y2=random(8,87);
    krok=sqrt(pow(x2-x1,2)+pow(y2-y1,2));// delka pohybu
    krokx=(x2-x1)/krok;                   // krok pro x
    kroky=(y2-y1)/krok;                   // krok pro y
    for (int j=0;j<int(krok);j++){        // pohyb po krocich
      TV.draw_rect(int(krokx*j+x1-1),int(kroky*j+y1),6,6,1);// vypsat
      TV.delay_frame(1);                  // pockat a synchronizovat
      TV.draw_rect(int(krokx*j+x1-1),int(kroky*j+y1),6,6,0);// smazat
    }
    x1=x2; y1=y2;                         // dalsi pohyb
  }
  for (int i=0; i<129; i++){              // odrolovani vpravo
    TV.shift(1,3);
    delay(5);
  }
}                                         // konec programu
```

Náměty:

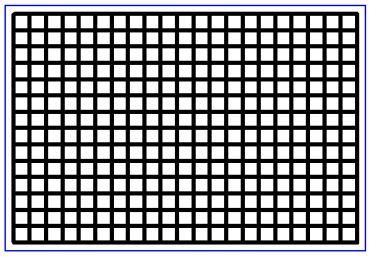
- Sestavte vlastní program, který vykreslí postupně (pozorovatelně zpomaleně) na celou plochu obrazovky soustředné obdélníky. Pro zvýraznění budou střídavě bílé a černé. (A)
- Sestavte vlastní program, který vykreslí na obrazovku soustavu kružnic s posunutým středem. (B)
- Sestavte vlastní program, který vykreslí na obrazovku síť. „Oka“ budou mít rozměr 3x3 body. (C) Potom upravte program tak, aby čáry byly černé a výplň ok bílá.



(A)



(B)

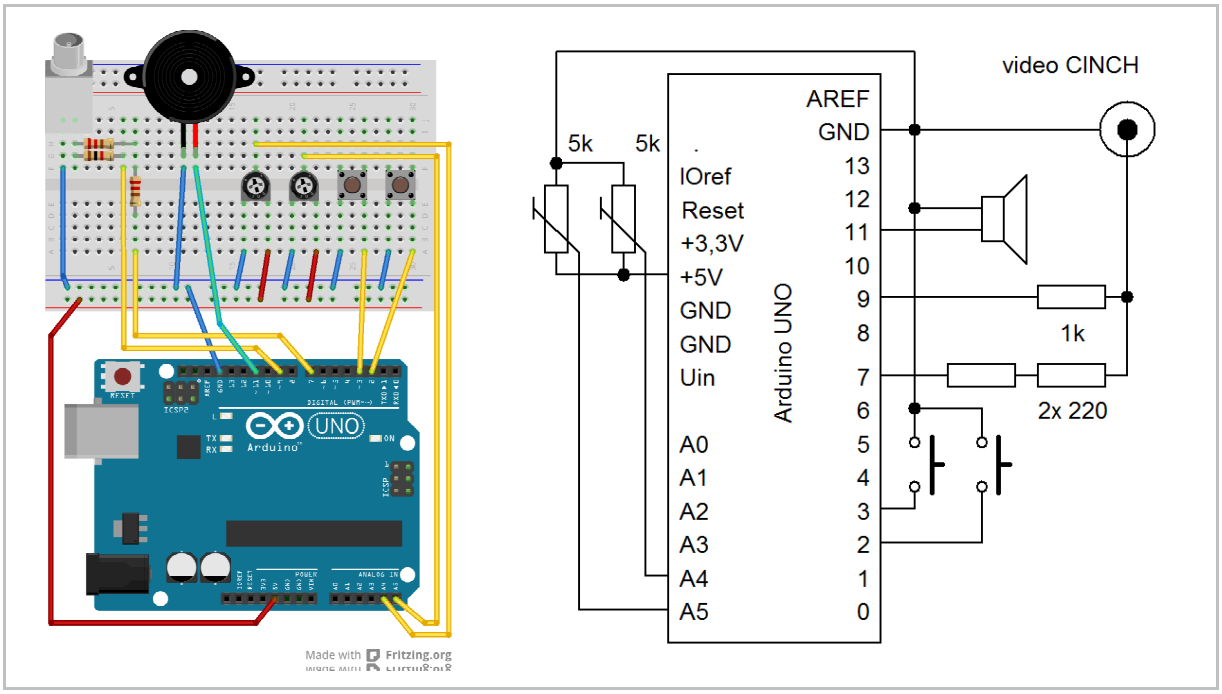


(C)

- Sestavte vlastní program, který pomalu vypíše na obrazovku tabulku znaků v rozsahu kódů 32 až 255 ve fontu 8x8 bodů. Psát se bude 16 znaků na řádek a tabulka postupně odroluje nahoru.
- Sestavte vlastní program, který fontem 6x8 vypíše na obrazovku tabulku pro malou násobilku (čísla 1 až 10 krát čísla 1 až 10).
- Sestavte vlastní program, který vykreslí tenký ráměček kolem celé ovládané plochy obrazovky a pak jej vyplní 1 000 bodů na náhodných pozicích.
- Sestavte program, který vykreslí vodorovné pravítko se stupnicí. Z plné vodorovné čáry budou dolů vybíhat krátké značky jednotek (po 5 bodech vodorovně na obrazovce), každá pátá značka bude 2x delší a každá značka desítek 4x delší. Pod značkou desítek bude vždy fontem 4x8 vypsána hodnota na stupnici, počáteční hodnota vlevo je nula.

- Sestavte program, který vykreslí na obrazovku sinusovku. Výška i šířka obrazovky budou vyplněny až do kraje a na šířku budou minimálně dvě periody průběhu.
- Sestavte vlastní program, který vytvoří digitální hodiny. Po nastavení hodin, minut a sekund (lze pro jednoduchost nastavením hodnot proměnných v programu) se bude čas zobrazovat fontem 8x8 běžným způsobem.
- Sestavte vlastní program, který vytvoří analogové kruhové hodiny s hodinovou a minutovou ručkou. Po nastavení hodin a minut se bude čas zobrazovat běžným způsobem. Bude-li část obrazu při běhu programu blikat (poblikávat), nevadí to.

Analogové měřidlo



Předchozí úloha sloužila jen k předvedení některých možností grafiky, ta další bude kratší a bude mít mnohem blíže k praktickému využití. Doplníme zapojení o dva trimry, dvě tlačítka a piezoreproduktor. Jeden trimr vytvoří změny napětí, to budeme měřit a zobrazíme pomocí virtuálního ručkového měřidla. Druhý trimr udává svým napětím mezní povolenou hodnotu. Je-li jím stanovená mez na prvním vstupu překročena, oznámí se to jako chyba přerušovaným tónem. Chyba a její indikace trvá i když se následně napětí sníží. Jedno tlačítko resetuje sledování minima a maxima, druhé tlačítko ruší hlášení chyby. Žádná část měřidla nesmí za provozu viditelně blikat.

```
//VIDEO2 - ANALOGOVE MERIDLO

#include <TVout.h>                          // knihovna TVout
#include <video_gen.h>
#include <fontALL.h>                        // vsechny fonty
TVout TV;                                   // objekt TV tridy TVout
float krok=0.004887586;                     // prevod bit/napeti V
int U1,U2,U1s,U2s,Umax,Umin,Umez;           // pracovni promenne napeti
unsigned long cykl;                         // pocitani cyklu
```

```
boolean chyba;                              // prekroceni meze napeti

void setup(){                               // nastaveni
  pinMode(2,INPUT_PULLUP);                  // vstup tlacitko
  pinMode(3,INPUT_PULLUP);                  // vstup tlacitko
  TV.begin(_PAL);                           // generovani signalu PAL
  TV.clear_screen();                        // smazani obrazovky
  TV.draw_rect(0,0,127,95,1);               // ramecek meridlo
  for (int i=14;i<=114;i=i+2){              // stupnice male dilky
    TV.draw_line(i,30,i,25,1);}
  for (int i=14;i<=114;i=i+10){             // stupnice velke dilky
    TV.draw_line(i,30,i,20,1);}
  TV.select_font(font4x6);                  // nastavit font 4x6
  for (int i=0;i<6;i++){                    // popis meritka
    TV.print(i*20+13,13,i);}
  TV.select_font(font6x8);                  // nastavit font 6x8
  TV.print(7,85,“Min“);                     // vypis Min
  TV.print(75,85,“Max“);                    // vypis Max
  Umax=0; Umin=1023; chyba=false;           // pocatecni stavy
}

void loop(){                                // smycka programu
  U1=analogRead(5);                         // mereni napeti
  U2=analogRead(4);                         // mereni meze
  TV.draw_rect(13,32,102,52,0,0);           // mazani rucky
  TV.draw_line(64,85,map(U1,0,1023,14,114),32,1); // rucka
  TV.draw_circle(64,85,3,1,1);              // stred meridla
  if (U1>Umax)Umax=U1;                      // kontrola mezi
  if (U1<Umin)Umin=U1;                      //
  TV.select_font(font6x8);                  // nastavit font 6x8
  TV.print(28,85,float(Umin*krok));         // vypis Min
  TV.print(98,85,float(Umax*krok));         // vypis Max
  Umez=map(U2,0,1023,14,114);               // vypocet meze
  TV.set_pixel(Umez,32,1);                  // ukazatel meze
  TV.draw_row(33,Umez-1,Umez+2,1);          // ukazatel meze
  if(digitalRead(2)==LOW){                  // TL - mazani min a max
    Umax=0; Umin=1023;}
  if(digitalRead(3)==LOW) chyba=false;      // TL - mazani chyby
  if(U1>U2) chyba=true;
  cykl++;                                   // pocitani cyklu
  if(chyba && cykl%25==0) TV.tone(800,250);// zvuk chyby
  TV.delay_frame(1);                        // synchronizace
}                                           // konec programu
```

Na tomto programu stojí za to si podrobněji všimnout nejen používání jednotlivých grafických příkazů a způsobu vytvoření virtuálních měřidel, ale ještě tří věcí. Podstatná část programu je uložena v bloku, který se provede jen jednou (void setup), a nejsou to zdaleka jen dekladace proměnných a nastavení pinů, ale vlastně vykreslení všeho, co se nebude při další práci měnit. Jen ty části měřidla, které se mění, čili musí překreslovat, se potom obsluhují v cyklu. Tím se výrazně šetří čas.

Druhou věcí je na první pohled nenápadná, ale velmi důležitá synchronizace, kterou má na starosti poslední příkaz. Pokud jej vypustíte, program samozřejmě poběží, ale uvidíte obrovský rozdíl. Programová smyčka trvá určitou dobu a pracuje v situaci, kdy se (naprosto pravidelně) odskakuje interruptem do zobrazení a přenáší obsah paměti na obrazovku. Když jsou tyto dva děje nezávislé na sobě, bude se stávat, že se například čára smaže těsně před tím, než by měla být zobrazena, pak proběhne zobrazení a čára se vykreslí do paměti v době, kdy už je po vykreslení. Důsledkem toho je, že čára není vidět, dokud se zase načasování zobrazení a běhu prohramu „nerozjede“. Na pohled se to projevuje jako velmi rušivé blikání některých nebo všech částí grafiky, které jsou pravidelně překreslovány.

Celá smyčka našeho programu je schopná proběhnout v době mezi vyhreslováním dat na obrazovku. Proto je na konci (mohlo by to být i na začátku) příkaz TV.delay_frame(1), který na rozdíl od „obyčejného“ delay nemá za úkol jen čekat, ale synchronizovat program podle zobrazení, čekat až proběhne vykreslení jednoho snímku na obrazovku (čas je udáván v těchto snímcích). Jeden průběh smyčky se dokončí dřív než se vykresluje, tak v době vykreslování už nedochází k žádným změnám, obraz je klidný, nebliká. Pokud by byl program složitější a trval déle, než se dá mezi snímky stihnout, bylo by nutné „rozsekat“ více příkazy delay_frame program na více částí, které by se stihnout mezi snímky daly, přitom každá část by musela plně obsloužit své objekty (čáry, rámečky, body, …). Zajistit klidný výsledný obraz znamená nejen naprogramovat úlohu tak, aby vůbec fungovala, ale současně ji celou správně načasovat a synchronizovat s generováním obrazu.

Náměty:

- Přepojte referenční napětí AD převodníků na 3,3 V a upravte program tak, aby pracoval od 0 do 3 V (vyžádá si to rozsáhlou změnu dílků stupnice i popisu stupnice, přepočtu v zobrazení, …)
- Doplňte původní program tak, aby se dosažené minimální a maximální napětí zobrazovalo nejen číselně při spodním okraji obrazovky, ale také jako vodorovná čára vedená těsně pod dílky stupnice. Zleva plná čára vede až k hodnotě minima, zprava plná čára vede až k hodnotě maxima, ručka se pohybuje stejně jako dosud ve volném prostoru mezi nimi.

- Doplňte původní program tak, aby se dosažené minimální a maximální napětí zobrazovalo nejen číselně při spodním okraji obrazovky, ale také jako krátké (asi poloviční) ručky. Pozor, neznamená to, že konec ručky bude pod příslušnou hodnotou stupnice, ale že malá ručka bude mít stejnou polohu (úhel) jako velká, když ukazuje danou hodnotu.
- Upravte původní program nebo napište nový se stejnou funkcí. Podoba virtuálního měřidla bude jiná, bude kruhové s dílky po obvodu (jako hodiny). Minimální a maximální hodnoty se zobrazí číselně vpravo vedle kruhového měřidla.
- Sestavte program, který bude zobrazovat funkční (kruhové) ručkové hodiny včetně sekundové ručky. Dbejte na to, aby žádný ze zobrazených objektů neblikal.

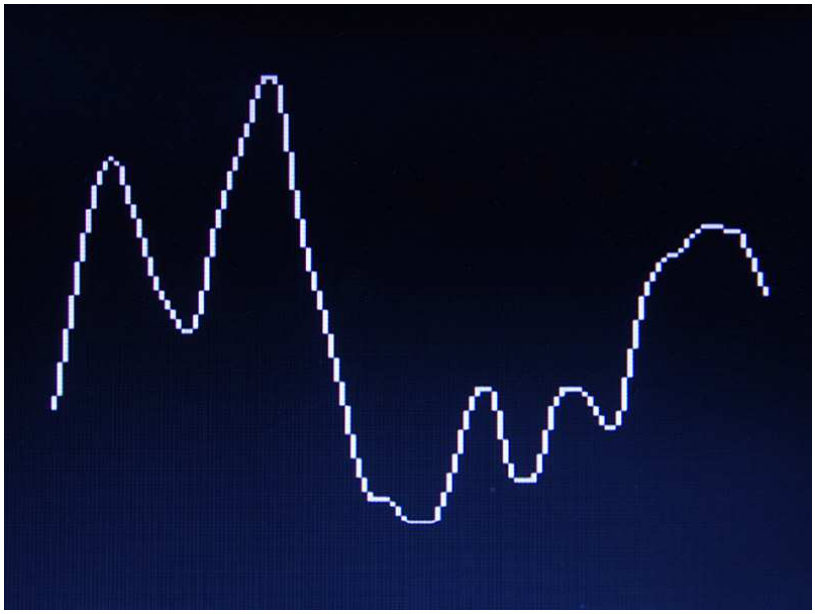
Záznam a grafické zobrazení hodnot

Budeme stále vycházet ze stejného zapojení, využije ale jen jediný trimr (A5) a pokusíme se zachytit změny napětí na jeho výstupu v čase. Obraz bude rolovat vlevo rychlostí 25 bodů/s a zaznamená křivku změn napětí za posledních přibližně 5 s. Program bude jen holý funkční, žádné stupnice s dílky ani popisy dělat už nyní nebudeme. Pokud bude změna příliš rychlá, musí se zaznamenat průběh jako strmá čára, ne samostatné body. Referenční napětí pro převodník je napájecí napětí 5 V.

```
//VIDEO3 – ZAZNAM NAPETI V CASE (25 BODU/S)

#include <TVout.h>                      // knihovna TVout
#include <video_gen.h>
TVout TV;                               // objekt TV tridy TVout
int U1,U1s;                             // promenne pro napeti

void setup(){                           // nastaveni
  TV.begin(_PAL);                       // spustit videosignal
  U1s=map(analogRead(5),0,1023,95,0);   // pocatecni hodnota
}

void loop(){                            // smycka programu
  U1=map(analogRead(5),0,1023,95,0);    // nova hodnota napeti
  TV.draw_line(126,U1s,127,U1,1);       // kresleni prubehu
  TV.shift(1,2);                        // posun obrazovky
  U1s=U1;                               // stara hodnota
  TV.delay_frame(2);                    // synchronizace na 1/25s
}                                       // konec programu
```

Náměty:

- Upravte program tak, aby snímání běželo 2x rychleji (posun o 50 bodů/s) a potom 2x pomaleji (posun o 12,5 bodů/s)



- Upravte původní program tak, aby zobrazoval velmi pomalu, posunoval o jeden bod za 2 s. Napětí se však na rozdíl od předchozích úloh bude číst mnohem častěji a program zobrazí dvě křivky, jedna bude odpovídat minumům dosaženým za jednotku času (2 s), druhá maximům. Pokud se napětí nebude měnit, obě křivky splynou. Jestliže rozdíl minima a maxima v daném časovém úseku bude větší než 2 V, následují 2 s bude znít výstražný tón
- Upravte původní program (25 bodů/s), podobně jako v přechozím případě bude zobrazovat dvě křivky, ale jedna bude odpovídat napětí z trimru na vstupu A5 a druhá trimru na vstupu A4.
- Do původního programu přidejte druhou křivku, která bude ukazovat stav tlačítka na pinu 2, výška změny bude 3 body. Pro původní křivku napětí pozměňte měřítko vykreslení, obě křivky se nikdy nesmí protnout ani dotknout.
- Upravte původní program, nebude ovládán žádným vstupem, jen bude zobrazovat ubíhající sinusovku vyplňující celý rozsah zobrazení na výšku, na šířku se musí do obrazu vejít nejméně dvě celé periody.
- Do původního zapojení přidejte generátor pulzů s obvodem 555, frekvence bude řízená trimrem 5 k a kondenzátorem 470 µF. Pro začátek nastavte nejpomalejší kmity. Signál z vývodu 2 obvodu 555 přiveďte na vstup A0 a upravte program tak, aby zobrazoval průběh tohoto signálu v rozsahu napětí 0 až 5 V.
- Totéž co předchozí námět, ale změňte rozsah zobrazení, průběh signálu musí vyplňovat (téměř) celou vykreslovanou plochu na výšku.

Je složité napsat hru?

Pokusíme se napsat velmi jednoduchou hru (či spíše hříčku) pro naše Arduino UNO s připojeným televizním přijímačem. Princip bude jednoduchý, někde na obrazovce se po stisku tlačítka náhodně objeví malý prázdný kroužek o poloměru 3 body. My máme k dispozici podobný kroužek plný a ovládáme jeho polohu dvěma trimry, jedním ve vodorovném a druhým ve svislém směru. Cílem je ovládaný kroužek překrýt s náhodně vytvořeným, testuje se vzdálenost středů menší než 4 body. Po „zásahu“ se ozve pípnutí, generovaný kroužek zmizí a vytvoří se na jiném místě, celkem 5x po sobě. Počítá se a nakonec zobrazí čas, který jsme na „zásah“ pěti kroužků potřebovali. Možnost, že se cíl náhodně objeví na pozici ovládané hráčem se nevylučuje.

Využijeme stále stejné zapojení se dvěma trimry, dvěma tlačítky a piezoreproduktorem. Pokud by měl být program využíván i jinak než jen k demonstraci funkce, musely by být trimry respektive potenciometry součástí nějakého bytelnějšího dvouosého ovladače nejlépe v podobě páky (joysticku)

//VIDEO4 - HRA ZASAHNI KROUZEK

```
#include <TVout.h>                          // knihovna TVout
#include <video_gen.h>
#include <fontALL.h>                        // vsechny fonty
TVout TV;                                   // objekt TV tridy TVout
int Xc,Yc,X,Y,Xs,Ys;                        // promenne poloha
float vzdal;                                // vzdalenost od cile
unsigned long cas;                          // ulozeni casu

void setup(){                               // nastaveni
  pinMode(2,INPUT_PULLUP);                  // vstup pro tlacitko
  TV.begin(_PAL);                           // spustit videosignal
  TV.select_font(font6x8);                  // nastaveni fontu
  X=64; Y=48;                               // pocatecni poloha
  TV.clear_screen();                        // smaz obrazovku
  TV.print(20,40,“Stiskni tlacitko“);       // uvodni text
  do {TV.delay_frame(1);}                   // cekani na TL
    while (digitalRead(2)==HIGH);
}

void loop(){                                // smycka programu
  cas=millis();                             // cas startu
  for (int i=5;i>=1;i--){                   // pro 5 pokusu
    TV.clear_screen();                      // smaz obrazovku
    TV.print(1,1,i);                        // pocet pokusu
    randomSeed(millis());                   // init random
    Xc=random(3,124);                       // poloha cile X
    Yc=random(12,91);                       // poloha cile Y
    TV.draw_circle(Xc,Yc,3,1,0);            // krouzek cile
    do {                                    // cyklus do zasahu
      X=map(analogRead(5),0,1023,3,124);    // hrac X
      Y=map(analogRead(4),0,1023,12,91);    // hrac X
      TV.draw_circle(X,Y,3,1,1);            // krouzek hrace
      TV.delay_frame(2);                    // synchronizace na 1/25s
      TV.draw_circle(X,Y,3,0,0);            // maz hrace
      vzdal=sqrt(float(pow(X-Xc,2))+pow(Y-Yc,2));
    } while(vzdal>4);                       // je prekryti
    TV.tone(400,250);                       // signal zasahu
```

```
  }                                       // 5 pokusu bylo
  TV.clear_screen();                      // smaz obrazovku
  TV.print(10,40,“Hotovo – cas:“);        // vypis konce
  TV.print(90,40,long(millis()-cas)/1000);// vypis konce
  do {TV.delay_frame(1);}                 // cekani na TL
    while (digitalRead(2)==HIGH);
  X=64; Y=48;                             // novy pocatek
}                                         // konec programu
```

Tento program šlo řešit dvěma způsoby. Buď počítat a vyhodnotit vzdálenost středů obou kružnic, protože jejich polohy jsou dostupné v proměnných a je to snazší, nebo vyhodnocovat, jestli se v prostoru pro ovládanou kružnici nenachází nějaký bílý bod na obrazovce, zkrátka pomocí čtení dat z obrazovky (TV.get_pixel()). K vyzkoušení možností, které nám toto čtení nabízí, poslouží některé z dalších námětů.

Na internetu lze najít celou řadu her pro Arduino založených na stejné knihovně TVout, kterou jsme použili, například Tetris, nebo televizní tenis. Cílem však nebylo hrát si ani psát hříčky, videodisplej z televize je nejlevnějším a současně velmi univerzálním prostředkem, jak připojit k Arduinu výstupní zařízení s podobnými znakovými i grafickými možnostmi. Nevýhodou je ovšem poměrně značná náročnost na pamět ve smyslu délky programu i obsazení RAM a také vytížení mikrokontroléru, to ale vadí málokdy.

Náměty:

- Doplňte program o obrazovku s uvodními instrukcemi ke hře a vyhodnocením času s větší přesností na 0,1 s.
- Doplňte program o průběžné vypisování vzdálenosti řízeného kroužku od cílového do pravého horního rohu obrazovky
- Sestavte vlastní program, který nejprve vypíše do horní poloviny obrazovky nějaký nápis a obrazec a potom obsah horní poloviny obrazovky přezrcadlí do dolní poloviny obrazovky (písmo bude „hlavou dolů“). Program musí pracovat na základě čtení obsahu obrazovky
- Sestavte vlastní program, který nejprve popíše obrazovku textem, a potom celý obsah obrazovky stranově převrátí (nahoře a dole je zachováno, ale přehodí se pravá a levá strana)
- Sestavte vlastní program, který nejprve vykreslí (podle zadaného zadání v programu) na obrazovku rámeček kolem plochy a několik kratších čar jako překážky. „Míček“ v podobě obdélníčku pak bude v tomto prostoru po počátečním náhodném zadání směru putovat (létat) a bude se odrážet od všech překážek i okrajů, přitom úhel dopadu je roven úhlu odrazu. Stisk tlačítka vygeneruje nové náhodné umístění a směr pohybu. Řešení musí k detekci překážek využívat výhradně čtení bodů z obrazovky.

Závěr

Po úlohách obsažených v této příručce i vlastních programech podle uvedených námětů jistě přijdou na řadu také vlastní projekty a s nimi potřeba využívat různé knihovny. Na rozdíl od standardních knihoven, které jsou poměrně dobře popsány a mnohokrát vyzkoušeny, k těmto specializovaným knihovnám často není úplná dokumentace zejména co se týče typů parametrů (proměnných), požadavků na prostředky mikrokontroléru (časovače, přerušení, …) a další podmínky. Proto v případě stažení a používání podobných knihoven od uživatelů z internetu důkladně prověřte vlastnosti dané knihovny. Případné problémy nemusí být jen důsledkem chyb ve vlastním programu, knihovny se zkrátka mohou chovat jinak, než očekáváte.

Podobně, jestliže časem nabídnete svoje vlastní knihovny nebo podprogramy ostatním, doplňte do nich velmi podrobný popis vstupů i výstupů a také poznámky o použitých prostředcích mikrokontroléru a možné kolizi s jinými funkcemi (příkazy). Dokumentujte i to, ce se vám zdá jasné, ostatním to na první pohled jasné být nemusí.

Na tuto základní sadu s Arduinem UNO navazují další menší rozšiřující sady tématicky zaměřené na určitou oblast využití nebo aplikaci daných technických prostředků. Pořizujete-li si rozšiřující sadu, věnujte prosím pozornost tomu, na co navazuje a jaké předchozí sady předpokládá, v některých případech mohou i rozšiřující sady navazovat na sebe, nejen na sadu základní.

Seznam dílů sady

č.	Název dílu	Počet ks
1	Základní deska s namontovaným Arduinem Uno a kontaktním polem	1
2	USB kabel (A-B)	1
3	Propojovací kablíky 5 cm (6x bílý, 4x žlutý, 6x červený, 12x modrý)	28
4	Propojovací kablíky 10 cm (8x bílý, 4x žlutý, 2x červený, 6xmodrý)	20
5	Propojovací kablíky 15 cm (6x bílý, 2x žlutý, 2x červený, 2x modrý)	12
6	LED červená (3 mm)	8
7	LED žlutá (3 mm)	2
8	LED zelená (3 mm)	4
9	DIL switch dvojnásobný	1
10	Tlačítko 6 x 6 s dlouhými vývody	3
11	Trimr 5 k s drátovými vývody	3
12	Tranzistor N typ BC337	2
13	Rezistor 47 k	1
14	Rezistor 1 k	2
15	Rezistor 220 Ω	20
16	Rezistor 10 k	2
17	Kondenzátor elyt 470 µF/6 low ESR	1
18	Rezistor 2k2	1
19	Kondenzátor elyt 10 µF/10 V	1
20	Kondenzátor keramický 100 nF	1
21	Stejnosměrný motorek s kolečkem (šipkou) a vývody	1
22	Můstek pro motory L9110	1
23	Posuvný registr 74HC164 nebo 74HCT164	1
24	Časovač 555 (CMOS)	1
25	Piezomenič PT1540-P s vývody	1
26	Dioda 1N4007	1
27	Šroubováček pro nastavení trimrů	1
28	Servo mikro Vigor	1
29	Trojice zlacených kontaktů pto připojení serva	1
30	Cinch konektor žlutý – zásuvka s vývody	1
31	CD se soubory	1

Obsah