PRŮVODCE SVĚTEM

ARDUINA

ZBYŠEK VODA & TÝM HW KITCHEN

ARDUINO.CZ

HW KITCHEN

WWW.HWKITCHEN.CZ

Název: Průvodce světem Arduina
Autoři, autorská práva: Zbyšek Voda & tým HW Kitchen
Obálka: Autoři a Pavel Stříž
Sazba: Martin a Pavel Střížovi
Kontakt: martin@striz.cz, www.striz.cz
Papírové vydání knihy: www.hwkitchen.cz/arduino-kniha-pruvodce-svetem-arduina-2-vydani/

Obsah

Část I

Autoři úvodem

V průběhu roku 2014 postupně vznikal na serveru HW Kitchen (`www.hwkitchen.cz`), seriál článků o Arduinu. Postupně byly představeny základní dovednosti potřebné pro zvládnutí práce s ním. Seriál se také podrobně věnoval některým ze shieldů pro Arduino. Tato publikace původně obsahovala osmnáct dílů tohoto seriálu. Od té doby text prošel spoustou drobných úprav. Toto je druhé velké vydání, jehož cílem bylo odstranit co nejvíce chyb a také zpřehlednit strukturu knihy. Dlouho nám ale v hlavě ležela myšlenka, že je škoda mít knihu pouze jako ebook. Proto jsme se rozhodli, že knihu vydáme i v tištěné podobě.

Jsme moc rádi, že nyní v této první české knize o Arduinu můžete listovat a užít si její knižní podobu. Věříme, že tato publikace pomůže ještě více přiblížit fenomén Arduina českým bastlířům, vývojářům, tvůrcům a také lidem, kteří chtějí tvořit interaktivní a zábavné projekty.

Tímto bychom chtěli poděkovat všem, kteří nám hlásili malé i velké chyby a tím pomohli tuto publikaci zlepšit.

Milý čtenáři. I přes všechnu naši snahu není vyloučeno, že se do textu vloudily chyby. Pokud na některou narazíš, napiš nám prosím na email `zbysek@arduino.cz`.

Část II

Seznámení s Arduinem

Když se v současné době začátečník podívá na trh s vývojovými platformami, může ho čekat nemilé překvapení. Existuje totiž celá řada více či méně vhodných desek a čipů, které výrobci nabízí. Počínaje samostatnými čipy (např. PICAXE), k jejichž programování stačí pouze sériový kabel, a výkonnými platformami (Raspberry Pi, Arduino Yún) s možností běhu přizpůsobeného operačního systému konče. Ve světě asi nejrozšířenější platformou je Arduino. To nabízí různé typy desek od méně výkonných a malých modelů po kompletní soustavy obsahující USB, HDMI, Ethernet či audio porty. V této části si některé z desek představíme a povíme si, co dovedou.

Kapitola 1
O Arduinu

Vývoj prvního Arduina započal v roce 2005, když se lidé z italského Interaction Design Institute ve městě Ivrea rozhodli vytvořit jednoduchý a levný vývojový set pro studenty, kteří si nechtěli nebo nemohli pořídit tehdy velmi drahé desky, jako například BASIC Stamp. Mezi studenty se Arduino uchytilo, a tak se tvůrci rozhodli poskytnout ho celému světu. (V roce 2010 vznikl zajímavý dokument o vzniku Arduina s rozhovory s jeho tvůrci: Arduino The Documentary (2010) English HD[1].) Velké rozšíření však nebylo způsobeno prodejem desek, ale hlavně sdílením všech schémat a návodů celému světu (jedná se o Open Source[2] projekt).

Programová část Arduina byla založena na Processing[3], což je knihovna pro jazyk Java, ke které je přidán i vlastní editor. Vše má za cíl zjednodušit výuku programování. V dnešní době se prodalo již několik stovek tisíc desek Arduino. Důkazem, že tato platforma není mrtvá, může být i to, že již vzniklo několik desek ve spolupráci s velkými společnostmi, například Intelem. Od roku 2005 již bylo vytvořeno spoustu různých typů Arduina. Jelikož se jedná o Open Source projekt, vznikalo společně s hlavní linií projektu i mnoho dalších, neoficiálních typů, takzvaných klonů. Nejdříve si ale představíme oficiální desky.

Obrázek 1.1: Oficiální logo platformy Arduino [1]

Kapitola 2
Typy desek

Srdcem většiny Arduin je procesor od firmy Atmel, který je obklopen dalšími elektronickými komponenty. Pro celou řadu desek je typické jednotné grafické zpracování s převažující modrou barvou. V eshopech i na oficiálních stránkách Arduina `www.arduino.cc` se můžeme setkat s deskami, které mají za svým názvem ještě přidáno například Rev3 nebo R3. Jedná se o číslo verze dané desky. Mezi jednotlivými verzemi se mohlo například mírně změnit rozložení součástek nebo design. Nejedná se však o velké změny, které by si vyžádaly vznik nové desky. Na většině desek je mimo hlavního čipu ještě převodník, který umožňuje komunikaci mezi PC (USB) a čipem. Setkáme se však i s typy, které převodník nemají. Může to být ze dvou důvodů. Prvním z nich je úspora místa a následná nutnost použití externího převodníku. Druhým typem jsou ty desky, jejichž čip má v sobě tento převodník zabudovaný.

Nyní si předvedeme jednotlivé desky, které jsou pro přehlednost seřazeny od těch nejmenších po největší.

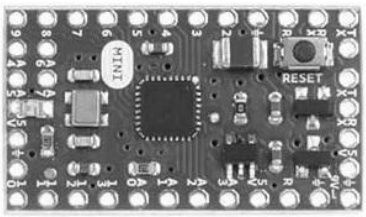
2.1 Arduino Mini

Arduino Mini je asi nejmenší oficiální verze Arduina, navržená pro úsporu místa. Daní za malé rozměry je však absence USB portu. K programování je tedy nutné použít externí USB-Serial převodník. Jeho výkon však nijak nezaostává za většími deskami. Běží na procesoru ATmega328 (dříve ATmega168) s taktem 16 MHz. Pro své malé rozměry je vhodný k použití například v chytrých vypínačích, dálkových ovladačích, či dnes velmi populárních zařízeních, připojených k internetu věcí.

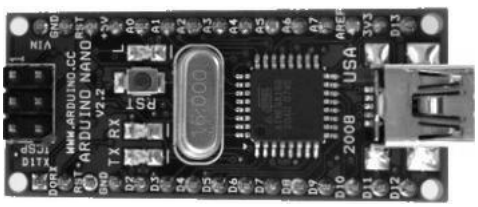
2.2 Arduino Nano

Arduino Nano se od svého menšího sourozence výbavou moc neliší. Největším rozdílem je zde však přítomnost USB portu a převodníku, kvůli němuž je celkové provedení o něco větší. Odpadá ale nutnost mít společně s deskou ještě externí USB-Serial převodník.

[1] Arduino The Documentary (2010) English HD – `http://vimeo.com/18539129`
[2] Open Source – `https://cs.wikipedia.org/wiki/Otevřený_software`
[3] Processing – `https://processing.org/`

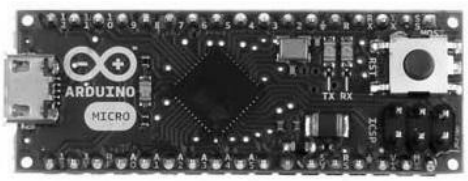


Obrázek 2.1: Arduino Mini [9]



Obrázek 2.2: Arduino Nano [10]

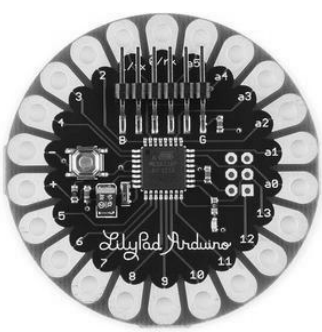
2.3 Arduino Micro



Obrázek 2.3: Arduino Micro [8]

Arduino Micro je jedna z desek, která má čip obsahující převodník. Tímto čipem je ATmega32u4. Jeho výhodou je, že se může pro počítač tvářit jako myš, nebo klávesnice a posílat příkazy, jako jsou stisk klávesy či posunutí myši. To je sice možné i s ostatními deskami, ale tato operace vyžaduje přeprogramování převodníku (nejčastěji založeném na čipu ATmega16u2, nebo ATmega8u2), což nemusí být úplně jednoduché. S touto deskou je tedy velice jednoduché vytvořit si vlastní klávesnici nebo herní ovladač.

2.4 LilyPad Arduino

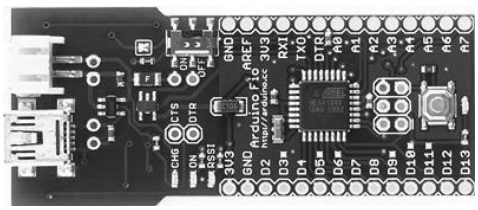


Obrázek 2.4: Arduino Lilypad [28]

Již při prvním pohledu je jasné, že Lilypad Arduino není úplně typická deska z rodiny Arduino. Jedná se totiž o verzi přizpůsobenou pro použití jako mozek různých wearable zařízení, tedy k tomu, že si LilyPad

přišijete na oblečení vodivou nití. Tak se dá vyrobit například cyklistická mikina s přišitými blinkry nebo například svítící obleček pro psa. Existuje více druhů této desky. Můžeme se setkat s verzí s USB a čipem ATmega32u4 nebo bez USB ve verzi ATmega328 a dalšími.

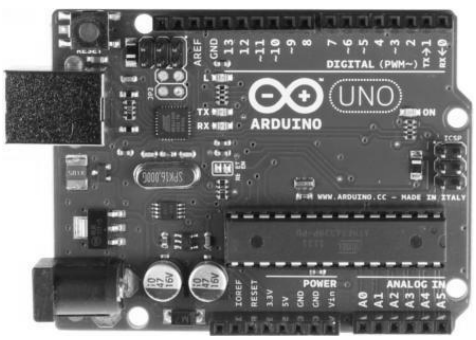
2.5 Arduino Fio



Obrázek 2.5: Arduino Fio [4]

Tato deska je přizpůsobená k připojení různých bezdrátových modulů (XBee moduly). Srdcem je procesor ATmega328P, který běží na frekvenci 8 MHz. Napětí je zde kvůli kompatibilitě s moduly sníženo oproti velké části ostatních desek z 5 V na 3,3 V.

2.6 Arduino Uno

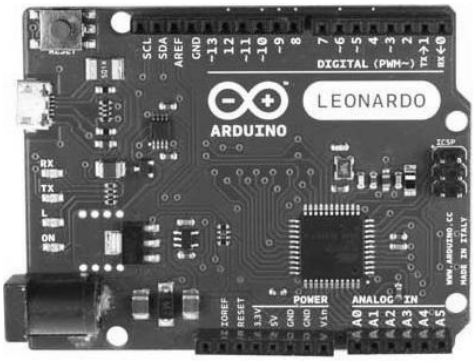


Obrázek 2.6: Arduino Uno [14]

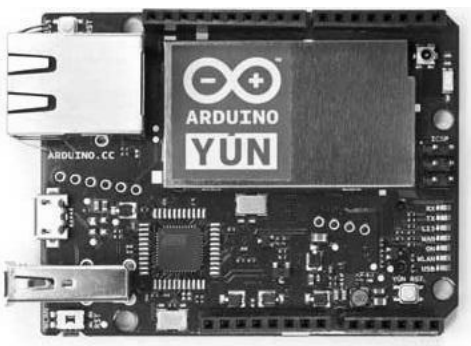
Arduino Uno je v současné době asi nejčastěji používaný typ desky. Je přímým pokračovatelem hlavní vývojové linie, která započala prvním Arduinem se sériovým portem místo USB, pokračující přes Arduino Extreme, NG, Diecimila a Duemilanove až k dnešnímu Uno. Na desce najdeme procesor ATmega328 a již klasické USB. Z této hlavní linie se vyvinuly i další dvě speciální desky. První z nich je Arduino Ethernet, které má stejnou výbavu jako Uno, místo USB portu zde ale najdeme Ethernet port pro připojení k síti. Příjemná je přítomnost slotu pro microSD karty. Druhou deskou je Arduino Bluetooth. Jak už název napovídá, místo USB zde najdeme bluetooth modul pro bezdrátovou komunikaci. Velmi odlehčenou verzí Arduina Uno je Arduino Pro. To postrádá USB port a je tedy nutné ho programovat externím převodníkem. Je určeno spíše k pevnému zabudování do nějaké konkrétní aplikace než k běžnému programování.

2.7 Arduino Leonardo

Arduino Leonardo designově navazuje na Arduino Uno. Liší se však použitým čipem. Tím je ATmega32u4, který byl popsán již u Arduino Micro. Podrobně se mu věnujeme v části Arduino jako klávesnice a myš na straně 73.



Obrázek 2.7: Arduino Leonardo [6]



Obrázek 2.8: Arduino Yún [15]

2.8 Arduino Yún

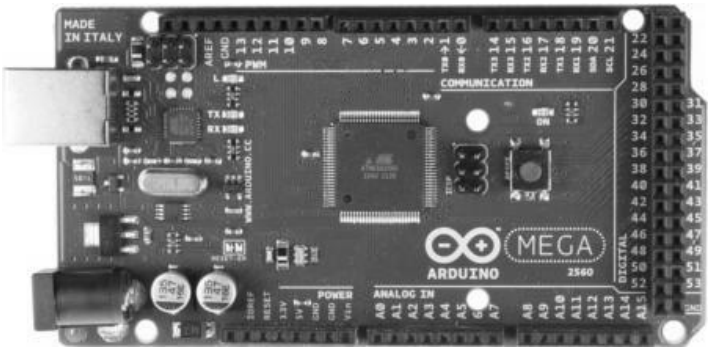
Model Arduino Yún sice také designově navazuje na Arduino Uno, jedná se však o naprostého průkopníka. Mimo již zmíněného čipu ATmega32u4, na kterém běží jádro Arduina, zde totiž najdeme i čip Atheros AR9331, který je schopný běhu odlehčeného operačního systému Linuxu OpenWrt-Yun. Ve výbavě je softwarový bridge (prostředník, most), který zajišťuje komunikaci mezi oběma čipy. V kompaktním obalu tedy získáme v porovnání s velikostí velmi výkonný stroj. Na desce najdeme mimo microUSB pro programování čipu ATmeaga32u4 i normální USB pro potřeby Linuxu a Ethernet port pro připojení k síti. Můžeme tedy například posílat naměřené hodnoty přímo na webový server. Příjemná je také přítomnost WiFi modulu přímo na desce.

2.9 Arduino Mega2560

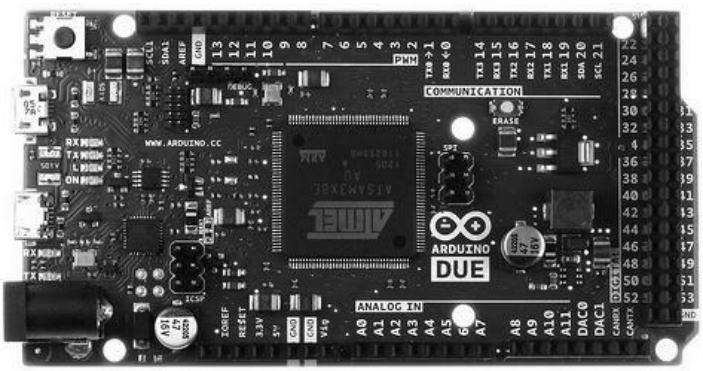
S Arduino Mega2560 se dostáváme do skupiny desek, jejichž vzhled vznikl prodloužením designu Arduina Uno. Zvětšení rozměrů přináší prostor pro větší a výkonnější čipy a také více pinů (zdířek). Předchozí verzí bylo Arduino Mega1280. Hodí se tam, kde je zapotřebí většího výpočetního výkonu. Zajímavou odnoží této desky je Arduino Mega ADK vybavené jedním USB navíc pro připojení zařízení s Androidem přes ADB. To je však v dnešní době ne zcela užitečné vzhledem k tomu, že Android 4.2+ přímo podporuje připojení Arduina tak, jak jej připojujeme k PC.

2.10 Arduino Due

Arduino Due je pokračovatelem Arduina Mega, avšak s tím rozdílem, že běží na daleko výkonnějším čipu. Je jím Atmel SAM3X8E, který tiká na taktovací frekvenci 84 MHz a jeho jádro je 32bitové, což je oproti ostatním deskám s 8bity a maximálně 16 MHz opravdu velký skok. Na desce nalezneme dva microUSB konektory. Jeden pro programování čipu, druhý pro připojení zařízení, jako jsou myši, klávesnice, telefony a jiné.

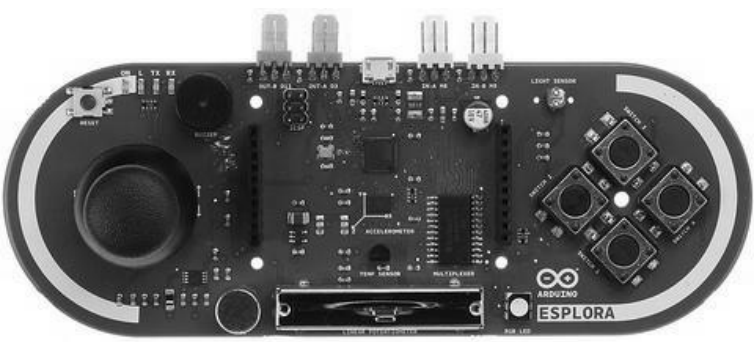


Obrázek 2.9: Arduino Mega2560 [7]



Obrázek 2.10: Arduino Due [2]

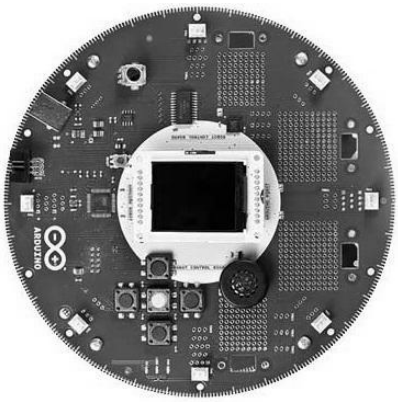
2.11 Arduino Esplora



Obrázek 2.11: Arduino Esplora [3]

Arduino Esplora je první z desek, které by se daly zařadit do kategorie „hybridní“. Na první pohled je viditelný joystick, tlačítka a posuvný potenciometr. Nalezneme zde ale také piezzo bzučák, teploměr, tříosý akcelerometr nebo piny pro připojení LCD displeje. To je velmi výhodné, protože u jiných Arduin bychom tyto komponenty museli připojit dodatečně. Jedná se totiž o typ Arduina, se kterým se dá vytvořit samostatný herní set nebo vlastní konzole pro ovládání her. Jednoduchou komunikaci s PC zajišťuje procesor ATmega32u4. Arduinu Esplora se více věnujeme v kapitole Arduino jako klávesnice a myš na straně 73.

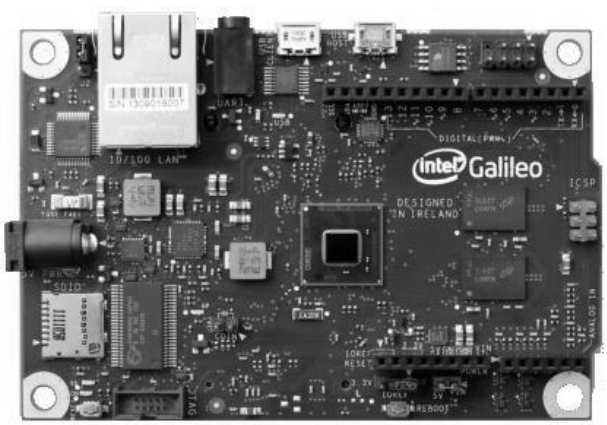
2.12 Arduino Robot



Obrázek 2.12: Arduino Robot [12]

Jak už název napovídá, jedná se o set pro vytvoření vlastního chytrého robota. Jeho mozkem je procesor ATmega32u4. Zajímavostí je přítomnost kompasu.

2.13 Arduino Intel Galileo

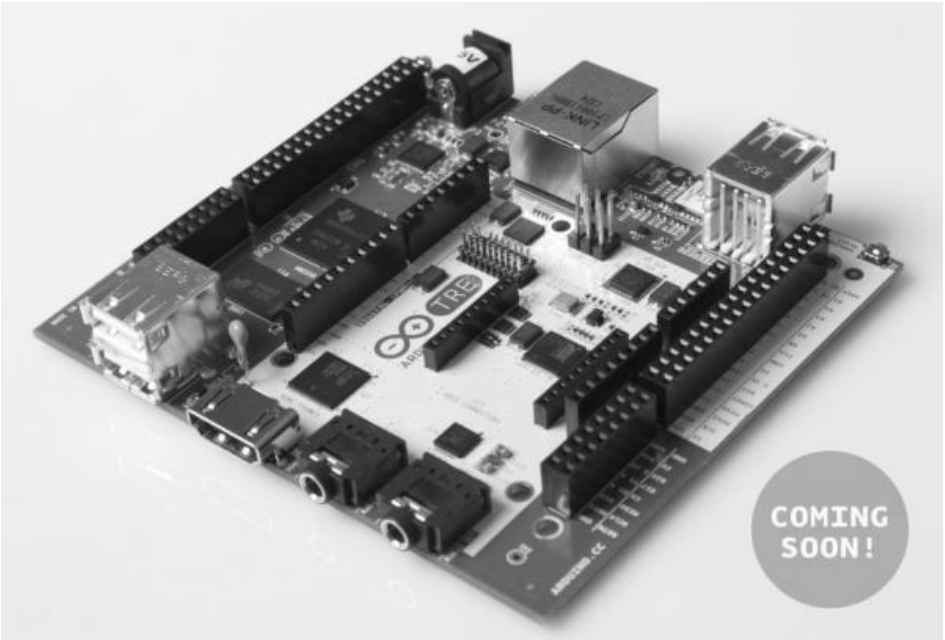


Obrázek 2.13: Arduino Intel Galileo [5]

Tato verze vznikla ve spolupráci se společností Intel. Jedná se o první desku, která běží na čipu Intel® Quark SoC X1000, což je 32bitový procesor s frekvencí 400 MHz. Najdeme zde dvě USB, microSD slot i Ethernet port. Užitečná může být také přítomnost mini-PCI Express slotu, pro připojení různých přídavných karet.

2.14 Arduino Tre

U Arduina Tre je situace nejistá. Již od roku 2014 je totiž v plánu, ale nic nenasvědčuje tomu, že by mělo skutečně přijít. Víme o něm to, že by se mělo jednat o zatím nejvýkonnější typ. Mělo by obsahovat 1GHz procesor schopný běhu velmi náročných výpočetních aplikací. Stejně jako Arduino Yún bude obsahovat dva procesory. Jeden pro jádro Arduina a druhý pro Linux. Na desce také nalezneme HDMI port, dva audio konektory, jeden USB port pro programování a 4 USB porty pro připojení dalších zařízení k linuxu. Už z hardwarové výbavy je patrné, že bude moci Arduino Tre konkurovat i jiným menším počítačům jako je například Raspberry Pi. Původně výrobce sliboval dostupnost na jaře 2014.

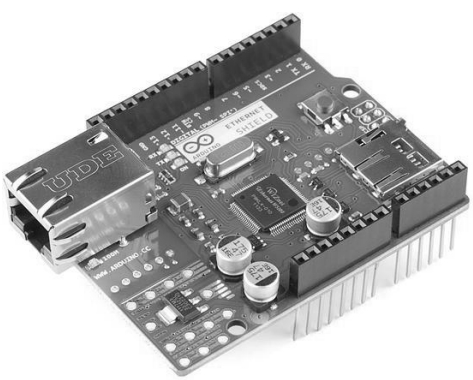


Obrázek 2.14: Arduino Tre [13]

Kapitola 3
Arduino shieldy

Když se chceme na běžném stolním počítači připojit k WiFi, většinou nemáme jinou možnost, než si dokoupit WiFi kartu. Když chceme poslouchat nebo nahrávat dobrou hudbu, musíme připojit kvalitní zvukovou kartu, hráči počítačových her se neobejdou bez kvalitní grafické karty... A stejné to je u Arduina. Když něco nezvládne, nemusí být ještě všemu konec. Stačí si vybrat z rozsáhlé nabídky tzv. shieldů a vybraný shield poté připojit k Arduinu. Stejně jako desek existuje i celá řada shieldů. Z těch oficiálních jsou to Ethernet shield, WiFi shield, Motor shield a další. Při výběru je však nutné dát si pozor na to, aby byl vybraný shield s vaším Arduinem kompatibilní.

Na obrázku vidíte, jak vypadá Ethernet shield.



Obrázek 3.1: Arduino Ethernet shield [24]

Kapitola 4
Arduino klony

Jak už jsme naznačili dříve, společně s oficiální řadou existuje ještě spousta dalších, neoficiálních desek. Jedná se o takzvané klony. Poznáme je podle toho, že mají často v názvu -duino (název Arduino je chráněný autorskými právy, -duino a podobné části jsou v názvu přípustné). Jelikož jsou všechna schémata, rozkresy součástek i software dostupné online zdarma, může si prakticky každý sestavit své Arduino takřka „na koleni“. Můžeme se tedy setkat s klony tvarově a výbavou totožnými s oficiálními modely. Není to však pravidlem. Často jsou k vidění i desky, které jsou uzpůsobené ke konkrétní činnosti. Příklady klonů jsou:

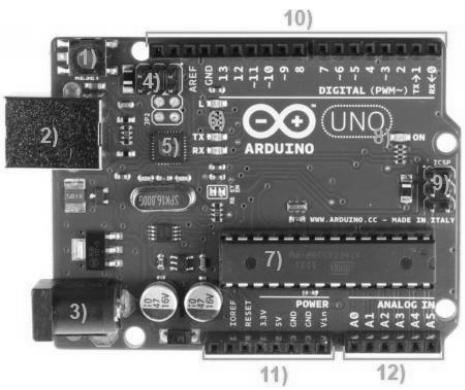
- ArduPilot – navržený pro ovládání autonomních létajících zařízení (letadla, kvadrokoptéry...).
- Freaduino, Seeeduino – o něco levnější kopie originálních desek.
- Rainbowduino – připravené k nasazení a řízení maticového RGB LED displeje, je možné je sestavovat do větších celků. O něm se můžete více dočíst na straně 126.
- A další..., viz `http://playground.arduino.cc/main/similarBoards`.

Část III

Programujeme Arduino

Kapitola 5
Výběr a seznámení

Před tím, než začneme s Arduinem pracovat, si musíme nějaký vhodný model vybrat. Pokud jsou cílem co nejmenší výrobky, bude nejlepší sáhnout po nějaké z menších desek (Mini, Nano, Micro). Pokud však hledáte velmi výkonné modely, které zastanou méně výkonné PC, vyberte spíše Due, Yún či Intel Galileo. Zlatou střední cestou jsou modely Uno a Mega 2560. Pro jednoduché zkoušení zapojení a nenáročných modulů je Uno plně dostačující. Na něj je také designována většina shieldů. Arduino Mega 2560 na druhou stranu nabídne daleko více vstupních a výstupních pinů. Vyberme si tedy pro účel následujícího popisu model Arduino Uno. Dále uvedené části najdeme v různých obměnách na většině desek.



Obrázek 5.1: Co nalezneme na desce Arduino [14]

1) Pod číslem jedna se skrývá resetovací tlačítko. To použijeme, pokud chceme náš program spustit znovu od začátku. U různých typů Arduina se můžeme setkat i s jiným umístěním tohoto tlačítka. Většinou je ale toto tlačítko popsané nápisem RESET.

2) USB konektor typu B. U nejstaršího modelu najdeme místo USB sériový port. U některých novějších modelů se setkáme s micro USB. Na některých deskách ho vůbec nenajdeme, protože mají buď jiný způsob připojení (Ethernet, Bluetooth), nebo pro programování vyžadují připojení externího programátoru.

3) Napájecí konektor. Využijeme ho, pokud nebudeme Arduino napájet z USB.

4) ICSP hlavice pro externí programování USB-serial převodníku. Běžný uživatel ji nepoužije. U verzí bez převodníku nebo s převodníkem obsaženým v hlavním čipu ji samozřejmě nenajdeme.

5) USB-serial převodník. Ten se stará o komunikaci mezi hlavním čipem a PC. Plní zde tedy roli jakéhosi tlumočníka. U verzí bez převodníku nebo s převodníkem obsaženým v hlavním čipu ho nenajdeme.

6) Indikační LED diody L, RX a TX. Dioda s popisem L je často využívaná. Je totiž připojená k výstupu číslo 13. S ní se tedy dá vyzkoušet blikání i bez připojené externí LEDky. Některá Arduina ji však vůbec neobsahují. Diody s popisem TX a RX blikají, pokud probíhá komunikace přes sériovou linku.

7) Hlavní čip celé desky. V různých podobách a typech ho najdeme na všech deskách.

8) Indikační LED dioda ON. Svítí, když je připojené napájení.

9) ICSP hlavice pro externí programování hlavního čipu. Využívají ji některé shieldy.

10) Digitální piny. Do těchto zdířek budeme připojovat všemožné obvody. Vývody označené vlnovkou podporují PWM modulaci, o které si povíme později.

11) Převážně napájecí výstupy Arduina.

12) Analogové vstupy. Sem připojíme vodiče, na kterých budeme chtít měřit nějakou analogovou hodnotu. Dají se využít i jako digitální vstupy a výstupy.

Kapitola 6
Arduino IDE

6.1 Historie

Arduino IDE (anglicky *integrated development enviroment*, integrované vývojové prostředí) je napsané v jazyce Java. Jedná se o software vzniklý z výukového prostředí Processing. To bylo mírně upraveno, byly přidané určité funkce a v neposlední řadě podpora jazyka Wiring.

6.2 Stažení a instalace

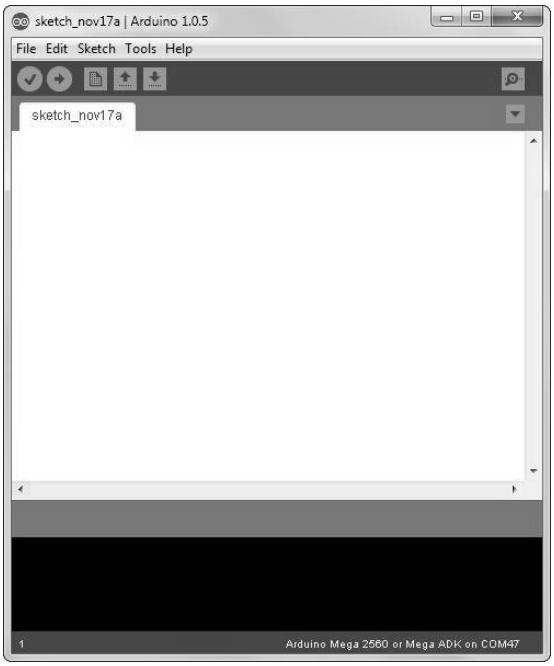
Arduino IDE si můžeme stáhnout ze dvou zdrojů. Prvním z nich je Arduino.cc[4], druhým Arduino.org[5]. Dnes totiž existují dva paralelní weby, které nabízejí vlastní Arduino IDE. Na webu `http://arduino.org` také nalezneme alternativní Arduino Studio. Ať už si vybereme IDE od `http://arduino.cc` nebo `http://arduino.org`, stáhněte si poslední verzi pro OS, který používáte.

Najdeme si tedy poslední finální verzi a stáhneme ji pro požadovaný operační systém. Pro Windows je nejjednodušší stáhnout si ZIP archiv, který je po rozbalení plně funkční. Pro Linux se instalace může lišit i podle distribucí. Popis pro jednotlivé distribuce nalezneme zde[6]. Musíme také zmínit další velký operační systém, jímž je Mac OS. Návod na instalaci nalezneme zde[7]. Následující část kapitoly se bude věnovat použití Arduina s operačním systémem Windows. Předpokládejme, že máme stažený ZIP archiv z Arduino.cc. Vybereme si složku, ve které chceme mít software pro Arduino a zde archiv rozbalíme. Po rozbalení obsahuje další složky a soubory. Složka `Drivers` obsahuje ovladače pro komunikaci Arduina s PC. V `Examples` nalezneme příklady kódů. Důležitou složkou je složka `Libraries`, kam se ukládají knihovny. To jsou balíky obsahující rozšiřující funkce pro programování. Ještě než si IDE spustíme, si vytvoříme v uživatelské složce `Dokumenty` složku `Arduino`. Tuto složku většinou nalezneme na umístění: `C:\Users\jmeno_uzivatele\Documents`. Zde si vytvoříme složku `Arduino` a v ní složku `Libraries`. Sem si budeme ukládat vytvořené programy a do složky `Libraries` přidané knihovny. Ty nám zde zůstanou stále stejné, i když přejdeme na novější verzi Arduino IDE. Nyní už nás bude zajímat pouze soubor `arduino.exe` ve staženém balíku, který spustí vývojové prostředí. Spusťme si ho tedy a ukažme si jeho nejdůležitější funkce.

6.3 Používání

Vývojové prostředí můžete vidět na obrázku č. 6.1. V prvním řádku navigačních prvků nás bude zajímat pouze rozbalovací nabídka *Tools*, ve které nalezneme nastavení pro připojení a programování desek. Její funkce si popíšeme později. V dalším řádku nalezneme několik ikon. Jako první zleva nalezneme ikonu s fajfkou – *Verify*. Ta po kliknutí spustí kontrolu programu a zkompiluje kód. Pokud nalezne nějakou chybu, v programu ji zvýrazní. Vedle nalezneme kulatou ikonu s šipkou – *Upload*. Ta spustí kontrolu programu, a pokud nenalezne žádné chyby, nahraje program do připojeného Arduina. Další je ikona se symbolem přeložené stránky – *New*, která po kliknutí vytvoří nový soubor. Další tlačítko s šipkou nahoru – *Open* – otevře nabídku pro otevření programů (včetně těch, které máme uložené ve složce `Dokumenty`). Tlačítko s šipkou dolů – *Save* – uloží současný program. Ve stejném řádku nalezneme úplně vpravo ještě ikonu s lupou – *Serial Monitor*. Ta spustí sériový monitor, o kterém si více povíme dále. Velký bílý prostor slouží k zápisu kódu a černý prostor dole zobrazuje informační a chybové výpisy z běhu prostředí a kompilátoru.

[4] Arduino.cc – `https://www.arduino.cc/en/Main/Software`
[5] Arduino.org – `http://www.arduino.org/software`
[6] Instalace IDE pro Linux – `http://playground.arduino.cc/Learning/Linux`
[7] Instalace IDE pro Mac OS X – `http://arduino.cc/en/Guide/MacOSX`



Obrázek 6.1: Arduino IDE

6.4 Programovací jazyk

Arduino je možné programovat v jazyce C nebo C++. Nejjednodušší je však používat C++ knihovnu nazvanou Wiring. Ta je v současné době pro programování Arduina velmi rozšířená. Kvůli její komplexnosti se o ní občas mluví jako o samostatném programovacím jazyku. My budeme v knize označení *Programovací jazyk Wiring* také používat. Pro první seznámení si otevřeme příklad *BareMinimum*. Klikneme na ikonu *Open* a v rozbalovacím seznamu *01.Basics* vybereme možnost *BareMinimum*. V editoru se nám zobrazí následující kód:

```
1 void setup() {
2     // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7     // put your main code here, to run repeatedly:
8
9 }
```

Na ukázkovém kódu si můžeme všimnout hned dvou věcí. První z nich je přítomnost dvou bloků programu. V horní části nalezneme blok (funkci) `void setup(){}`. Mezi složené závorky se v tomto bloku píše kód, který se provede pouze jednou na začátku programu. To znamená buď po připojení napájení, zmáčknutí tlačítka RESET nebo nahrání kódu do Arduina. Druhým blokem (funkcí) je `void loop(){}`, do jehož složených závorek se zapisuje kód, který se bude opakovat neustále dokola až do odpojení napájení, restartu nebo nahrání nového programu. Tyto dvě části musí být v programu VŽDY – tedy i když neobsahují žádné příkazy. Při jejich absenci by program skončil chybou. Dále bychom si měli všimnout dvojitého lomítka. To nám značí komentáře v programu. Část kódu nebo textu zapsanou za dvojitým lomítkem bude program ignorovat. Používá se, když si chceme k části kódu zapsat poznámku nebo když chceme na chvíli vyřadit část kódu z provozu. Můžeme se setkat s dvěma typy komentářů:

Jednořádkový komentář:

```
1 //kdyz radek zacina dvojitym lomitkem, jedna se o jednoradkovy komentar
2 //vse, co je v radku za nim program ignoruje
3 to co se vsak nachazi na dalsim radku je brano jako normalni kod
```

Víceřádkový komentář:

```
1 /* ty zacinaji lomitkem a hvezdickou
2 mohou
3 mit
4 libovolny
5 pocet
6 radku
7 a konci hvezdickou a lomitkem */
```

Kapitola 7
Blikáme LED

7.1 Budeme potřebovat

1. Arduino Uno (nebo jinou desku).
2. USB kabel.
3. Nepájivé kontaktní pole.
4. LED dioda – stačí obyčejná za 1 Kč.
5. Vodiče – dobrou kombinací obsahující nepájivé pole i vodiče je například tento set[8].
6. 330 Ω rezistor – zapojení sice bude fungovat i bez něj, ale je nutné ho použít.

7.2 Připojení Arduina a PC

Arduino Uno připojíme pomocí USB k počítači. Mělo by se nám objevit instalační okno. Pokud se instalace nového zařízení nepovede, neděste se. Nyní přichází na řadu složka *Drivers*, stažená v balíčku Arduino IDE. Ve Windows v nabídce *Start* → *Ovládací panely* otevřeme *Správce zařízení*. Zde najdeme naše Arduino. Pravým tlačítkem otevřeme okno *Vlastnosti* a na kartě *Ovladač* zmáčkneme tlačítko *Aktualizovat ovladač* (k tomu je však nutné mít oprávnění správce). Poté vybereme možnost *Vyhledat ovladač v počítači* a navedeme instalační program do umístění složky `Drivers`. Nakonec zmáčknutím tlačítka *Další* nainstalujeme ovladač.

Nyní ještě musíme nastavit Arduino IDE. Otevřeme nabídku *Tools* a v seznamu *Boards* vybereme Arduino Uno. Poté v *Tools* → *Serial Port* vyberte sériový port, na který je Arduino připojeno (většinou je to jediný port v seznamu). Tímto je za námi nastavování a můžeme se pustit do zapojování.

Pokud máte v seznamu Ports *více portů, asi nejjednodušší způsob, jak zjistit, na jakém portu je Arduino přopojeno, je ho odpojit a sledovat, který port zmizel.*

7.3 Zapojení

Vezmeme si LED diodu, vodiče a rezistor a vše zapojíme tak, jak je vidět na obrázku č. 7.1.

7.4 Program

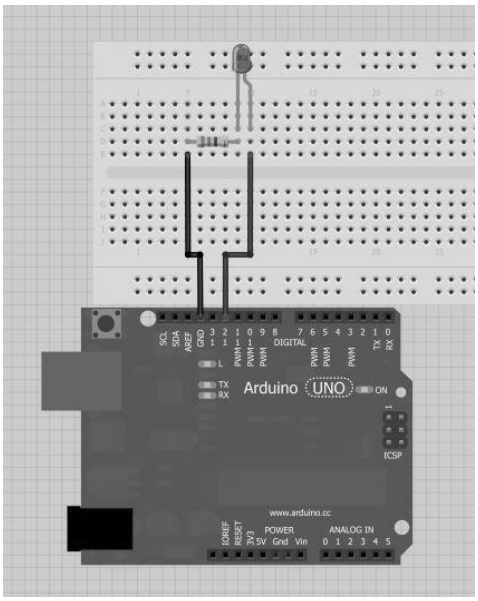
Nyní už nám nezbývá nic jiného, než do editoru vložit následující program a stiskem tlačítka *Upload* nahrát kód do Arduina. Pokud vše proběhlo v pořádku, LED dioda začne blikat.

```
1 void setup() {
2     pinMode(12,OUTPUT); //nastav pin 12 jako vystup
3 }
4
5 void loop() {
```

[8] Set nepájivého kontaktního pole a vodičů – `http://www.hwkitchen.com/products/breadboard-with-wire-kit/`

```
6       digitalWrite(12,HIGH); //na pinu 12 pust proud
7       delay(1000); //pockej 1000 ms = 1 s
8       digitalWrite(12,LOW); //na pinu 12 vypni proud
9       delay(1000);
10  }
```

Vidíme, že budeme čekat dvakrát jednu vteřinu. Celý cyklus bude tedy přibližně 2 s dlouhý.



Obrázek 7.1: Schéma zapojení příkladu `Blink`

Část IV

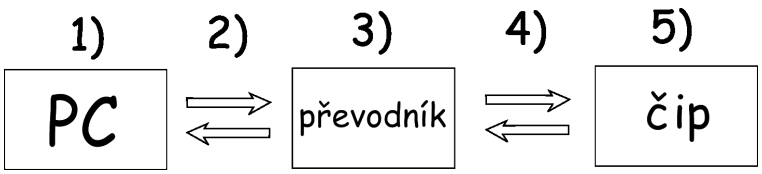
Základní struktury jazyka Wiring

V předchozí části jsme si ukázali první program, ve kterém Arduino blikalo LED. Úvodní seznámení je tedy za námi a můžeme se pustit do dalšího programování. V této části se podíváme na základní náležitosti jazyka Wiring. Na začátek si vysvětlíme, jak Arduino komunikuje s PC. Poté si řekneme, jak používat proměnné a jak pracovat se vstupy a výstupy Arduina.

Kapitola 8
Princip komunikace s PC

Aby mohlo Arduino správně komunikovat s PC, musí mít několik základních součástí. Následující popis postihuje většinu desek Arduino. Najdou se však i speciální desky, které podporují jiný způsob programování (Bluetooth, Ethernet, WiFi...), těmi se zde ale dále zabývat nebudeme. Popišme si tedy proces programování, se kterým se setkáme u většiny desek.

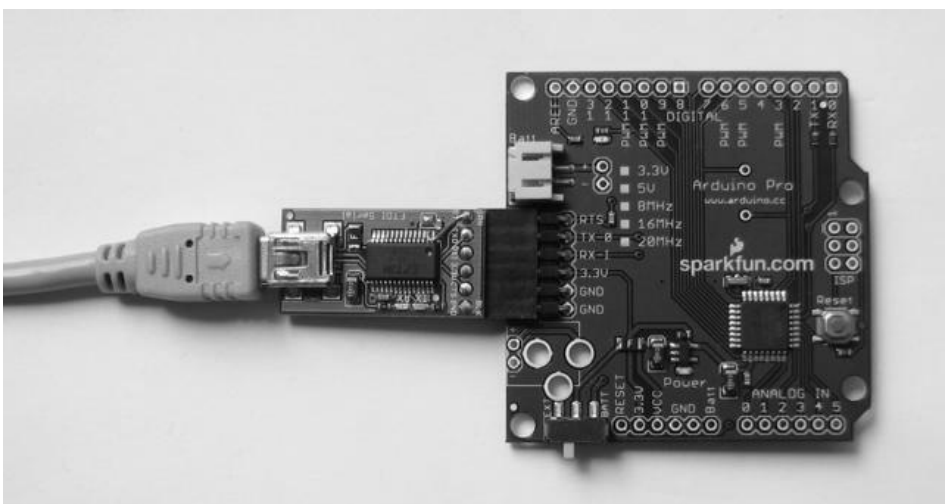
Co je ke komunikaci potřeba:



Obrázek 8.1: Schéma sériové komunikace

1) Základním předpokladem je mít PC s USB portem.
2) USB kabel.
3) S převodníkem se nám schéma trochu komplikuje. Můžeme se totiž setkat se třemi základními typy převodníku. Všechny tři převodníky fungují stejně. Liší se pouze způsobem připojení.
   (a) Převodník, který je napevno připájený k základní desce Arduina.
   (b) Převodník, který mají některé čipy (ATmega32u4...) přímo v sobě.
   (c) Externí převodník, který musíme při programování k Arduinu připojit.
4) Připojení převodníku a čipu. Při použití externího převodníku se většinou jedná o šest vodičů, které vystupují z desky. U zbylých dvou typů převodníku jsou to pouze kontakty na plošném spoji nebo propojení uvnitř čipu.
5) Mozek, který přijímá přeložené instrukce od převodníku.

Obrázek č.8.2 ukazuje Arduino Pro s připojeným externím převodníkem.



Obrázek 8.2: Arduino Pro s externím převodníkem [11]

Sériová komunikace se ale dá využít k více věcem než jen k programování. Pomocí ní totiž můžeme komunikovat s Arduinem, i když už na něm běží náš program. Poté můžeme například číst hodnoty ze senzorů a posílat je do PC nebo ovládat Arduino jednoduchými textovými příkazy. Používání těchto funkcí si popíšeme za chvíli, kdy už budeme mít dostatek informací k jejich pochopení.

Kapitola 9
Proměnné

Proměnná je pojmenované místo vyhrazené v paměti, do kterého se dají ukládat data. Každá proměnná má vlastní jméno, typ dat, které uchovává, a hodnotu. Často se používají například tam, kde se v programu dokola opakují ty samé hodnoty. Praktické využití proměnných si ukážeme na následujícím programu (psáno v pseudokódu):

Představme si, že máme několik očíslovaných světel. Vždy si vybereme jedno, se kterým budeme blikat.

```
1  zapniSvetlo(10); //zapni desate svetlo
2  vypniSvetlo(10); //vypni desate svetlo
3  zapniSvetlo(10);
4  vypniSvetlo(10);
5  zapniSvetlo(10);
6  vypniSvetlo(10);
7  ...
```

Nyní si stejný program přepíšeme s užitím proměnných.

```
1  cislo A = 10;
2   //promenna se jmenuje A, je datoveho typu cislo a ma hodnotu 10
3
4  zapniSvetlo(A);
5  vypniSvetlo(A);
6  zapniSvetlo(A);
7  vypniSvetlo(A);
8  zapniSvetlo(A);
9  vypniSvetlo(A);
10 ...
```

Kdybychom chtěli změnit světlo, se kterým blikáme, museli bychom v prvním případě změnit všechny čísla 10 na jiná. V případě druhém nám stačí přepsat pouze hodnotu proměnné a program ji už sám dosadí na všechna potřebná místa.

9.1 Práce s proměnnými

Nyní už se podíváme na to, jak pracovat s proměnnými v jazyce Wiring. Jedním z číselných datových typů je typ `integer` (zkracuje se na `int`). Ukažme si tedy, jak vytvořit proměnnou, která v sobě uchová číselnou hodnotu. Abychom mohli v programu s proměnnou pracovat, musíme ji nejprve deklarovat („vytvořit“). Poté jí můžeme přiřadit hodnotu.

```
1  //deklarace promenne x
2  int x;
3  //prirazeni hodnoty
4  x = 10;
5
6  //tyto dve operace se daji spojit do jedne
7  int y = 10;
```

Vytvořit proměnnou ale nemůžeme jen tak někde. Když budeme chtít používat danou proměnnou všude v programu, musíme ji vytvořit vně všech funkcí (tedy i mimo funkce `setup` a `loop`). Pokud nám stačí používat proměnnou uvnitř jedné funkce a nikde jinde ji nepotřebujeme, stačí, když ji deklarujeme uvnitř funkce. Výhodou v tomto případě je, že nám proměnná vytvořená uvnitř funkce nebude nikde jinde, než v těle funkce „překážet“.

```
int x = 10; //tuto promennou muzeme pouzit vsude

void setup() {
    int y = 11;
    //uvnitr teto funkce muzeme pouzit promenne x a y
}

void loop() {
    int z = 12;
    //zde muzeme pouzit promenne x a z
}
```

Pokud bychom se pokusili do Arduina nahrát kód, ve kterém používáme proměnnou `y` ve funkci `loop` (nebo `z` ve funkci `setup`), překlad kódu skončí s chybovou hláškou a do Arduina se nic nenahraje.

9.2 Datové typy

Jak už jsme naznačili dříve, každá proměnná má svůj datový typ. Ten nám říká, jaká data můžeme v proměnné najít. Může se jednat o logické hodnoty (`true`/`false`), znaky, nebo čísla. Pojďme si nyní představit základní typy.

9.2.1 Číselné datové typy

- `byte` – Proměnná datového typu byte má velikost 8 bitů a slouží k uchování celých čísel. Její rozsah je 2^8 hodnot – 0 až 255.
- `integer` – V programech se používá jen zkratka `int`. Slouží k ukládání celých čísel. Rozsah tohoto datového typu se liší podle použitého procesoru a zasahuje jak do kladných, tak i do záporných čísel. Nula leží přibližně v polovině rozsahu. U desek s procesory Atmega (tedy naprostá většina) uchovává 16bitovou hodnotu – tedy od $-32\,768$ do $32\,767$. U Arduino Due s procesorem SAM je tato hodnota 32bitová a může obsahovat čísla od $-2\,147\,483\,648$ do $2\,147\,483\,647$.
- `long` – Slouží k uchování celočíselných 32bitových hodnot od $-2\,147\,483\,648$ do $2\,147\,483\,647$.
- `float` – Tento datový typ je určený pro uchování čísel s desetinnou čárkou. V jazyce Wiring se však, stejně jako ve většině dalších programovacích jazyků, používá desetinná tečka. Jeho velikost je 32 bitů. Můžeme v něm ukládat hodnoty od $-3{,}402\,823\,5 \cdot 10^{38}$ do $3{,}402\,823\,5 \cdot 10^{38}$.

9.2.2 Logický datový typ

- `boolean` – Proměnné tohoto datového typu v sobě uchovávají pouze dvě hodnoty. Buďto `true` (pravda), nebo `false` (nepravda).

9.2.3 Znakový datový typ

- `char` – Tento datový typ slouží k uchování jednoho znaku textu. Znak je zde uchován jako jeho číselná hodnota v ASCII tabulce znaků. Písmena, slova i věty se píší v uvozovkách. K uchování řetězců textu slouží typ `string`, kterým se budeme zabývat za chvíli.

```
//byte
    byte a = 12;
//integer
    int b = 400;
//long
    long c = 12121212;
//float
    float d = 1.256;

//boolean
    boolean e = false;
```

```
12
13 //char
14     char f = 'A';
15     char f = 65; //v ASCII tabulce znaku ma A hodnotu 65
```

Existují i další číselné datové typy, ale ty se nepoužívají moc často. Jejich popis nalezneme v Arduino Reference[9] *v prostředním sloupci v části Data Types.*

Kapitola 10
Pole

Pole (anglicky *array*) je speciální typ proměnné. Umožňuje shromáždit více hodnot do jedné proměnné. Můžete si jej představit jako krabičku, která má jednotlivé přihrádky očíslované. Když víme, jakou krabičku používáme a do jaké přihrádky se chceme podívat, můžeme se dostat k požadované hodnotě. V programátorské terminologii se číslům přihrádek říká *index*.

10.1 Deklarace pole

Jejich deklarace je podobná jako u proměnných – každé pole má datový typ hodnot, které v něm najdeme, jméno, hodnoty a navíc i velikost pole.

```
1  //pole muzeme deklarovat nekolika zpusoby
2  int jmeno[6]; //deklarace pole s sesti bunkami, u takto deklarovaneho pole musi byt
       uvedena velikost
3  int jmeno[] = {2, 3, 4, 5}; //prvky v poli oddelujeme carkami
4  int jmeno[4] = {2, 3, 4, 5}; //v tomto pripade velikost pole uvest muzeme, ale
       nemusime
5
6  //zvlastnim typem pole je pole znaku (nazyvane retezec – string)
7  //umoznuje totiz specificky zpusob prirazeni hodnoty
8  char jmeno[15]; //deklarace retezce
9  char jmeno[] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
10 char jmeno[7] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
11 char jmeno[] = "arduino";
12 char jmeno[7] = "arduino";
```

10.2 Přístup k hodnotám v poli

Ve většině programovacích jazyků jsou indexy v poli číslovány od nuly. Prvek na prvním místě má tedy index 0. Čtení hodnoty prvku pole poté probíhá stejně jako u proměnných, jen musíme připojit ještě jeho index.

```
1 int a[] = {1,2,3,5,7,11,13,17}; //deklarace pole a
2
3 a[0]; //prvek s indexem 0 ma hodnotu 1
4 a[5]; //prvek s indexem 5 ma hodnotu 11
5 ...
```

Kapitola 11
Digitální vstup a výstup

Jelikož je Arduino určeno k dalšímu rozšiřování, obsahuje vstupy a výstupy (nazývané piny), ke kterým se dá vodičem připojit další obvody, čipy, relé, paměti atd. My už jsme se s takovýmto případem setkali v dřívější části, když jsme blikali LED diodou. K práci s těmito piny má Arduino k dispozici jednoduché funkce. Nejdříve ze všeho je však potřeba programu říci, jestli s pinem pracujeme jako se vstupem, nebo výstupem.

[9] Arduino Reference – `http://arduino.cc/en/Reference/HomePage`

11.1 Vstup, nebo výstup?

K nastavení pinu slouží funkce `pinMode()`. Ta pro svoji správnou činnost potřebuje dva vstupní parametry – `pinMode(cislo_pinu, INPUT/OUTPUT);`. Pokud chceme daný pin používat jako vstup, bude druhý parametr `INPUT`, pokud jako výstup, bude to `OUTPUT`. Číslo pinu je většinou natištěno na desce Arduina u příslušného pinu. Ve verzi Arduino Uno tedy můžeme používat piny 0 až 13. Tím to ale nekončí, protože můžeme k digitálním operacím používat i piny označené jako ANALOG IN, jenom místo samotného čísla musíme před číslo napsat A. U Arduina Uno jsou to tedy A0 až A5.

```
1 byte cislo = 13;
2
3 pinMode(cislo, OUTPUT); //nastaveni pinu 13 na vystup
4 pinMode(12, INPUT); //a pinu 12 na vstup
```

11.2 Ovládání výstupu

K ovládání výstupu se používá funkce `digitalWrite()`. S touto funkcí jsme se setkali již dříve, když jsme blikali LED diodou. Stejně jako `pinMode()` potřebuje i tato funkce dva parametry – číslo pinu a informaci o proudu. Pokud proud teče, je to `HIGH`, pokud ne, tak `LOW`.

```
1 digitalWrite(13, HIGH);
2 digitalWrite(12, LOW);
```

11.3 Čtení vstupu

K detekci napětí na pinu se používá funkce `digitalRead()`. Ta, narozdíl od předchozích dvou funkcí, potřebuje pouze jeden parametr, kterým je číslo pinu, na kterém hodnotu čteme. Tato funkce navíc vrací hodnotu. Když proud teče, vrátí hodnotu `HIGH`, když ne, tak `LOW`.

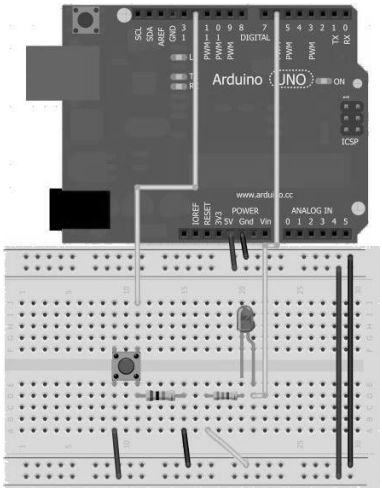
```
1 int cteni;
2 byte vstup = 13;
3
4 cteni = digitalRead(vstup); //pokud proud tece, do promenne cteni se ulozi hodnota
      HIGH, pokud ne, tak LOW
```

Kapitola 12
Příklad: Tlačítko a LED dioda

Ukážeme si program, který bude zjišťovat, jestli je stisknuté tlačítko. Pokud bude, rozsvítí se LED dioda. K této ukázce budeme potřebovat:

1. Desku Arduino.
2. PC.
3. Nepájivé kontaktní pole s vodiči.
4. Tlačítko.
5. 10 kΩ rezistor.
6. 220 Ω rezistor.
7. LED dioda.

Vše zapojíme podle schématu. Poté nahrajeme do Arduina uvedený program.



Obrázek 12.1: Zapojení tlačítka a LED diody

```
1  int cteni;
2  int led = 6;
3  int tlacitko = 12;
4
5  void setup() {
6      pinMode(led, OUTPUT);
7      pinMode(tlacitko, INPUT);
8  }
9
10 void loop() {
11     cteni = digitalRead(tlacitko);
12     digitalWrite(led, cteni);
13 }
```

Na závěr je nutno dodat, že pokud program nefunguje, nejčastější chybou je chybějící středník na konci řádku. Pokud tedy IDE napíše nějakou chybovou hlášku, nejdříve zkontrolujte středníky! Ty se píší na konec řádku za: deklarací proměnné, za přiřazením hodnoty proměnné a za voláním funkce atd.

Část V

Pokročilejší struktury jazyka Wiring

V této části si ukážeme další možnosti jazyka Wiring. Na začátku si řekneme, co jsou to konstanty a jak je používat. Poté si ukážeme, jak pracovat s analogovým vstupem a výstupem, pomocí něhož se dají získávat data z různých analogových senzorů. Nakonec se dostaneme k velmi důležité součásti jakéhokoliv programovacího jazyka, kterou jsou podmínky.

Kapitola 13
Konstanty

Konstanty si můžeme představit jako proměnné, které mají přednastavenou hodnotu, definovanou tvůrci Arduina. Mají za úkol zpřehlednit práci a přiblížit kód lidskému jazyku. My jsme s některými z nich pracovali již dříve. Můžeme je rozdělit do tří skupin.

13.1 Logické konstanty

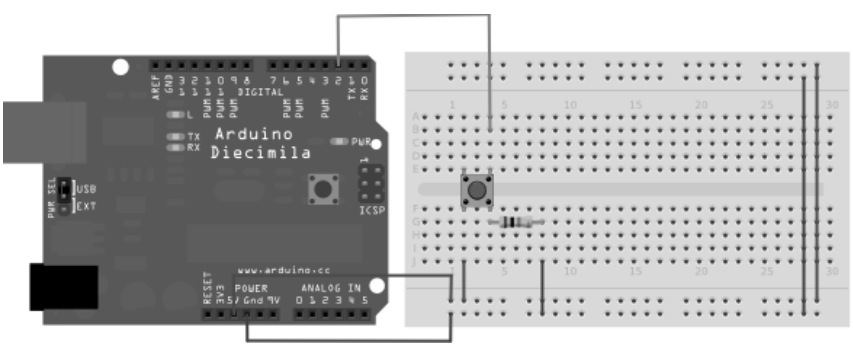
Jsou pouze dvě hodnoty, a to pravda a nepravda. Ve Wiring jim odpovídají konstanty `true` a `false`. Používají se tam, kde je třeba rozhodovat pouze mezi dvěma stavy.

- `false`: Konstanta `false` má hodnotu 0.
- `true`: U konstanty `true` je situace trochu komplikovanější. Mohlo by se zdát, že má hodnotu 1, ale není to úplně přesné. Při logických operacích totiž jazyk Wiring vnímá jakékoliv nenulové číslo typu `integer` jako `true`.

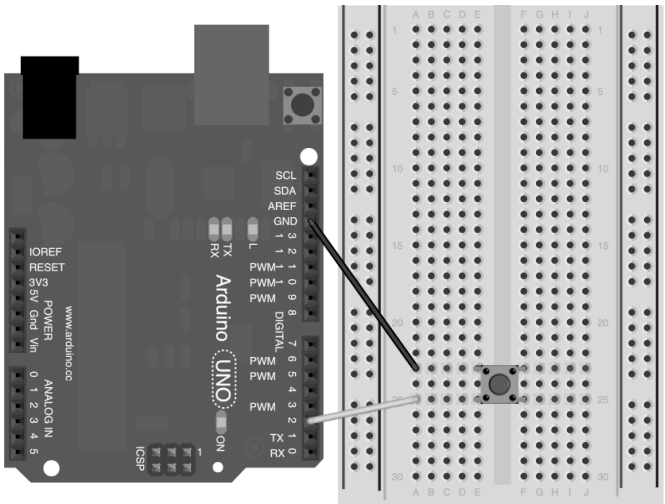
13.2 Typ digitálního pinu

V minulém části jsme při určování, zda se bude pin chovat jako vstup, nebo jako výstup, používali ve funkci `pinMode()` dvě konstanty – `OUTPUT` a `INPUT`. Nyní si k nim přidáme ještě třetí možnost `INPUT_PULLUP`.

- `OUTPUT`: Při použití této konstanty je pin nastaven jako výstup. Ten snese proud do 40 mA. Při stavu `HIGH` tedy tento výstup může poskytnout proud do 40 mA a při stavu `LOW` může stejně velký proud přijmout.
- `INPUT`: Nastaví pin jako vstup. Ten se používá ke čtení hodnot z digitálních senzorů (nejjednodušší jsou tlačítka), ale i ke komunikaci. Použití s tlačítkem jsme si již ukázali. V jeho zapojení si všimněme, že je tento pin stále připojen ke GND přes 10 kΩ rezistor. Při nezmáčknutém tlačítku je tedy výsledek funkce `digitalRead()` hodnota `LOW`. Po zmáčknutí tlačítka dojde k připojení k +5V a změny hodnoty funkce na `HIGH`.
- `INPUT_PULLUP`: Funguje podobně jako `INPUT`. Rozdíl je v tom, že dojde k připojení interního rezistoru. Ten je uvnitř čipu zapojen mezi digitálním vstupem a +5V. Výchozí hodnota funkce `digitalRead()` při použití `INPUT_PULLUP` je tedy `HIGH`. Když chceme hodnotu změnit, musíme vstup připojit na GND. Při použití příkladu s tlačítkem má tedy funkce hodnotu `HIGH`, když je tlačítko uvolněno, a `LOW`, když je zmáčknuto.



Obrázek 13.1: Tlačítko – `INPUT`



Obrázek 13.2: Tlačítko – `INPUT_PULLUP`

13.3 Napětí na digitálních pinech

Pokud pracujeme s digitálními vstupy a výstupy, můžeme rozlišovat pouze dva stavy napětí. Jsou to `HIGH` a `LOW`.

- `HIGH`:
  - Při nastavování stavu napětí pomocí funkce `digitalWrite()` je v případě `HIGH` mezi GND a výstupem hodnota napětí 5 V (u Arduin pracujících na 5 V).
  - Pokud čteme napětí na vstupu pomocí `digitalRead()`, situace je trochu jiná. Jako `HIGH` bude stav vstupu vyhodnocen tehdy, když na něm bude hodnota napětí mezi 3 V a 5 V, což je ale vítaná vlastnost.
- `LOW`:
  - Při zápisu je hodnota napětí mezi GND a pinem 0V. Důležité je si uvědomit, že výstup nastavený na `LOW` může zároveň „přijmout“ napětí do 5 V, což výstup nastavený na `HIGH` nemůže. To nám při některých aplikacích přijde vhod (například u maticových displejů).
  - Při čtení je hodnota na vstupu vyhodnocena jako `LOW`, naměříme-li mezi GND a pinem napětí menší než 2 V.

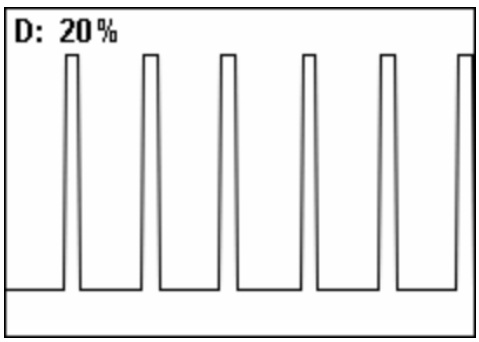
Kapitola 14
Analogový vstup a výstup

S digitálním vstupem a výstupem jsme se setkali už v předchozí části. Co když ale potřebujeme pracovat i s analogovými hodnotami? Na to má Arduino ve výbavě užitečné funkce. Ke čtení a zápisu se zde používají funkce `analogRead()` a `analogWrite()`. Ty jsou však limitovány pro použití pouze na určených pinech. Pojďme si je postupně představit.

14.1 `analogWrite()`

Jak už z názvu vyplývá, jedná se o funkci sloužící k nastavení „analogové“ hodnoty na pinu. Můžeme ji použít pouze na pinech označených PWM (u Arduina Uno jsou to piny: 3, 5, 6, 9, 10, 11). Používá se u ní syntaxe `analogWrite(číslo_pinu, hodnota)`, kdy hodnota může být v rozsahu 0 až 255. Slovo

analogové jsme dali do uvozovek, protože se ve skutečnosti o žádné analogové hodnoty nejedná. Pokud bychom chtěli skutečně analogovou hodnotu v rozsahu například 0 – 5 V, museli bychom použít nějaký externí D/A převodník. Funkce `analogWrite()` totiž na vybraných pinech generuje *PWM* signál, který svým chováním může v některých situacích analogový signál nahradit. PWM v praxi funguje tak, že rychle střídá 0 V a 5 V. To se projeví sníženou „účinností“. LED svítí slaběji (ve skutečnosti rychle bliká a střídá pouze dva stavy napětí a snížená intenzita je způsobena setrvačností oka), motor se točí pomaleji atd. Podle poměru času, ve kterém je na výstupu +5 V ku stavu 0 V se pak odvíjí intenzita svícení LED diody a podobně. Při volání funkce `analogWrite(pin, 127)` je tedy přibližně 50 % času nastaveno napětí +5 V a 50 % času 0 V. Graf napětí v čase můžete vidět na obrázku.



Obrázek 14.1: Graf PWM modulace [33]

14.2 `analogRead()`

Funkce `analogRead()` slouží ke čtení analogové hodnoty na vstupech k tomu určeným. Jsou to tedy piny označené písmenem A (například A2). Čtení analogových hodnot je užitečné u různých senzorů (teplota, vlhkost atd.). Většina desek Arduina má analogové vstupy s rozlišením 10 bitů. Pětivoltovou stupnici tedy můžeme „rozkouskovat“ na $2^{10} = 1024$ hodnot (0 – 1023). Například u Arduino Due se ale můžeme setkat až s 12bitovým rozlišením. Zde se dá nastavit požadované rozlišení pomocí funkce `analogReadResolution()`. My se ale budeme zabývat obyčejným Arduinem Uno. Syntaxe je jednoduchá – naměřenou hodnotu uložíme do proměnné pomocí výrazu `promenna = analogRead(pin)`. Nejjednodušším příkladem použití je měření hodnot na potenciometru. Pokud bychom chtěli měřit například stav fotorezistoru, museli bychom ho zapojit do děliče napětí s vhodným rezistorem. Použití obou analogových funkcí si ukážeme na zapojení s LED diodou a potenciometrem.

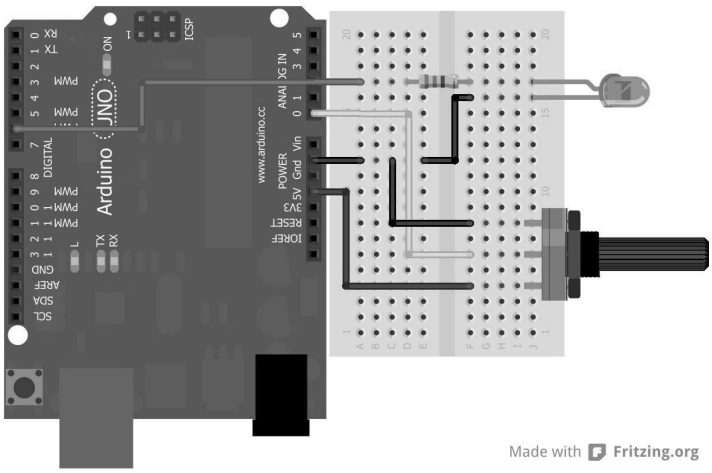
Kapitola 15
Příklad: Regulace jasu LED

Jako příklad si ukážeme zapojení, ve kterém budeme regulovat jas LED pomocí potenciometru.

Budeme potřebovat:

1. Deska Arduino.
2. Nepájivé kontaktní pole s vodiči.
3. LED.
4. Potenciometr.
5. 330 Ω rezistor.

Na trhu nalezneme celou řadu potenciometrů. Nejčastěji se používají ty s odporem kolem 10 kΩ s lineárním průběhem (značeno B10K). Pokud máme všechny komponenty připravené, můžeme je zapojit podle následujícího schématu.



Obrázek 15.1: LED a potenciometr

V programu si musíme dát pozor na hodnoty, se kterými funkce pracují. Funkce `analogRead()` vrací hodnoty 0 až 1023, kdežto `analogWrite()` očekává na rozsah hodnot 0 až 255. Musíme tedy zajistit převod hodnot. To je v tomto případě jednoduché, protože 256 (2^8) se vejde do 1024 (2^{10}) čtyřikrát. Nejjednodušším způsobem je tedy vydělení hodnot z `analogRead()` čtyřmi. (Existuje i elegantnější způsob převodu hodnot, o kterém si povíme dále.) Kód tohoto příkladu bude také velmi jednoduchý. Měření budeme provádět v těle funkce `loop()`.

```
byte led = 6; //pin s LED diodou
byte pot = A0; //pin s pripojenym potenciometrem
int val; //promenna pripravena k uchovani hodnot

void setup() {
    //sem nic nepiseme
}

void loop() {
    val = analogRead(pot)/4; //cteni hodnoty na A0 a uprava rozsahu
    analogWrite(led, val);  //generovani PWM
}
```

Nyní by se měl jas LED měnit v závislosti na úhlu otočení osy potenciometru.

Kapitola 16
Podmínky

Pokud chceme, aby se určitá část kódu provedla pouze v určitých případech, přicházejí na řadu podmínky. Existují tři základní možnosti, které rozdělí program podle zadaných podmínek. V lidské řeči by se daly vyjádřit jako:

1. Pokud platí podmínka, udělej to a to.
2. Pokud platí podmínka, udělej to a to. Pokud neplatí, udělej to a to.
3. Pokud je hodnota proměnné `xy`, udělej to a to. Pokud je `yz`, udělej to a to...

Než se však pustíme do psaní podmínek, musíme se podívat na způsob, jakým se dají v jazyce Wiring zadávat.

16.1 Porovnávací operátory

Porovnávací operátory slouží k zápisu podmínek. Jedná se o systém značek, které jsou pro počítač srozumitelné. Výsledkem porovnávací operace je logická hodnota `true`, nebo `false`. Rozlišujeme šest operátorů.

- `A == B`: `A` je rovno `B`. Vrátí hodnotu `true`, pokud `A` má stejnou hodnotu jako `B`. Opravdu je nutné použít `==` (Zápis `=` slouží k přiřazení hodnoty do proměnné).
- `A != B`: `A` není rovno `B`. Vrátí hodnotu `true`, pokud má `A` jinou hodnotu než `B`.
- `A < B`: `A` je menší než `B`. Vrátí hodnotu `true`, pokud je `A` menší než `B`.
- `A > B`: `A` je větší než `B`. Vrátí `true`, pokud je `A` větší než `B`.
- `A <= B`: `A` je menší nebo rovno `B`. Vrátí `true`, pokud je `A` menší nebo rovno `B`.
- `A >= B`: `A` je větší nebo rovno `B`. Vrátí `true`, pokud je `A` větší nebo rovno `B`.

```
10 == 5 //neni pravda
10 != 5 //je pravda
10 < 5 //neni pravda
10 > 5 //je pravda
10 <= 5 //neni pravda
10 >= 5 //je pravda
```

16.2 Složené podmínky

Někdy dospějeme do situace, ve které je potřeba pracovat s nějakou složitější podmínkou. K tomuto účelu slouží tzv. logické operátory. Můžeme si je představit jako definici vztahu mezi více porovnávacími operátory.

- `X && Y`: a (konjunkce). Výsledkem je `true` pouze v případě, když jsou `true` `X` i `Y`.
- `X || Y`: nebo (disjunkce). Výsledkem je `true` v případě, kdy je alespoň jedna z `X` a `Y` `true`.
- `!X`: negace. Výsledkem je `true`, pokud je `X` `false` a naopak.

```
(1 == 2) && (2 == 2) //false
(1 == 1) && (2 == 2) //true

(1 == 2) || (2 == 3) //false
(1 == 2) || (2 == 2) //true
(2 == 2) || (2 == 3) //true
(2 == 2) || (3 == 3) //true

!(1 == 1) //false
!(1 == 2) //true
!(false) //true
```

16.3 `if()`

Ve většině programovacích jazyků se pro zápis podmínek používá slovo `if`. V jazyce Wiring je možné použít několik způsobů zápisu. Ty ale vždy začínají: `if(podmínka)`.

Pokud chceme mít podmíněný pouze jeden příkaz, můžeme podmínku zapsat takto:

```
//podminky s jednim prikazem

if(x > 120) digitalWrite(LEDpin, HIGH);

if(x > 120)
digitalWrite(LEDpin, HIGH);

if(x > 120){ digitalWrite(LEDpin, HIGH); }
```

Pokud ale má podmínka ovlivnit více příkazů, musíme je uzavřít do složených závorek.

```
1 //podminky s vice prikazy – je nutne pouzit slozene zavorky
2 if(x > 120){
3   digitalWrite(LEDpin1, HIGH);
4   digitalWrite(LEDpin2, HIGH);
5   ...
6 }
```

16.4 else if()

Pokud chceme do podmínky přidat více možností, používá se zápis `else if()`.

```
1 if (A >= 500){
2     //prikazy
3 }
4 else if ((A < 500) && (A > 200)){
5     //prikazy
6 }
```

16.5 else

K části `else` se nepíší další podmínky. Slouží k určení příkazů, které se provedou, pokud ani jedna z předchozích podmínek není splněna.

```
1 if (A >= 800){
2     //prikazy
3 }
4 else if ((A > 200) && (A < 500)){
5     //prikazy
6 }
7 else{
8     //prikazy
9 }
```

Pokud je `A` větší nebo rovno 800, provede se první podmínka a druhá se přeskočí. Pokud je mezi 200 a 500, provede se druhá podmínka. Pokud nic z toho neplatí, provede se třetí blok příkazů.

16.6 switch

Příkaz `switch` je speciální druh podmínky. Speciální je v tom, že se zabývá pouze proměnnou a její hodnotou. Program prochází každou větev konstrukce `switch` a testuje hodnotu proměnné. Další rozdíl je v tom, že se může provést i více větví (což u `if` nelze). Pokud ale chceme, aby po provedení větve program pokračoval až za koncem konstrukce `switch`, musíme použít na konci větve příkaz `break;`.

Syntaxe je následující:

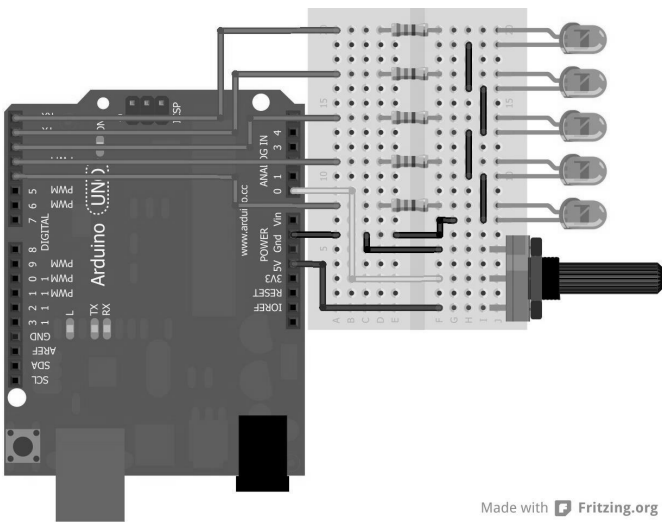
```
1  switch (promenna){
2      case 1:
3          //pokud je hodnota promenne 1, provede se tato cast kodu
4          break; //po provedeni teto casti konstrukce switch konci
5      case 2:
6          //pokud je hodnota promenne 2, provede se tato cast kodu
7          break;
8      default:
9          /* pokud se hodnota promenne nerovna zadne z nabizenych moznosti,
10         provede se tato cast */
11   }
```

Kapitola 17
Příklad: Pás LED diod

Na závěr části si trochu pohrajeme s podmínkami. Vytvoříme aplikaci, která bude číst hodnotu z potenciometru a podle ní vybere LED diody, které se rozsvítí. Pro předváděcí účely jsme zvolili pět diod.

Budeme potřebovat:

1. Deska Arduino.
2. Nepájivé kontaktní pole s vodiči.
3. 5× LED.
4. 5× 330 Ω rezistor.
5. Potenciometr.



Obrázek 17.1: Řada LED a potenciometr

Kód vyhodnocující data z potenciometru by mohl vypadat následovně.

```
byte led[] = {0,1,2,3,4}; //pole s piny pripojenych LED diod
byte pot = A0;
int val;

void setup() {
    pinMode(led[0], OUTPUT);
    pinMode(led[1], OUTPUT);
    pinMode(led[2], OUTPUT);
    pinMode(led[3], OUTPUT);
    pinMode(led[4], OUTPUT);
}

void loop() {
    val = analogRead(pot);

    if(val > 800){
        digitalWrite(led[0],HIGH);
    }
```

```
19      else if(val > 600){
20          digitalWrite(led[1],HIGH);
21      }
22      else if(val > 400){
23          digitalWrite(led[2],HIGH);
24      }
25      else if(val > 200){
26          digitalWrite(led[3],HIGH);
27      }
28      else{
29          digitalWrite(led[4],HIGH);
30      }
31
32      delay(250);
33      digitalWrite(led[0],LOW);
34      digitalWrite(led[1],LOW);
35      digitalWrite(led[2],LOW);
36      digitalWrite(led[3],LOW);
37      digitalWrite(led[4],LOW);
38  }
```

Když se nyní podíváte na kód, jsou zde vidět opakování, ve kterých se mění pouze jedno číslo. Jak si v takovýchto případech ulehčit práci si ukážeme dále.

Kapitola 18
Cykly

Jistě si vzpomenete, že jsme se v předchozích příkladech dostali do situace, kdy jsme museli psát prakticky stejnou věc stále dokola třeba jen s menší obměnou (většinou to byla změna čísla). Pokud se při programování stane, že se nám něco pořád opakuje, přicházejí na řadu cykly. Existují tři druhy cyklů. Ty si teď popíšeme a vysvětlíme si rozdíly mezi nimi. Než se ale pustíme do vysvětlování, ukážeme si složené operátory, které se v cyklech (a nejen tam) často používají.

18.1 Složené operátory

Anglicky nazývané *compound operators* jsou operátory, které nám usnadní práci. Zkracují totiž zápis operací, kdy upravujeme hodnotu pouze jedné proměnné. Popišme si je na příkladech.

```
1  //scitani
2  x = x + 2; //zvetsili jsme hodnotu x o 2
3  x += 2; //dosahneme stejneho vysledku s kratsim zapisem
4
5  //odcitani
6  x = x - y; //od x odecteme hodnotu y a vysledek zapiseme do x
7  x -= y; //vysledek je stejny
8
9  //na stejnem principu funguje i nasobeni a deleni
10 x *= 20;
11 x /= 30;
```

Speciálním druhem těchto operátorů jsou `x++` a `x--`. Ty použijeme tehdy, když chceme hodnotu proměnné snížit, nebo zvýšit pouze o 1. Oba dva nejdříve předají svoji hodnotu, a pak dojde k jejich úpravě.

```
1  x = 10;
2  x++; //x ma nyni hodnotu 11
3  x--;
4  int a = x--; //ted uz ma hodnotu 9, ale a ma hodnotu 10
```

Existují ještě `++x` a `--x`. U nich dochází nejdříve k úpravě hodnoty a až následně k vrácení.

```
x = 10;
++x; //x ma nyni hodnotu 11
--x;
int a = --x; //a i x maji hodnotu 9
```

18.2 Cyklus `while()`

Všechny příkazy v cyklu se provádí, dokud je podmínka pravdivá. Za zmínku stojí, že program kontroluje platnost podmínky vždy na začátku cyklu, pokud je nepravdivá, cyklus skončí. Vyjádřeno slovy: „Pokud je podmínka pravdivá, udělej něco a vrať se na začátek. Pokud není, skonči.“ Cyklus `while()` se tedy nemusí provést vůbec. Používá se následující syntaxe:

```
while(podminka){
    prikazy...
}
```

Příklad 1 – ukázka cyklu

```
boolean x = true;

void setup() {
  Serial.begin(9600);
  while(x){
    Serial.println("OPAKUJI");
    x = false; //cyklus se tedy provede jednou
  }
}

void loop() {

}
```

Příklad 2 – výpis všech čísel od 0 do 99

```
byte x = 0;

void setup() {
  Serial.begin(9600);
  while(x < 100){
    Serial.println(x);
    x++;
  }
}

void loop() {

}
```

18.3 Cyklus `do while()`

Cyklus `do while()` se od `while()` liší pouze v tom, že se podmínka kontroluje až na konci. Nejdříve tedy dojde k provedení příkazů a poté se kontrolují podmínky. Slovně: „Udělej něco, a když platí podmínky, vrať se na začátek. Jinak skonči.“ V praxi to tedy znamená, že se tento cyklus provede minimálně jednou. Syntaxe je následující:

```
do{
    prikazy...
}while (podminky); //<--vsimneme si, ze je zde na konci cyklu strednik - ten zde
    musi byt
```

Příklad 1 – ukázka cyklu

```
boolean x = true;

void setup() {
  Serial.begin(9600);
  do{
    Serial.println("OPAKUJI");
    x = false; //cyklus se take provede pouze jednou
  } while(x);
}

void loop() {

}
```

Příklad 2 – počítání od 0 do 99

```
byte x = 0;

void setup() {
  Serial.begin(9600);
  do{
    Serial.println(x);
    x++;
  }while(x < 100);
}

void loop() {

}
```

18.4 Cyklus `for()`

Tento cyklus se používá asi nejčastěji. Jedná se většinou o případy, kdy známe počet opakování cyklu. Na začátku musíme nastavit nějakou proměnnou na počáteční hodnotu a na konci cyklu ji vždy nějak upravíme. Před každým opakováním testujeme zadané podmínky. Když neplatí, cyklus se neprovede.

```
for(inicializace promenne a nastaveni hodnoty; podminka; operace provadene
    pri kazdem opakovani){
    prikazy...
}
```

Příklad 1 – výpis čísel od 0 do 99

```
void setup() {
    Serial.begin(9600);
    for(int i = 0; i < 100; i++){
        Serial.println(i);
    }
}

void loop() {

}
```

Kapitola 19
Příklad: Had z LED diod

Vytvoříme si hada z pěti LED diod (počet je však volitelný), na kterém si ukážeme, jak se dá cyklus `for` využít v praxi. Budeme potřebovat:

1. Desku Arduino.
2. PC.
3. Nepájivé kontaktní pole s vodiči.
4. 5× 330 Ω rezistor.
5. 5× LED diodu.

```
byte led[] = {2,3,4,5,6}; //piny s LED diodami
byte pocet = 5; //pocet diod
int rychlost = 1000; //jakou prodlevu maji jednotliva bliknuti

void setup() {
    for(int i = 0; i < pocet; i++){
        pinMode(led[i], OUTPUT); //nastaveni pinu
    }
}

void loop() {
    for(int i = 0; i < pocet; i++){
        digitalWrite(led[i], HIGH);
        delay(rychlost/2);
        digitalWrite(led[i], LOW);
        delay(rychlost/2);
    }
}
```

Část VI

Sériová komunikace

V této části si ukážeme, jak může Arduino komunikovat přes sériovou linku s jinými zařízeními a deskami Arduino.

Kapitola 20
Sériová komunikace

Na straně 28 jsme si popsali, jak sériová komunikace funguje. Neřekli jsme si ale, k čemu se dá využít. Pokud chceme od Arduina získávat nějaké hodnoty nebo mu je posílat, přichází na řadu právě sériová komunikace. Pracujeme-li například na projektu meteostanice, určitě se hodí získané hodnoty nějakým způsobem zpracovat. Nejlepší je posílat je do PC a tam je v nejjednodušším případě zobrazovat jako text nebo je pomocí dalších programů zpracovávat. Poté přijde na řadu právě sériová komunikace. Ta je také důležitým nástrojem při ladění programů. Když nám něco nefunguje a my nevíme proč, může být užitečné si nechat vypisovat hodnotu proměnné nebo určitý text pouze v podmínce, takže máme kontrolu nad tím, jaká část kódu se právě provádí. Dá se také využít při komunikaci Arduina a dalších zařízení (jiné Arduino, moduly atd.). K tomu slouží piny popsané RX a TX (u Uno odpovídají pinům 0 a 1). Všechny funkce využívající sériovou komunikaci obsahují slovo `Serial`. My si představíme pár nejužitečnějších z těchto funkcí.

Větší desky (Mega, Due…) mají k dispozici více kanálů pro sériovou komunikaci a připojení dalších zařízení. Piny se pak jmenují: RX1, TX1, RX2, TX2… Jejich použití je popsáno zde[10]*. S PC ale komunikuje pouze jedna základní linka, ostatní slouží k připojení periferií.*

K již zmíněnému čtení informací v textové podobě na PC se používá tzv. *Serial monitor*. Ikonu pro jeho spuštění nalezneme v IDE v horním panelu s ikonami úplně vpravo (ikona s lupou). Po spuštění Serial monitoru musíme ještě nastavit rychlost komunikace pomocí rolovací nabídky v pravé dolní části. Ukažme si nyní, jak může Arduino komunikovat s PC právě přes sériovou linku a Serial monitor.

Od verze 1.6.6 obsahuje Arduino IDE velmi zajímavou funkci, kterou je *Serial Plotter*, který nalezneme v nabídce *Tools*. Ten slouží k zobrazení číselných informací posílaných po sériové lince do grafu.

20.1 Zahájení komunikace – `Serial.begin()`

Tato funkce se používá k zahájení sériové komunikace. Do závorek se píše parametr rychlosti této komunikace, který odpovídá počtu přenosů za sekundu. Při komunikaci s PC ale tato rychlost není pouze na našem výběru. Může mít jen několik vyhrazených hodnot (300, 600, 1200, 2400, 4800, 9600, 14 400, 19 200, 28 800, 38 400, 57 600 a 115 200). Asi nejčastěji se používá hodnota 9600. Funkce `Serial.begin()` se většinou volá v části `setup`.

```
1 void setup(){
2     Serial.begin(9600);
3
4     Serial2.begin(9600); //pro Arduina s vice seriovymi linkami
5 }
```

20.2 Odeslání dat – `Serial.print()` a `Serial.println()`

Tyto funkce slouží k odeslání hodnoty z Arduina do PC nebo jiného zařízení. Liší se od sebe tím, že funkce `Serial.print()` posílá data stále na jednom řádku. Funkce `Serial.println()` vždy na konci odeslaných dat přidá znak pro zalomení řádku. Rozdíl mezi těmito dvěma funkcemi můžete vidět na následujících příkladech.

```
1 //ukazka funkce Serial.print();
2 void setup(){
3     Serial.begin(9600);
4 }
5
6 void loop(){
7     Serial.print("abc ");
8     delay(500);
9 }
```

[10] `http://arduino.cc/en/Reference/Serial`

```
1 //ukazka Serial.println();
2 void setup(){
3     Serial.begin(9600);
4 }
5
6 void loop(){
7     Serial.println("abc ");
8     delay(500);
9 }
```

Při pohledu na použití těchto funkcí je jasné, že obě mají jeden povinný parametr, kterým je odesílaná hodnota. V praxi fungují tak, že se odesílá ASCII kód odesílaných znaků, který se poté v PC patřičně interpretuje. V ASCII tabulce znaků má malé `a` v desítkové soustavě hodnotu 97. Při volání funkce `Serial.print('a')` se tedy odesílá číslo 97 a až v PC dojde k jeho překladu. `Serial.print()` a `Serial.println()` mají ale ještě jeden nepovinný parametr. Ten může udávat buď typ odesílaných dat (v jaké soustavě se číslo ve výpisu zobrazí – `DEC` = desítková, `OCT` = osmičková, `HEX` = šestnáctková a `BIN` = dvojková), nebo počet desetinných míst (což je použitelné u datového typu `float`). Pokud u typu `float` neuvedeme počet desetinných míst, automaticky se odesílají pouze dvě místa. Pokud je to potřeba, čísla se zaokrouhlí.

O číselných soustavách si můžete více přečíst na Wikipedii[11].

```
1  //pouziti jednoho parametru
2  Serial.print(97); //posle: 97
3  Serial.print(2.123456); //posle: 2.12
4  Serial.print('a'); //posle: "a"
5  Serial.print("ABCDEFGHIJ"); //posle: "ABCDEFGHIJ"
6
7  //pouziti nepovinneho operatoru pro typ dat
8  Serial.print(97, DEC); //posle: 97
9  Serial.print(97, BIN); //posle: 1100001 (coz je 97 ve dvojkove soustave)
10 Serial.print(97, OCT); //posle: 141 (=97 v osmickove soustave)
11 Serial.print(97, HEX); //posle: 61 (=97 v sestnactkove soustave)
12
13 //pouziti nepovinneho operatoru pro delku cisel
14 Serial.print(4.56789, 0); //posle: 5
15 Serial.print(4.56789, 1); //posle: 4.6
16 Serial.print(4.56789, 3); //posle: 4.568
```

Před chvílí jsme zmínili, že i znaky jsou přenášeny jako číslo, které jim odpovídá v ASCII tabulce znaků. Dá se s nimi tedy pracovat stejně jako s čísly. Následující kód přenáší znak `a`, kterému odpovídá hodnota 97 v různých soustavách.

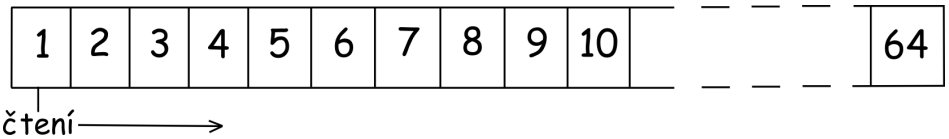
```
1 Serial.print('a', DEC); //posle: 97
2 Serial.print('a', BIN); //posle: 1100001
3 Serial.print('a', OCT); //posle: 141
4 Serial.print('a', HEX); //posle: 61
```

20.3 Čtení dat – `Serial.available()` a `Serial.read()`

Odesílání dat z Arduina již máme za sebou. Nyní se podíváme na možnosti čtení informací, které do Arduina posílá PC nebo jiné zařízení. Nejdříve si musíme říci o tom, jak vlastně posílání dat vypadá na nižší úrovni. Když do Arduina přijdou informace po sériové lince, nezpracovávají se hned, ale jsou uchovány v „zásobníku“ (anglicky *buffer*). Ten dokáže uchovat až 64 bytů. Čtení poté probíhá tak, že se vezme první byte z bufferu, zpracuje se procesorem, jeho místo se uvolní a uchované byty se posunou dopředu. Poté se vezme další byte, zpracuje se atd.

Když chceme Arduinu odeslat nějakou hodnotu, nejjednodušším způsobem je napsat ji do textového pole v horní části Serial monitoru. Jelikož se odesílá ASCII hodnota znaků, má každý znak (i jednotlivé číslice) vlastní byte paměti.

[11] Wikipedia: Číselné soustavy – `http://cs.wikipedia.org/wiki/Číselná_soustava`



Obrázek 20.1: Buffer

Funkce `Serial.available()` a `Serial.read()` jsou uvedeny dohromady, protože spolu souvisí a využívají se obě najednou. Jako první přichází na řadu funkce `Serial.available()`. Ta vrátí počet bytů dostupných v bufferu. Poté dojde na funkci `Serial.read()`, která vezme první byte z bufferu a přečte ho. Zároveň dojde k vyjmutí bytu z bufferu, což se projeví snížením hodnoty `Serial.available()` při dalším čtení. Při spuštění následujícího skriptu a odeslání řetězce „AHOJ“ z PC do Arduina budou v bufferu obsazené 4 byty (za každý znak jeden). Všimněme si, že při vícenásobném odeslání řetězce se počet bytů v bufferu zvětšuje, protože stále nedošlo ke zpracování předchozích dat.

```
1  void setup(){
2      Serial.begin(9600);
3      Serial.println("Komunikace zahajena");
4  }
5
6  void loop(){
7      Serial.print("V bufferu je nyni: ");
8      Serial.print(Serial.available());
9      Serial.println(" bytu.");
10     delay(1000);
11 }
```

A nyní si ukažme použití funkce `Serial.read()`. V tomto příkladu čte Arduino byte po bytu buffer a vypisuje přijaté hodnoty zpět po sériové lince.

```
1  char prijato;
2
3  void setup() {
4      Serial.begin(9600);
5  }
6
7  void loop() {
8      if (Serial.available() > 0) {
9          prijato = Serial.read();
10         Serial.print("Prijato: ");
11         Serial.println(prijato);
12     }
13 }
```

20.4 Ukončení komunikace – `Serial.end()`

Tato funkce se v praxi moc často nepoužívá. Při jejím volání dojde k ukončení sériové komunikace. Nepotřebuje žádný parametr.

```
1  Serial.end();
2
3  Serial1.end(); //u vetsich desek s vice seriovymi linkami
```

Existuje ještě řada dalších funkcí, které pracují se sériovou linkou. My jsme si představili ty nejzákladnější. Popis zbylých funkcí naleznete v jejich dokumentaci[12].

[12] Dokumentace Arduino Serial – `http://arduino.cc/en/reference/serial`

Část VII

Užitečné funkce

Kapitola 21
Čas

V základní výbavě mají desky Arduino čtyři funkce pro práci s časem. Jsou to tyto funkce: `delay()`, `delayMicroseconds()`, `millis()` a `micros()`. První dvě a druhé dvě fungují na stejném principu, jenom pracují s jinými časovými jednotkami. Jsou to milisekundy a mikrosekundy. Důležité je si připomenout převodní vztah mezi jednotkami času kdy: 1 sekunda = 1 000 milisekund = 1 000 000 mikrosekund.

21.1 `delay()`

S touto funkcí jsme se již setkali. Má jediný parametr, a to čas čekání v milisekundách. Rozsah parametru je od 0 do 4 294 967 295. Velkou nevýhodou funkce `delay` i následující funkce `delayMicroseconds()` je fakt, že dojde k zastavení téměř veškeré činnosti (pozastavení čtení hodnot ze senzorů, nemožnost ovládat logické hodnoty na pinech atd.). Nedojde však k zastavení těch funkcí, které nejsou přímo závislé na procesoru. Jedná se zejména o příjem informací z RX linky, kdy se přijatými byty naplňuje buffer a ke zpracování dojde až po skončení funkce delay a také o funkci `analogWrite()`. Generování PWM signálu totiž probíhá mimo hlavní blok procesoru.

```
1  int cekejSekund = 1;
2  long cekejMilisekund = cekejSekund*1000;
3
4  void setup() {
5      pinMode(13, OUTPUT);
6  }
7
8  void loop() {
9      delay(cekejMilisekund);
10     digitalWrite(13, HIGH);
11     delay(cekejMilisekund);
12     digitalWrite(13, LOW);
13 }
```

21.2 `delayMicroseconds()`

Funkce je obdobná, jako `delay()`, jenom s tím rozdílem, že parametr je zde čas v mikrosekundách. Rozsah parametru je od 0 do 65 535.

21.3 `millis()`

Pomocí funkce `millis()` se dá zjistit hodnota uložená ve vnitřním časovači procesoru. Zde je uchována informace o délce běhu programu od jeho spuštění. Tato funkce tedy nepotřebuje žádný parametr a vrací počet milisekund od začátku programu. Tento počet však není nekonečný. Maximální vrácená hodnota je 4 294 967 295. Po překročení dojde k takzvanému přetečení a časovač znovu začne počítat od nuly. Funkce `millis()` se využívá například tam, kde je třeba spouštět určité příkazy jen v daný čas, ale není žádoucí, aby byl přerušen chod programu.

```
1  //program, ktery posle kazdou sekundu zpravu o poctu ms po seriove lince
2
3  long cas = 0;
4
5  void setup() {
6      Serial.begin(9600);
7  }
8
9  void loop() {
10     if(millis() >= cas + 1000){
11         cas = millis();
12         Serial.println(cas);
13     }
14 }
```

K přetečení časovače dojde přibližně jednou za 50 dní. (4 294 967 295 ms = 4 294 967 s = 71 582 min = 1193 h = 49,7 dní)

21.4 micros()

Funkce `micros()` je stejná jako `millis()`, pouze vrací hodnotu v mikrosekundách. Rozsah hodnot je stejný, ale jelikož platí, že 1 milisekunda = 1000 mikrosekund, doba přetečení bude tisíckrát menší, tedy asi 71,5 minuty. Nutno dodat, že rozlišení funkce je u 16MHz procesorů 4 mikrosekundy a u 8MHz 8 mikrosekund. Výstupem funkce tedy bude násobek čtyř, resp. osmi.

Možná se ptáte, proč jsou maximální hodnoty parametrů funkcí nebo čísel, které vrácí, takové, jaké jsou. Je tomu tak, protože funkce pro práci s časem používají datové typy **`unsigned int`** *a* **`unsigned long`***. Datový typ* **`unsigned int`** *může uchovat stejný počet hodnot jako* **`int`***, jenom je tento rozsah posunut směrem do kladných čísel. Typ* **`int`** *může uchovat čísla od −32 768 do 32 767. U typu* **`unsigned int`** *se nepracuje se zápornými čísly. Rozsah je u něj tedy od 0 do 65 535. Stejná situace je i u* **`unsigned long`***, jen s větším rozsahem.*

Kapitola 22
Matematické funkce

Nyní si pojďme ukázat, jaké matematické operace Arduino podporuje. Než ale začneme, připomeňme si základní operátory.

22.1 Matematické operátory

Většina těchto operátorů je nám dobře známá z hodin matematiky. Jedinou výjimkou je operátor `%` (anglicky *modulo*), který vrací zbytek po celočíselném dělení.

```
1  1 + 2 = 3 //scitani
2  2 - 1 = 1 //odcitani
3  2 * 3 = 6 //nasobeni
4  9 / 3 = 3 //deleni
5
6  //modulo
7  9 % 6 = 3
8  //na prvni pohled je tato operace pomerne zvlastni
9  //slovy se da ale jednoduse vyjadrit jako:
10 9 deleno 6 je 1 zbytek 3
11 //operace modulo nam vrati prave tento zbytek
```

Praktické využití nachází operace modulo mimo jiné i v určování dělitelnosti. Pokud je zbytek po dělení `a` číslem `b` nula, potom je `a` dělitelné `b`.

```
1 int a = 20;
2 int b = 5;
3
4 if(a % b == 0){
5     //a je delitelne b
6 }
7 else{
8     //a neni delitelne b
9 }
```

22.2 min()

Funkce `min()` slouží k výběru menšího z čísel. Vstupními parametry jsou dvě čísla a výstupem hodnota menšího z nich. Používá se například při hledání nejmenšího čísla v poli nebo k omezení hodnot ze senzoru (aby nedošlo k překročení určité meze).

```
1  //hledani nejmensiho cisla v poli
2  void setup() {
3      int delka = 10;
4      int pole[] = {2, 4, -8, 3, 2, 100, 200, 50, 99, 358};
5      int nejm = pole[0]; //vezme prvni prvek pole
6
7      Serial.begin(9600);
8
9      for(int i = 1; i < delka; i++){
10         nejm = min(nejm, pole[i]);
11     }
12
13     Serial.println(nejm);
14 }
15
16 void loop() {
17
18 }
```

22.3 max()

Syntaxe této funkce je shodná s `min()`. Jejími parametry jsou také dvě čísla a vrácenou hodnotou je větší z nich. Může být použita například při hledání největšího čísla v poli.

```
1  //hledani nejvetsiho cisla v poli
2  void setup() {
3      int delka = 10;
4      int pole[] = {2, 4, -8, 3, 2, 100, 200, 50, 99, 358};
5      int nejv = pole[0]; //vezme prvni prvek pole
6
7      Serial.begin(9600);
8
9      for(int i = 1; i < delka; i++){
10         nejv = max(nejv, pole[i]);
11     }
12
13     Serial.println(nejv);
14 }
15
16 void loop() {
17
18 }
```

22.4 abs()

Tato funkce vrátí absolutní hodnotu čísla. Vstupem je tedy číslo a výstupem jeho absolutní hodnota.

Absolutní hodnota čísla x je nezáporné reálné číslo. Pokud je $x >= 0$, $\text{abs}(x) = x$*. Pokud je $x < 0$, potom* $\text{abs}(x) = -x$*. Jednoduše řečeno je absolutní hodnota vzdálenost čísla od nuly na číselné ose.*

```
1  x = abs(150) //x = 150
2  x = abs(-150) //x = 150
```

22.5 constrain()

Tuto funkci si můžeme představit jako kombinaci `min()` a `max()` s vhodnými parametry. Slouží totiž k omezení rozsahu proměnné jak shora, tak zdola. Má tři parametry: upravovanou hodnotu, dolní mez a horní mez. Pokud vstupní číslo klesne pod spodní hranici, výstupem je hodnota spodní hranice. Překročí-li horní hranici, výslednou hodnotou je hodnota horní hranice. Pokud je hodnota v mezích, výstupem funkce bude stejná hodnota jako na vstupním parametru.

```
x = constrain(upravovana hodnota, dolni mez, horni mez);

x = constrain(1,10,100); //x = 10
x = constrain(150,10,100); //x = 100
x = constrain(50,10,100); //x = 50
```

22.6 map()

Stejně jako funkce `constrain()` slouží i funkce `map()` k úpravě rozsahu proměnné. Na rozdíl od předchozí funkce však nedochází k oříznutí hodnot, ale k rovnoměrnému „roztažení“, nebo „zmáčknutí“ celé stupnice. Dá se použít například k úpravě hodnoty získané při čtení analogového vstupu (0 – 1023) a jejich použití ve funkci `analogWrite`, která pracuje s hodnotami od 0 do 255. Syntaxe je následující: `x = map (hodnota, minimumPůvodníStupnice, maximumPůvodníStupnice, minimumNovéStupnice, maximumNovéStupnice);`.

```
//uprava jasu LED pomoci potenciometru

int analog, pwm;

void setup() {
    Serial.begin(9600);
}

void loop() {
    analog = analogRead(A0);
    pwm = map(analog, 0, 1023, 0, 255);
    Serial.print("Analog: ");
    Serial.print(analog);
    Serial.print(" PWM: ");
    Serial.println(pwm);

    analogWrite(11, pwm);
}
```

22.7 pow()

Funkce pro umocnění čísla na jiné číslo. Vstupními parametry jsou číslo a mocnina.

```
pow(10,3) = 1000;
pow(10,4) = 10000;
pow(2,5) = 32;
```

```
//ukazka pouziti

int a = 10;
int b = 3;
float c;

void setup() {
  Serial.begin(9600);
}

void loop() {
  c = pow(a,b);
  Serial.println(c);
  delay(1000);
}
```

22.8 sqrt()

Funkce `sqrt()` vrátí druhou odmocninu vstupního čísla.

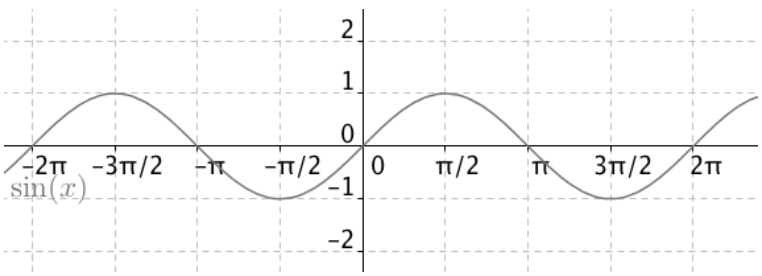
```
1 sqrt(25) = 5;
2 sqrt(256) = 16;
3 sqrt(10000) = 100;
```

22.9 Goniometrické funkce

Než si představíme jednotlivé funkce, povězme si něco o jednotkách úhlů. My jsme totiž většinou zvyklí měřit úhel ve stupních. Plný úhel je zde 360 stupňů. Matematicky přesnější je ale použití radiánů. Tyto jednotky vycházejí z jednotkové kružnice[13]. Plný úhel (360 stupňů) je roven 2π radiánů, $180° = \pi$ radiánů atd. Platí mezi nimi převodní vztah: radiány $=$ (stupně $\cdot\, \pi)/180$. Goniometrické funkce nacházejí uplatnění při výpočtu stran a úhlů v trojúhelnících a dalších rovinných i prostorových útvarech. Všechny goniometrické funkce jsou periodické – po určitém intervalu se jejich hodnoty opakují.

22.9.1 sin()

Funkční hodnota funkce sinus je dána jako poměr strany protilehlé ku přeponě v pravoúhlém trojúhelníku. Amplituda funkce je 1 a perioda 360 stupňů, čili 2π radiánů. Grafem funkce je tzv. sinusoida. Parametrem funkce je úhel v radiánech a výstupní hodnotou je hodnota sinu pro daný úhel. Následující příklad vypíše část sinusoidy otočenou o 90 stupňů vykreslenou pomocí pomlček přes sériovou linku.



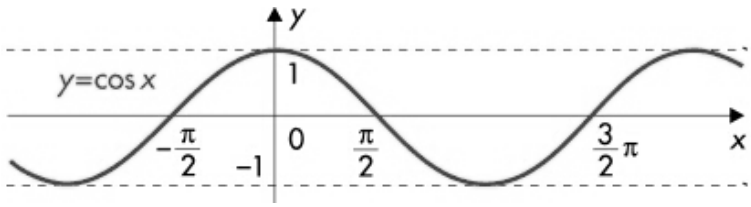
Obrázek 22.1: Graf funkce sinus [42]

```
1  float pi = 3.14159265359;
2  int amplituda = 10; //aby byla sinusoida viditelna, zvetsime jeji amplitudu 10x
3  float perioda = 2*pi;
4  float hodnota;
5
6
7  void setup() {
8      Serial.begin(9600);
9      //tento cyklus nalezne hodnotu funkce sinus po desetinnach PI
10     for(float i = 0; i <= perioda; i+=(pi/10)){
11         //pocet vygenerovanych pomlcek
12         hodnota = sin(i)*amplituda + amplituda;
13
14         for(int j = 0; j < hodnota; j++){
15             Serial.print('-');
16         }
17         Serial.println(' ');
18     }
19 }
20
21 void loop() {
22 }
```

[13] Jednotková kružnice – `http://cs.wikipedia.org/wiki/Jednotková_kružnice`

22.9.2 `cos()`

Funkce `cos()` slouží k určení kosinu dané hodnoty. Je definovaná jako délka přilehlé strany ku přeponě v pravoúhlém trojúhelníku. Jejím grafem je kosinusoida. Sinusoida a kosinusoida jsou si podobné. Kosinusoida je vlastně sinusoida posunutá o $\pi/2$ radiánů doprava. Mezi funkcemi `sin()` a `cos()` platí převodní vztah $\sin(x) = \cos(x - \pi/2)$. Perioda $= 2\pi$, amplituda $= 1$. Na příkladu můžete vidět výpis kosinusoidy.



Obrázek 22.2: Graf funkce kosinus [26]

```
1  float pi = 3.14159265359;
2  int amplituda = 10;
3  float perioda = 2*pi;
4  float hodnota;
5
6
7  void setup() {
8      Serial.begin(9600);
9      for(float i = 0; i <= perioda; i+=(pi/10)){
10         hodnota = cos(i)*amplituda + amplituda;
11
12         for(int j = 0; j < hodnota; j++){
13             Serial.print('-');
14         }
15     Serial.println(' ');
16     }
17 }
18
19 void loop() {
20 }
```

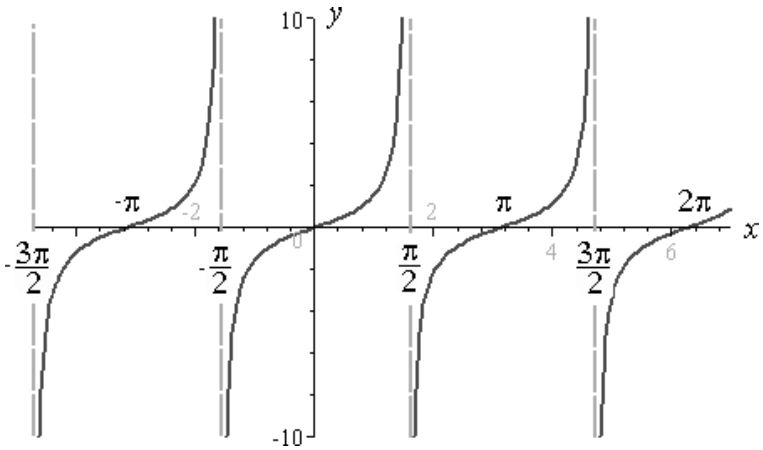
22.9.3 `tan()`

Poslední goniometrickou funkcí, se kterou umí Arduino pracovat, je funkce tangens. Ta je dána jako poměr protilehlé strany ku přilehlé v pravoúhlém trojúhelníku. Má periodu jednoho π.

Funkci kotangens zde nenajdeme. Mezi tg a cotg ale platí vztah $\mathrm{cotg}() = 1/\mathrm{tg}()$, takže jej v případě potřeby můžeme lehce dopočítat.

Kapitola 23
Náhodná čísla

Stroje na rozdíl od člověka neumí jednoduše vytvořit náhodné číslo. Za „náhodným“ číslem totiž většinou stojí složitá série algoritmů, která je však statisticky předpovidatelná. Takovýmto číslům se říká pseudonáhodná. Na první pohled jako náhodná opravdu vypadají, ale ve skutečnosti nejsou. Principy generování opravdu náhodných čísel jsou většinou založeny na měření fyzikální veličiny, která je považována za náhodnou (pohyby plynů a kapalin, fázový šum v laseru…). To je ale pro Arduino poměrně složité.



Obrázek 22.3: Graf funkce tangens [45]

23.1 `random()` a `randomSeed()`

Funkce `random()` slouží ke generování pseudo-náhodných čísel. Může mít jeden, nebo dva parametry.

```
1 random(max); //vygeneruje "nahodne" cislo mezi 0 a max−1
2 random(min, max); //vygeneruje "nahodne" cislo mezi min a max−1
```

Ke správné funkci generátoru je ještě potřeba použít funkci `randomSeed()`. Ta slouží k nastavení výchozí hodnoty pro generátor. Má pouze jeden číselný parametr. Jako hodnota parametru se používá funkce `analogRead()` u analogového pinu, ke kterému není nic připojeno. Dochází kolem něj totiž k zachytávání elektromagnetického šumu, který může sloužit jako náhodná vstupní hodnota. Celý program by tedy mohl vypadat takto:

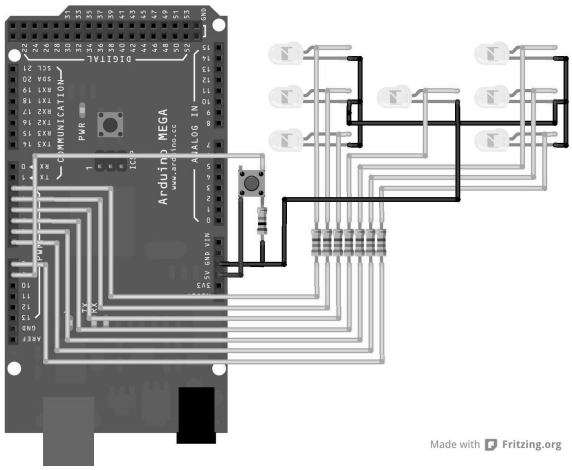
```
1 void setup() {
2     Serial.begin(9600);
3     randomSeed(analogRead(A0));
4 }
5
6 void loop() {
7     Serial.println(random(256));
8     delay(500);
9 }
```

Kapitola 24
Příklad: Hrací kostka

Pomocí funkce `random()` si vytvoříme jednoduchou hrací kostku. Výsledek budeme zobrazovat pomocí LED diod uspořádaných stejně jako černé body na hrací kostce. Využijeme také tlačítko. Po jeho zmáčknutí se vygeneruje nové číslo.

Budeme potřebovat:

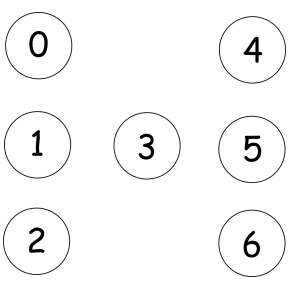
1. Nepájivé kontaktní pole s vodiči.
2. 7× LED dioda.
3. 7× 330 Ω rezistor.
4. Tlačítko.
5. 10 kΩ rezistor.



Obrázek 24.1: Hrací kostka

Vše zapojíme podle obrázku. Není důležité, na jaký pin připojíme jakou LED diodu. Vše se dá jednoduše upravit v programu.

Před uploadem programu do Arduina musíme upravit pole s informacemi o tom, na jaké piny jsou připojeny LED a tlačítko. Jedná se o pole `leds[]` a proměnnou `tlacitko`. Led diody na kostce jsou očíslovány následovně (číslo LED odpovídá jejímu indexu v poli).



Obrázek 24.2: Schéma kostky

Zdrojový kód programu by poté mohl vypadat například takto.

```
byte leds[7] = {2,3,4,5,6,7,8}; //piny, na ktere jsou pripojeny LED
//index odpovida cislu na predchozim obrazku

//logicke stavy LED pouzitych u jednotlivych cisel
byte cisla[7][7] = {{}, /*prazdne pole - 0 se nezobrazuje*/
                    {0,0,0,1,0,0,0}, /*1*/
                    {1,0,0,0,0,0,1}, /*2*/
                    {1,0,0,1,0,0,1}, /*3*/
                    {1,0,1,0,1,0,1}, /*4*/
                    {1,0,1,1,1,0,1}, /*5*/
                    {1,1,1,0,1,1,1}}; /*6*/

byte tlacitko = 9; //pin s tlacitkem

byte randn; //promenna pro nahodnou hodnotu

void setup() {
    Serial.begin(9600);
    randomSeed(analogRead(A0)); //inicializace generatoru
```

```
20      for(int i = 0; i <= 6; i++){
21          pinMode(leds[i], OUTPUT);
22          digitalWrite(leds[i], HIGH); //kontrola funkce LED
23      }
24      pinMode(tlacitko, INPUT);
25      delay(1000);
26      for(int i = 0; i <= 7; i++){
27          digitalWrite(leds[i], LOW); //vypnuti vsech LED
28      }
29  }
30
31  void loop() {
32      if(digitalRead(tlacitko) == 1){
33          for(int i = 0; i <= 7; i++){
34              digitalWrite(leds[i], LOW); //vypnuti vsech LED
35          }
36          randn = random(1,7);
37
38          for(int i = 0; i <= 6; i++){
39              if(cisla[randn][i] == 1){
40                  digitalWrite(leds[i],HIGH);
41              }
42          }
43
44          delay(1000);
45      }
46  }
```

Pokud vše proběhlo bez chyby, měla by se hodnota na kostce změnit vždy po zmáčknutí tlačítka.

Část VIII

Uživatelem definované funkce

Uživatelem definované funkce, anglicky *user defined functions*, jsou funkce, které může uživatel vytvořit sám. Jak už jsme zjistili v předchozích kapitolách, funkce je jakýsi soubor instrukcí „zabalený“ v jednom příkazu. Může mít vstupní parametry, se kterými dále pracuje. Obsahuje blok příkazů, které se při volání (spuštění) funkce provedou, a také může vracet hodnotu. Zajímavé je, že každá funkce má určitý datový typ. Ten se liší podle typu dat, které vrací. Pokud funkce žádnou hodnotu nevrací, používá se speciální datový typ `void`. Důležité je nezapomenout na to, že proměnné definované v těle funkce není možné používat mimo tuto funkci. Ve funkci však lze používat proměnné definované mimo tělo funkce. Funkce musí být definována mimo tělo jiných funkcí, nezáleží však, jestli je definovaná před, mezi nebo za funkcemi `setup()` a `loop()`.

```
1  //funkce muze byt tedy definovana:
2  //tady
3  void setup() {
4      //tady ne
5  }
6  //tady
7  void loop() {
8      //tady ne
9  }
10 //tady
```

Kapitola 25
Definice funkce

Aby funkce pracovala bez problému, potřebuje mít datový typ, název a závorky.

```
1 datovy_typ nazev_funkce(prostor, pro, parametry){
2     prikazy...
3 }
```

U funkcí bez vstupních parametrů se kulaté závorky nechají prázdné (ale musí zde být).

```
1 void text(){
2     Serial.println("TEXT TEXT");
3 }
```

S parametry se pracuje stejně jako s proměnnými. Pokud funkce nějaké má, musíme je nadefinovat. Definice probíhá v kulatých závorkách. Pokud má funkce více parametrů, oddělují se čárkami.

```
1 void zprava(char a[], char b[]){
2     Serial.print(a);
3     Serial.print(' ');
4     Serial.println(b);
5 }
```

Kapitola 26
Volání funkce

V této chvíli se ale po spuštění programu nic nestane. Funkce je sice vytvořená, ale ještě jsme ji nikde nezavolali (nepoužili). Následující kód vypíše po sériové lince text:

```
1 Ahoj Karle
2 TEXT TEXT
```

```
1  void setup() {
2      Serial.begin(9600);
3      zprava("Ahoj", "Karle");
4      text();
5  }
6
7  void loop(){
8  }
9
10 void text(){
11     Serial.println("TEXT TEXT");
12 }
13
14 void zprava(char a[], char b[]){
15     Serial.print(a);
16     Serial.print(' ');
17     Serial.println(b);
18 }
```

Kapitola 27
Funkce, které vrací hodnotu

Pokud má funkce něco vracet, musí mít jiný datový typ než `void`. Pro vrácení vybrané hodnoty se používá příkaz `return`. Pokud chceme vrátit řetězec znaků, nepoužívá se pole `char[]`, ale datový typ `string`. Také není možné jednoduchým způsobem vrátit pole. Ostatní datové typy se používají stejně.

```
String slovo(){
    return "Ahoj";
}

//volani
Serial.println(slovo()); //vypise: Ahoj
```

Jednoduchá funkce pro sečtení dvou čísel a vrácení součtu poté vypadá takto:

```
void setup() {
    Serial.begin(9600);
    Serial.println(secti(10,11));
}

void loop(){
}

int secti(int a, int b){
    int soucet = a + b;
    return soucet;
}
```

Pro výpočet faktoriálu čísla[14] si můžeme sestavit vlastní funkci:

```
void setup() {
    Serial.begin(9600);
    Serial.println(fact(-2));
}

void loop(){
}

int fact(int n){
    int vysledek;
    if(n <= 0){
        vysledek = 1;
        //pro n <= 0 se nebude nic provadet
        //pro zaporne hodnoty neni faktorial definovan
        //a pro nulu neni potreba nic delat
    }
    else{
        vysledek = n;
        for(int i = n-1; i > 0; i--){
            vysledek *= i;
        }
    }
    return vysledek;
}
```

Výhodnou vlastností je také možnost volání funkce v těle jiné funkce. Ukažme si to na výpočtu Eulerova čísla[15]. Parametrem této funkce bude požadovaná přesnost.

```
void setup() {
    Serial.begin(9600);
    Serial.println(euler(10), 10); //Eulerovo cislo s deseti desetinnymi misty
}

void loop(){
}

float fact(float n){
```

[14] Faktoriál – http://cs.wikipedia.org/wiki/Faktoriál
[15] Eulerovo číslo – http://cs.wikipedia.org/wiki/Eulerovo_číslo

```
10      if(n == 0){
11          return 1;
12      }
13      float vysledek = n;
14      for(int i = n-1; i > 0; i--){
15          vysledek *= i;
16      }
17      return vysledek;
18  }
19
20  float euler(int presnost){
21      float e = 0.0;
22      for(int i = 0; i <= presnost; i++){
23          e += (1/fact(i));
24      }
25      return e;
26  }
```

Kapitola 28
Převody datových typů

Možná jste se již při programování dostali do situace, kdy si program dělal s čísly a datovými typy, co chtěl. Mohlo to být tím, že s čísly pracoval jako s jiným datovým typem, než bychom zrovna potřebovali. Pokud chceme mít jistotu, jaký datový typ z dané operace vyjde, použijeme funkce pro převod datových typů.

28.1 char()

Jak už jsme si řekli před časem, i datový typ `char` je vlastně číslo, které odpovídá číslu znaku v ASCII tabulce.

```
1  Serial.println(char(107)); //vypise: k
```

28.2 byte()

Převede danou hodnotu na datový typ `byte`. Pokud je hodnota větší než rozsah tohoto typu, výsledná hodnota se řídí pravidlem:

`vysledek = vstup % 256;`

```
1  int a = 255;
2  int b = 256;
3  Serial.println(byte(a)); //vypise: 255
4  Serial.println(byte(b)); //vypise: 1
```

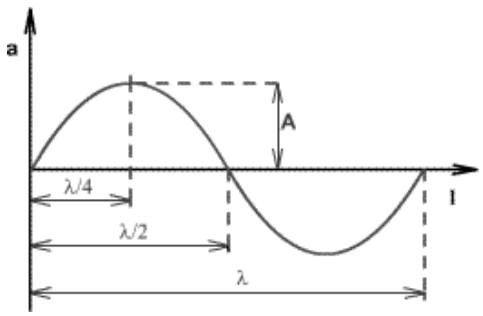
28.3 int(), long(), float()

Konverze těchto typů probíhá stejně jako u těch předchozích. Rozdílem je pouze jiný rozsah výchozích hodnot. U datového typu `float` navíc zadáváme jako druhý parametr počet desetinných míst.

```
1  float a = 12.345;
2  Serial.println(int(a)); //vrati 12
3  Serial.println(float(a), 3); //vrati 12.345
4  Serial.println(long(a)); //vrati 12
```

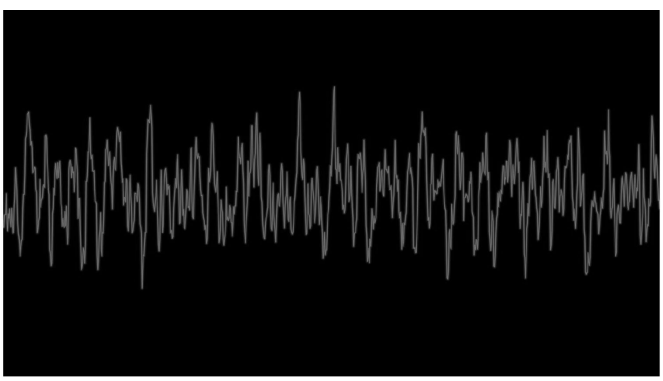
Kapitola 29
Zvuk a tón

Zvuk si můžeme představit jako mechanické kmity částic vzduchu či jiného materiálu. Slyšíme, protože kmitající částice narážejí do ušního bubínku a rozkmitávají ho. Jednotlivé vlny jsou převáděny na nervové signály, které jsou poté zpracovány mozkem. Jednoduchá zvuková vlna může vypadat například jako sinusoida.



Obrázek 29.1: Graf funkce sinus [41]

Ve skutečnosti ale nejsou zvukové vlny ideální a matematicky přesně popsatelné jako sinusoida. Jejich záznam může vypadat třeba takto:



Obrázek 29.2: Graf záznamu zvukové stopy [47]

U Arduina je možné generovat zvuk pouze v nejjednodušší podobě. Neumožňuje totiž generovat analogové hodnoty. Rozlišuje tedy pouze 0 V a 5 V. Výsledná vlna nazývaná squarewave vycházející z Arduina je na obrázku č. 29.3.

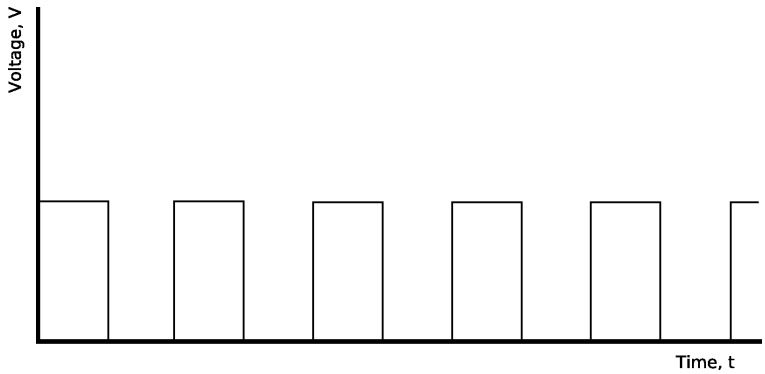
Výška tónu závisí na frekvenci, to je počet opakování „hřebenů“ vlny za jednu sekundu, což vztaženo na Arduino znamená počet změn z 0 na 5 V za sekundu. Jednotkou frekvence je hertz (Hz). Lidské ucho je schopné rozlišit přibližně tóny mezi 20 Hz a 20 000 Hz. Rozsah se však liší mezi jedinci.

Jde samozřejmě generovat i jiné vlny než čtvercové, ale k tomu je potom potřeba přídavný hardware, například D/A převodník.

29.1 `tone()`

Funkce `tone()` slouží ke generování tónu. Má dva povinné a jeden nepovinný parametr. Prvním z nich je pin, na kterém bude připojen reproduktor, druhým je frekvence tónu a nepovinný parametr je délka tónu v milisekundách.

```
1 tone(pin, frekvence, delka);
```



Obrázek 29.3: Graf Squarewave [43]

29.2 noTone()

Touto funkcí se vypne generování tónu na daném pinu. Používá se tedy, když není nastavena délka tónu ve funkci `tone()`.

29.3 Příklad: Třítónový bzučák

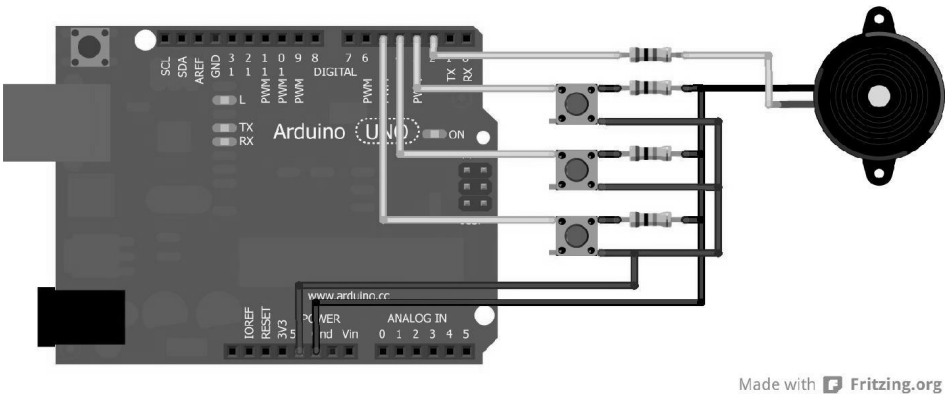
Vytvoříme si příklad, ve kterém budeme vybírat tón pomocí tří tlačítek. Abychom nemuseli zadávat frekvenci tónu stále jen čísly, existuje jakýsi „slovník“, ve kterém je vždy název tónu a patřičná frekvence. Do programu se přidá příkazem `#include "pitches.h"` umístěným na začátku programu. Poté ještě musíme soubor fyzicky přidat k programu. Pod ikonou pro spuštění Sériové komunikace naleznete šipku, která rozevře rozbalovací nabídku. Zvolíme možnost *New Tab* a do pole pro název zadáme `pitches.h`[16]. Do vzniklé záložky zkopírujeme obsah souboru `pitches.h`. Poté vše zapojíme podle schématu. Budeme potřebovat:

1. Arduino.
2. Piezzo reproduktor.
3. Nepájivé kontaktní pole s vodiči.
4. 3× tlačítko.
5. 3× 10 kΩ rezistor.
6. 1× 100 kΩ rezistor (doporučený k Piezo).

Program poté vypadá následovně (tóny záleží na našem výběru):

```
#include "pitches.h"

void setup() {
}

void loop() {
    if(digitalRead(5) == 1){
        tone(2, NOTE_A4, 200);
    }
    if(digitalRead(4) == 1){
        tone(2, NOTE_C5, 200);
    }
    if(digitalRead(3) == 1){
        tone(2, NOTE_E5, 200);
    }
}
```

[16] `Pitches.h – http://files.hwkitchen.com/200001323-097bc0a751/pitches.h`



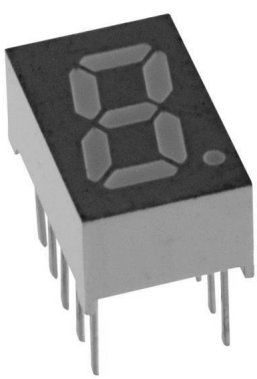
Obrázek 29.4: Bzučák s tlačítky

Kapitola 30
Segmentové displeje

Občas se hodí, když můžeme přímo zobrazit určitý znak nebo číslo bez použití sériové komunikace. K tomuto účelu slouží různé displeje. Nyní se nebudeme zabývat LCD displeji, ale podíváme se na nejjednodušší způsob zobrazování, kterým jsou segmentové displeje. Jedná se o několik LED zalitých v jednom pouzdře, které dohromady vytvářejí znaky. Často mají tyto LED společnou jednu z elektrod. Podle typu je to buďto anoda, nebo katoda (typ lze vyčíst v dokumentaci daného displeje). Nejčastějším typem je sedmisegmentový displej, který jistě všichni znáte.

30.1 Sedmisegmentový displej

Tento displej je dobře známá osmička, kterou můžeme najít v pokladnách, různých čítačích a dalších zobrazovacích zařízeních. Většinou má sedm vývodů pro jednotlivé segmenty čísla, vývod pro tečku a společnou anodu/katodu.



Obrázek 30.1: Sedmisegmentový displej [40]

V dokumentaci od výrobce nalezneme mimo jiného i vnitřní zapojení displeje.

V tomto případě máme displej se společnou anodou. Jednotlivé segmenty se tedy budou zapínat tím, že nastavíme logickou hodnotu jim odpovídajícím pinům na `LOW`. Společná anoda bude připojena k +5V.

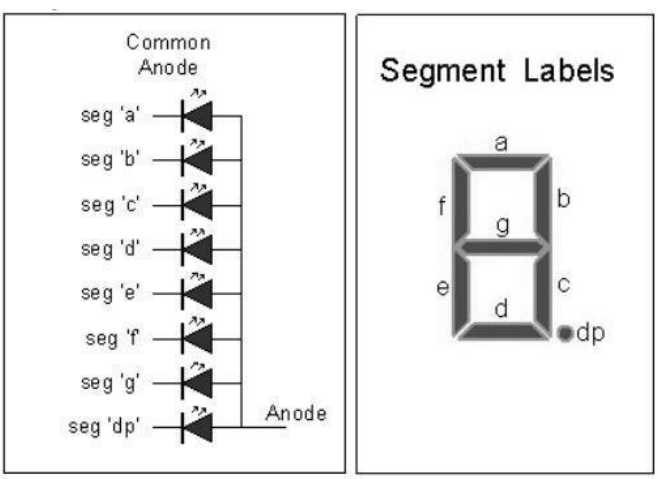
Program, který na displeji vypíše čísla od 0 do 9, bude vypadat následovně:

```
1 byte segmenty[8] = {2,3,4,5,7,8,9,10}; //jake segmenty se pouzivaji pri jakem cisle
2 byte cislice[10][8] =
```

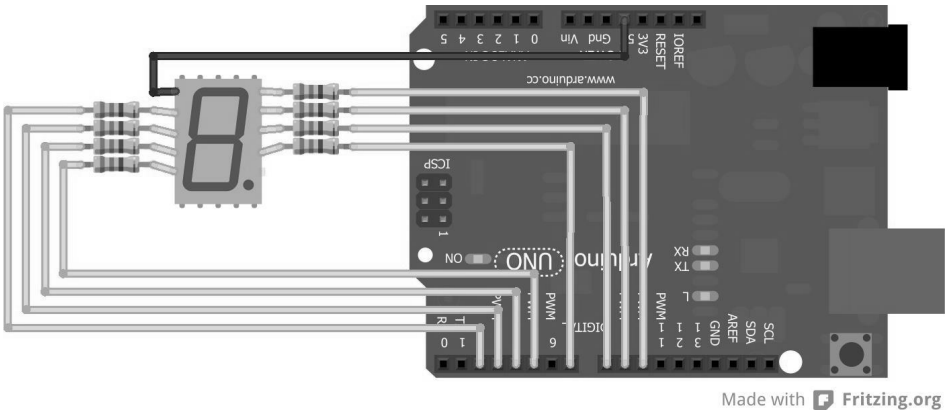
```
{{1,0,1,1,0,1,1,1},{0,0,0,0,0,1,1,0},
{0,1,1,1,0,0,1,1},{0,1,0,1,0,1,1,1},
{1,1,0,0,0,1,1,0},{1,1,0,1,0,1,0,1},
{1,1,1,1,0,1,0,1},{0,0,0,0,0,1,1,1},
{1,1,1,1,0,1,1,1},{1,1,0,0,0,1,1,1}};

void setup() {
    for(int i = 0; i < 8; i++){
        pinMode(segmenty[i], OUTPUT);
        digitalWrite(segmenty[i], LOW);
        delay(500);
        digitalWrite(segmenty[i], HIGH);
    }
    for(int i = 0; i < 10; i++){
        for(int j = 0; j < 8; j++){
            if(cislice[i][j] == 1){
                digitalWrite(segmenty[j], LOW);
            }
            else{
                digitalWrite(segmenty[j], HIGH);
            }
        }
    delay(1000);
    }
    for(int i = 0; i < 8; i++){
        digitalWrite(segmenty[i], HIGH); //vypne vse
    }
}

void loop() {
}
```

30.2 Vícesegmentové displeje

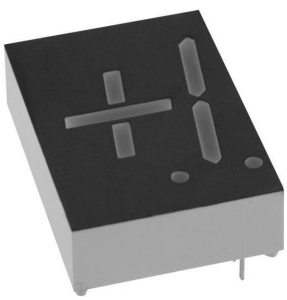
Segmentových displejů existuje celá řada. Může se jednat o displeje schopné zobrazit pouze znak 1 a znaménko, až po šestnácti a vícesegmentové displeje pro zobrazování písmen a dalších znaků. Jelikož je použití stejné jako u sedmisegmentových, jen s jiným počtem pinů, nebudeme se jimi více zabývat.



Obrázek 30.2: Vnitřní schéma sedmisegmentového displeje



Obrázek 30.3: Zapojení sedmisegmentového displeje



Obrázek 30.4: Dvousegmentový displej [20]

Obrázek 30.5: Šestnáctisegmentový displej [44]

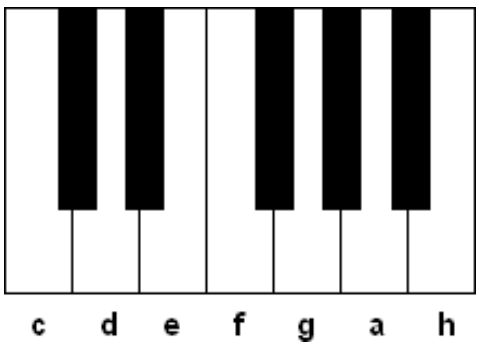
Kapitola 31
Příklad: Klavír

V tomto příkladu si ukážeme, jak vytvořit jednoduchý klavír s možností volby mezi oktávami pomocí potenciometru. Číslo oktávy si necháme posílat pomocí sériové linky. Budeme potřebovat:

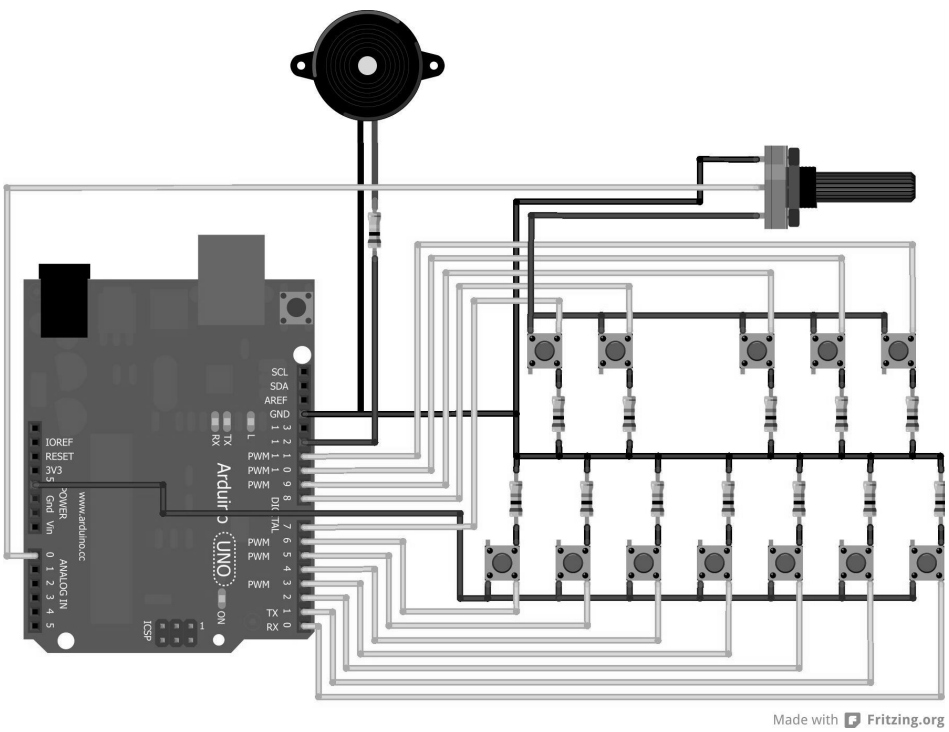
1. Arduino.
2. Piezzo reproduktor.
3. Nepájivé kontaktní pole s vodiči.

4. 12× tlačítko.
5. 12× 10 kΩ rezistor.
6. 1× 100 kΩ rezistor (doporučený k Piezo).
7. Potenciometr.

Na nepájivém kontaktním poli poté tlačítka poskládáme jako na klaviatuře. Jedna oktáva má sedm bílých kláves a pět černých. Každé klávese bude odpovídat jedno tlačítko.



Obrázek 31.1: Klaviatura



Obrázek 31.2: Klavír

Opět musíme k programu přidat soubor `pitches.h`, který obsahuje frekvence jednotlivých tónů. Budeme vybírat ze čtyř oktáv.

```
1  #include "pitches.h";
2  byte oktava;
3  byte piezo = 12;
4
5  byte klavesy[12] = {6,7,5,8,4,3,9,2,10,1,11,0}; //piny jednotlivych tlacitek zleva
       doprava
6  //tony v jednotlivych oktavach
7  int oktavy[4][12] =
8  {{NOTE_C3, NOTE_CS3, NOTE_D3, NOTE_DS3, NOTE_E3, NOTE_F3,
9    NOTE_FS3, NOTE_G3, NOTE_GS3, NOTE_A3, NOTE_AS3, NOTE_B3},
10 {NOTE_C4, NOTE_CS4, NOTE_D4, NOTE_DS4, NOTE_E4, NOTE_F4,
11  NOTE_FS4, NOTE_G4, NOTE_GS4, NOTE_A4, NOTE_AS4, NOTE_B4},
12 {NOTE_C5, NOTE_CS5, NOTE_D5, NOTE_DS5, NOTE_E5, NOTE_F5,
13  NOTE_FS5, NOTE_G5, NOTE_GS5, NOTE_A5, NOTE_AS5, NOTE_B5},
14 {NOTE_C6, NOTE_CS6, NOTE_D6, NOTE_DS6, NOTE_E6, NOTE_F6,
15  NOTE_FS6, NOTE_G6, NOTE_GS6, NOTE_A6, NOTE_AS6, NOTE_B6}};
16
17 void setup() {
18     tone(piezo, 440, 500);
19 }
20
21 void loop() {
22     oktava = map(analogRead(A0),0,1023,0,3);
23     for(int i = 0; i < 12; i++){
24         if(digitalRead(klavesy[i]) == HIGH){
25             tone(piezo, oktavy[oktava][i], 100);
26         }
27     }
28 }
```

Pokud se nedaří nahrát program do Arduina, odpojte napájení desky tlačítek. Po uploadu programu je opět připojte. Při nahrávání programu je totiž potřeba mít volné piny TX a RX.

Část IX

Arduino jako klávesnice a myš

V této části se podíváme na již dříve zmiňované desky, které jsou založeny na čipu ATmega32u4. Ukážeme si, jak se může Arduino s tímto čipem vydávat za klávesnici nebo myš. Také si podrobně popíšeme funkce Arduino Esplora.

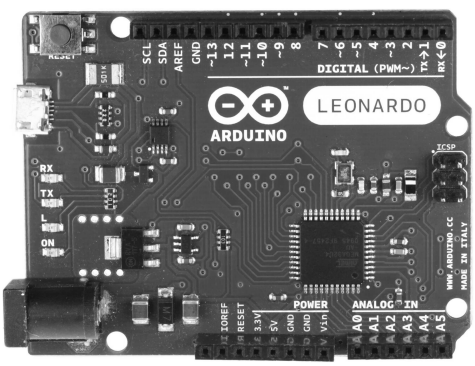
Kapitola 32
Úvod

V první části jsme se zmínili o několika Arduinech, jejichž mozkem je procesor ATmega32u4. Byly to desky Arduino Micro, Lilypad Arduino, Arduino Leonardo, Arduino Yún, Arduino Esplora a Arduino Robot. Jejich největší výhodou je to, že ke komunikaci s PC nepotřebují žádný převodník. Použitý čip má totiž USB převodník přímo v sobě. Díky absenci externího převodníku je daleko jednodušší naprogramovat tato Arduina tak, aby se vydávala za myš či klávesnici (nebo oboje dohromady). V této kapitole se budeme zabývat hlavně deskami Arduino Leonardo a Arduino Esplora. Popsané postupy jsou však použitelné u většiny zmíněných desek.

Níže popsané funkce může využívat i Arduino Due. Nemá sice procesor ATmega32u4, ale jiný, který také pracuje bez přídavného převodníku a může vystupovat jako vstupní zařízení pro PC.

Převodníky dalších originálních desek jsou často založeny na čipu ze stejné řady, jako je ATmega32u4. Není tedy nemožné naprogramovat i jiné desky jako vstupní zařízené pro PC. Jedná se ale většinou o neoficiální postupy. Jeden z nich naleznete například zde[17] *– vše však na vlastní nebezpečí.*

Kapitola 33
Arduino Leonardo



Obrázek 33.1: Arduino Leonardo [6]

Leonardo je velmi podobné dalším deskám z hlavní řady. Jelikož má standardní rozložení konektorů, funguje s ním většina shieldů určená pro Uno. Co se funkčnosti týče, není oproti ostatním nijak omezeno. Navíc má velkou výhodu, a to právě v použitém procesoru. Díky němu je jeho standardní „arzenál“ funkcí rozšířen o ty, které z desky udělají myš nebo klávesnici. Pojďme si nyní představit jednotlivé funkce, které můžeme použít.

Může dojít k tomu, že Arduino nepůjde znovu naprogramovat. Před uploadem programu totiž musí být čip resetován. Za normálních okolností reset probíhá na dálku přes USB. Může se však stát, že čip zrovna dělá něco jiného a upload skončí chybou. Například: „Could not find Leonardo on the selected port.“ V tomto případě musíme čipu s nahráváním pomoci. Před spuštěním uploadu zmáčkneme a držíme tlačítko RESET na desce. Poté zahájíme upload. Ve stavovém řádku se objeví: „Compiling sketch...“. Jakmile tento nápis zmizí a vystřídá ho: „Uploading...“, pustíme tlačítko RESET. Poté by se měl program v pořádku nahrát.

Také si musíme uvědomit, že při používání následujících funkcí přebírá Arduino kontrolu nad PC. Je tedy možné vytvořit program, který zasílá 100 stisků klávesy za sekundu, což se počítači líbit nebude. Pokud se dostaneme do této situace, odpojíme Arduino od USB. Při dalším připojení ihned zmáčkneme tlačítko RESET a poté postupujeme tak, jak bylo popsáno výše.

[17] Změna firmware USB převodníku – `http://mitchtech.net/arduino-usb-hid-keyboard/`

Kapitola 34
Mouse

První skupinou jsou funkce pro ovládání myši. Ty umožňují ovládat pohyb kurzoru, zmáčknutí tlačítek a rolování kolečka myši. První z funkcí je `Mouse.begin()`, která říká programu, že má očekávat práci s myší. Většinou se umisťuje do bloku `setup()`, ale není to podmínkou. Musí však být umístěna v těle funkce (`setup`, `loop`, nebo uživatelem vytvořené). S touto funkcí úzce souvisí `Mouse.end()`, která ovládání myši ukončí.

```
1  void setup() {
2      Mouse.begin();
3      //prikazy mysi
4      Mouse.end();
5  }
6
7  void loop() {}
```

34.1 Mouse.click()

Dalším příkazem je `Mouse.click()`, který odpovídá zmáčknutí a uvolnění vybraného tlačítka. Pokud do závorek neuvedeme žádný parametr, je příkaz vyhodnocen jako zmáčknutí levého tlačítka. V opačném případě může parametr nabývat těchto hodnot:

- `MOUSE_LEFT`: Stisk levého tlačítka myši (výchozí hodnota).
- `MOUSE_RIGHT`: Stisk pravého tlačítka.
- `MOUSE_MIDDLE`: Stisk prostředního tlačítka (kolečka).

Mějme tři tlačítka (`LEFT`, `MIDDLE`, `RIGHT`) zapojená na piny 8, 9, 10, která po řadě odpovídají tlačítkům myši `LEFT`, `MIDDLE` a `RIGHT`. Tyto tlačítka budou fungovat stejně, jako ty u myši.

```
1  byte pin[3] = {8,9,10};
2  byte tlacitko[3] = {MOUSE_LEFT, MOUSE_MIDDLE, MOUSE_RIGHT};
3  byte i;
4
5  void setup() {
6      Mouse.begin();
7      for(i = 0; i < 3; i++){
8          pinMode(pin[i], INPUT);
9      }
10 }
11
12 void loop() {
13     for(i = 0; i < 3; i++){
14         if(digitalRead(pin[i]) == HIGH){
15             Mouse.click(tlacitko[i]);
16         }
17     }
18     delay(100);
19 }
```

34.2 Mouse.move()

Příkaz `Mouse.move(x,y,kolečko)` slouží k pohybu kurzoru myši po obrazovce a také k pohybu kolečka. Důležité je si uvědomit, že pohyb kurzoru pomocí tohoto příkazu není absolutní, ale relativní. Příkaz `Mouse.move(10,-10,0)` tedy neposune kurzor na souřadnice `x = 10`, `y = -10`, ale změní souřadnice vůči aktuální poloze kurzoru o 10 doprava a o 10 nahoru. Berme na vědomí, že osa x leží na horní hraně obrazovky, osa y na levém okraji a počátek os je v levém horním rohu. Pro pohyb kurzoru zleva doprava můžeme využít předchozí zapojení a následující kód:

```
byte pin[3] = {8,9,10};
byte i;

void setup() {
    Mouse.begin();
    for(i = 0; i < 3; i++){
        pinMode(pin[i], INPUT);
    }
}

void loop() {
    if(digitalRead(pin[2]) == HIGH){
        Mouse.move(1,0,0);
    }
    else if(digitalRead(pin[0]) == HIGH){
        Mouse.move(-1,0,0);
    }
    delay(3);
}
```

34.3 `Mouse.press()`, `Mouse.release()` a `Mouse.isPressed()`

Dalším příkazem je `Mouse.press()`. Ten slouží ke zmáčknutí tlačítka a jeho držení. Aby tlačítko nezůstalo zmáčknuté napořád, existuje příkaz `Mouse.release()`, který zmáčknuté tlačítko uvolní. Posledním příkazem je `Mouse.isPressed()`, který zjišťuje, jestli je nějaké tlačítko stisknuté. Parametr všech funkcí může nabývat stejných hodnot jako u `Mouse.click()`. Jeho výchozí hodnota je také `MOUSE_LEFT`.

34.4 Příklad: Myš

Využijeme již hotové zapojení, ke kterému přidáme čtyři tlačítka (každé odpovídající jednomu směru). Rolování kolečka pro zjednodušení vynecháme. Budeme potřebovat:

1. Arduino Leonardo, nebo jiné zmíněné v úvodu.
2. Nepájivé kontaktní pole s vodiči.
3. 7× tlačítko.
4. 7× 10 kΩ rezistor.

```
byte smery[4] = {10,8,9,11}; //nahoru, dolu, doprava, doleva
byte pinyTlacitka[3] = {4,3,2}; //leve, prostredni, prave
byte tlacitka[3] = {MOUSE_LEFT, MOUSE_MIDDLE, MOUSE_RIGHT};
byte posledniStavTlacitek[3] = {LOW,LOW,LOW};
byte hodnotySmeru[4][2] = {{0,-1},{0,1},{1,0},{-1,0}};
//nahoru, dolu, doprava, doleva
byte i;
byte limitPohyb = 2;
//jak casto se bude pohybovat mysi, aby nebyla informace zasilana moc casto
long posledniPohyb = 0;
//promenna pro ulozeni casu posledniho odeslani prikazu

void setup() {
    Mouse.begin();
    for(i = 0; i <= 2; i++){
        pinMode(pinyTlacitka[i], INPUT);
    }
    for(i = 0; i <= 3; i++){
        pinMode(smery[i], INPUT);
    }
```

```
21 }
22
23 void loop() {
24     for(i = 0; i <= 2; i++){
25         if(digitalRead(pinyTlacitka[i]) != posledniStavTlacitek[i]){ //stav
               tlacitka se zmenil
26             if(digitalRead(pinyTlacitka[i]) == HIGH){
27                 Mouse.press(tlacitka[i]);
28             }
29             else{
30                 Mouse.release(tlacitka[i]);
31             }
32             posledniStavTlacitek[i] = !(posledniStavTlacitek[i]);
33         }
34     }
35
36     if(posledniPohyb + limitPohyb < millis()){
37         for(i = 0; i <=3; i++){
38             if(digitalRead(smery[i]) == HIGH){
39                 Mouse.move(hodnotySmeru[i][0],hodnotySmeru[i][1],0);
40             }
41         }
42         posledniPohyb = millis();
43     }
44 }
```

Kapitola 35
Keyboard

Jak už název napovídá, druhá část funkcí slouží k odesílání příkazů ve formě úhozů kláves. Jak vidíte, jsou si funkce pro `Mouse` a `Keyboard` velmi podobné. Nalezneme zde dvojici funkcí `Keyboard.begin()` a `Keyboard.end()` sloužící pro zahájení a ukončení odesílání úhozů kláves.

35.1 Keyboard.write()

Příkazem `Keyboard.write()` se odešle jeden úhoz klávesy. Parametrem je daná klávesa. Pokud chceme odeslat znak, stačí volat funkci s parametrem znaku v uvozovkách nebo s jeho odpovídajícím číslem v ASCII tabulce. Když chceme stisknout nějakou z funkčních kláves (CTRL, F1...), můžeme využít předdefinované konstanty. Jejich seznam nalezneme v oficiální dokumentaci[18]. Vytvořme si program, který nás po zmáčknutí tlačítka pozdraví.

```
1  byte tlacitko = 10;
2
3  void setup() {
4      Keyboard.begin();
5      pinMode(tlacitko, INPUT);
6  }
7
8  void loop() {
9      if(digitalRead(tlacitko) == 1){
10         Keyboard.write('A');
11         Keyboard.write('h');
12         Keyboard.write('o');
13         Keyboard.write('j');
14     }
15     delay(1000);
16 }
```

[18] Dokumentace tlačítek – `http://arduino.cc/en/Reference/KeyboardModifiers`

Před spuštěním programu se musíme přepnout do nějakého textového editoru, aby mělo Arduino kam psát (stačí i nový projekt v Arduino IDE).

35.2 Keyboard.press(), Keyboard.release() a Keyboard.releaseAll()

Funkce `Keyboard.press()` slouží k odeslání informace, že je zmáčknutý znak. Odeslaný znak je stále zmáčknutý, dokud nedojde k jeho uvolnění pomocí funkce `Keyboard.release()`, který uvolní právě daný znak, nebo `Keyboard.releaseAll()`, která uvolní všechny stisknuté znaky. Jako demonstraci si vytvořme herní konzoli pro ovládání hry Sonic[19]. Využijeme zapojení z příkladu s myší. K ovládání hry slouží klávesy a jim odpovídající hodnoty: doleva – `KEY_LEFT_ARROW`, mezerník – 32 a doprava – `KEY_RIGHT_ARROW`.

```
byte pinyTlacitek[3] = {11,10,9}; //doleva, nahoru, doprava
byte posledniStavTlacitek[3] = {LOW,LOW,LOW};
byte tlacitka[3] = {KEY_LEFT_ARROW, 32, KEY_RIGHT_ARROW};
byte i;

void setup() {
    Keyboard.begin();
    for(i = 0; i < 3; i++){
        pinMode(pinyTlacitek[i], INPUT);
    }
}

void loop() {
    for(i = 0; i < 3; i++){
        if(digitalRead(pinyTlacitek[i]) != posledniStavTlacitek[i]){
            //abychom PC nezahltili, posleme prikaz pouze, kdyz se zmeni stav tlacitka
            if(digitalRead(pinyTlacitek[i]) == HIGH){
                Keyboard.press(tlacitka[i]);
            }
            else{
                Keyboard.release(tlacitka[i]);
            }
            //zmena stavu tlacitka v poli pomoci negace
            posledniStavTlacitek[i] = !(posledniStavTlacitek[i]);
        }
    }
}
```

35.3 Keyboard.print() a Keyboard.println()

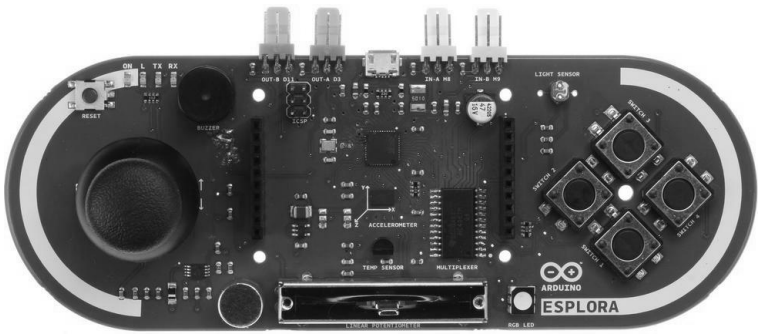
Další dvojicí funkcí jsou `Keyboard.print()` a `Keyboard.println()`. Stejně jako u sériové komunikace, i zde je rozdíl v tom, že po příkazu `Keyboard.println()` následuje odřádkování. Parametrem obou funkcí může být znak nebo řetězec znaků. Pokud je `Keyboard.println()` volána bez parametru, dojde pouze k odřádkování. Program se stejnou funkčností jako předchozí, který nás pozdravil, jen s kratším zápisem, může vypadat takto:

```
byte tlacitko = 10;

void setup() {
    Keyboard.begin();
    pinMode(tlacitko, INPUT);
}

void loop() {
    if(digitalRead(tlacitko) == 1){
```

[19] Hra Sonic – `http://www.superhry.cz/games/14/`

```
10          Keyboard.print("Ahoj");
11      }
12      delay(1000);
13  }
```

Kapitola 36
Arduino Esplora



Obrázek 36.1: Arduino Esplora [3]

Arduino Esplora se od ostatních desek velmi liší a to nejenom svým tvarem, ale i tím, že na něm najdeme spoustu přídavných periferií jako piezo bzučák, teploměr, joystick nebo tlačítka. Také se mírně liší ve způsobu programování. Arduino Esplora se totiž tváří stejně jako Leonardo, jen má vlastní knihovnu příkazů (balíček funkcí), které slouží k využití všech částí desky. U této verze je ještě více než u ostatních patrná snaha o zjednodušení programování a minimalizaci zapojování. Je tedy určena pro všechny, kteří se nechtějí moc zdržovat správným zapojením komponent a chtějí rovnou programovat. Abychom mohli s Esplorou pracovat, musíme na začátek programu vložit: `#include <Esplora.h>`. Pojďme si nyní představit funkce pro práci s jednotlivými částmi desky.

36.1 Joystick

Joystick jistě všichni znáte z různých herních konzolí. Umožňuje ovládání ve dvou osách. Joystick Arduina Esplora má navíc tlačítko, k jehož zmáčknutí dojde při stlačení joysticku směrem k desce. Pro čtení hodnot z joysticku se používají funkce `Esplora.readJoystickX()` a `Esplora.readJoystickY()`. Ty vrací hodnoty od -512 do 512. Pro osu x jsou záporné hodnoty ve směru doleva a kladné doprava. Po ose y jsou kladné hodnoty nahoru a záporné dolů. Pokud je tedy joystick nevychýlený, jeho souřadnice jsou $x = 0$, $y = 0$. Pro zjišťování stavu tlačítka se používá funkce `Esplora.readJoystickButton()`. Pokud je tlačítko zmáčknuto, vrací hodnotu `LOW`. Sestavme si aplikaci, která bude pomocí joysticku ovládat kurzor myši a po jeho zmáčknutí dvakrát klikne levým tlačítkem.

```
1  #include <Esplora.h>
2
3  int x,y;
4
5  void setup(){
6      Mouse.begin();
7  }
8
9  void loop(){
10     x = map(Esplora.readJoystickX(), -512, 512, 2, -2);
11     y = map(Esplora.readJoystickY(), -512, 512, -2, 2);
12     Mouse.move(x,y,0);
13     delay(1);
```

```
14
15      if(Esplora.readJoystickButton() == LOW){
16          Mouse.click();
17          delay(100);
18          Mouse.click();
19      }
20  }
```

36.2 Směrová tlačítka

Rozložení čtyř velkých tlačítek přímo vybízí k jejich použití jako směrové klávesy, nikde však není psáno, že musí být použity právě na směry. Pomocí nich mohou být vytvořeny třeba různé funkční klávesy. Ke zjištění stavu tlačítek slouží funkce `Esplora.readButton()`. Stejně jako u joysticku i zde vrací funkce hodnotu `LOW`, pokud je tlačítko stisknuté. Funkce má jeden parametr, který může nabývat čtyř hodnot, a to:

- `SWITCH_UP`: Nahoru
- `SWITCH_DOWN`: Dolů
- `SWITCH_LEFT`: Doleva
- `SWITCH_RIGHT`: Doprava

Herní konzole pro hru ovládanou šipkami tedy může vypadat takto:

```
1  #include <Esplora.h>
2
3  byte sipky[] = {SWITCH_UP, SWITCH_DOWN, SWITCH_LEFT, SWITCH_RIGHT};
4  byte zkratky[] = {KEY_UP_ARROW, KEY_DOWN_ARROW, KEY_LEFT_ARROW,
       KEY_RIGHT_ARROW};
5  byte posledniStav[] = {1,1,1,1};
6  byte i;
7
8  void setup(){
9      Keyboard.begin();
10     Serial.begin(9600);
11 }
12
13 void loop(){
14     for(i = 0; i < 4; i++){
15         if(Esplora.readButton(sipky[i]) != posledniStav[i]){
16             if(Esplora.readButton(sipky[i]) == LOW){
17                 Keyboard.press(zkratky[i]);
18             }
19             else{
20                 Keyboard.release(zkratky[i]);
21             }
22             posledniStav[i] = !(posledniStav[i]);
23         }
24     }
25 }
```

V oficiální dokumentaci[20] *funkce* ***`readButton()`*** *nalezneme popis dalších parametrů. Všechny však slouží k práci se čtyřmi základními tlačítky a jsou tedy shodné.*

[20] Dokumentace `Esplora.readButton()` – `http://arduino.cc/en/Reference/EsploraReadButton`

36.3 Lineární potenciometr

Lineární potenciometr, anglicky nazývaný *slider*, je umístěn při dolním okraji desky. Funguje stejně jako nám dobře známé potenciometry. Liší se pouze v mechanické konstrukci. Pro čtení hodnot se používá funkce `Esplora.readSlider()`. Ta vrací hodnoty od 0 do 1023. Na nule je hodnota funkce, když je posuvník potenciometru vpravo. Vlevo je to 1023. Příkladem může být jednoduchá funkce odesílající hodnoty ze slideru po sériové lince.

```
1  #include <Esplora.h>
2
3  int hodnota;
4
5  void setup(){
6      Serial.begin(9600);
7  }
8
9  void loop(){
10     hodnota = Esplora.readSlider();
11     Serial.println(hodnota);
12     delay(500);
13 }
```

36.4 Mikrofon

Dalším přídavným hardwarem je standardní elektretový mikrofon. Ten zde slouží převážně k zjištění intenzity okolního hluku než k jiným audio účelům.

Pro čtení hodnot slouží funkce `Esplora.readMicrophone()`. Ta vrací hodnoty v rozmezí 0 až 1023. Jako příklad si uveďme jednoduchou aplikaci, která bude po sériové lince vypisovat „audiovlnu“ se vzorkovací frekvencí cca 1 kHz (kolikrát za sekundu dojde k měření). Pokud bychom takovouto vlnu přehráli, bude velmi zdeformovaná. Pro lidské ucho se totiž doporučuje vzorkovací frekvence 40 kHz a vyšší. Spousta informací obsažených v původní vlně se tedy při 1 kHz ztratí.

```
1  #include <Esplora.h>
2
3  int hodnota;
4
5  void setup(){
6      Serial.begin(9600);
7  }
8
9  void loop(){
10     hodnota = Esplora.readMicrophone();
11     hodnota = map(hodnota, 0, 1023, 0, 50);
12     for(int i = 0; i < hodnota; i++){
13         Serial.print('|');
14     }
15     Serial.println();
16     delay(1);
17 }
```

36.5 Světelný senzor

Světelný senzor zde není nic jiného než obyčejný fotorezistor. Ke čtení hodnot se využívá konkrétně funkce `Esplora.readLightSensor()`, která vrací hodnoty od 0 (pro tmu) do 1023 (pro maximální osvětlení). Příklad práce s měřením světla je stejný jako u mikrofonu, jen se zaměněním funkcí pro získání hodnot, čili:

```
10     hodnota = Esplora.readLightSensor();
```

36.6 Teploměr

Arduino Esplora má také přímo na desce teploměr. Ten měří teplotu ve stupních Celsia nebo Fahrenheita. Připomeňme si převodní vztah, kdy $°C = (°F - 32) \cdot 5/9$. Ve fyzice se častěji používá Kelvinova stupnice. Její nula je definovaná jako absolutní nula ($0° = -273{,}15\,°C$). Velikost jednoho stupně Kelvina odpovídá jednomu stupni Celsia, jen je celá stupnice posunuta směrem k absolutní nule. Pro získání naměřené teploty slouží funkce `Esplora.readTemperature()`. Funkce má jeden parametr, který nabývá dvou hodnot (podle volby požadované stupnice). Pro hodnotu `DEGREES_C` vrátí funkce hodnotu ve stupních Celsia (od -40 do $+150\,°C$). Pro parametr `DEGREES_F` bude vrácená hodnota ve stupních Fahrenheita (od -40 do $+302\,°F$). Níže vidíte program, který bude při zmáčknutí tlačítka každou sekundu vypisovat naměřené hodnoty přes sériovou linku.

```
#include <Esplora.h>

int hodnota;

void setup(){
    Serial.begin(9600);
}

void loop(){
    if(Esplora.readButton(SWITCH_UP) == LOW){
        hodnota = Esplora.readTemperature(DEGREES_C);
        Serial.print(hodnota);
        Serial.println(" C");
        delay(1000);
    }
}
```

Více o Celsiově stupnici naleznete na Wikipedii[21].

36.7 Akcelerometr

Akcelerometr je součástka schopná měřit zrychlení. U Arduina Esplora jej nalezneme ve verzi, která je schopná detekovat zrychlení ve třech na sebe kolmých osách (x, y, z). Nachází se uprostřed desky. Na zařízení působí zrychlení, i když je v klidu a nijak s ním nepohybujeme. Zrychlení totiž vyvolává tíhová síla, kterou na něj působí Země. Poté závisí na úhlu desky vůči zemi, jakou bude zrychlení mít velikost v jednotlivých osách. Pokud je naměřená hodnota zrychlení na nějaké ose nula, znamená to, že právě působí kolmo na danou osu. Osa x jde od středu Esplory směrem ke tlačítkům. Osa y ze středu k USB portu a osa z ze středu vystupuje kolmo nad desku. Pokud tedy Esplora leží na vodorovné podložce, měly by být hodnoty: $x = 0$, $y = 0$ a z přibližně 150. Ke čtení hodnot se používá funkce `Esplora.readAccelerometer()` s parametrem se třemi možnými hodnotami: `X_AXIS`, `Y_AXIS` a `Z_AXIS`. Pro demonstraci můžeme využít příklad z dokumentace. Ten nám bude posílat po sériové lince naměřené hodnoty zrychlení na jednotlivých osách.

```
#include <Esplora.h>

void setup(){
    Serial.begin(9600);

void loop(){
    int xAxis = Esplora.readAccelerometer(X_AXIS);
    int yAxis = Esplora.readAccelerometer(Y_AXIS);
    int zAxis = Esplora.readAccelerometer(Z_AXIS);

    Serial.print("x: ");
    Serial.print(xAxis);
    Serial.print("ty: ");
    Serial.print(yAxis);
    Serial.print("tz: ");
```

[21] Wikipedia: Stupeň Celsia – `http://cs.wikipedia.org/wiki/Stupeň_Celsia`

```
16      Serial.println(zAxis);
17
18      delay(500);
19  }
```

36.8 Piezo bzučák

Na desce také najdeme bzučák, který jsme si už jednou popsali. Použití je stejné, jen ubyde parametr pro výstup, na kterém se tón generuje. Pomocí funkce `Esplora.tone()` se generuje tón. Funkce má jeden povinný parametr pro frekvenci a nepovinný parametr pro trvání tónu. Tón se vypne funkcí `Esplora.noTone()`.

36.9 RGB LED

Na pravé straně slideru nalezneme tříbarevnou LED diodu. Od každé barvy je možné nastavit 256 různých sytostí (0 – 255). Máme-li tři barvy, můžeme teoreticky generovat až 256^3 různých barev. K ovládání se nám nabízejí čtyři různé funkce. `Esplora.writeRed()`, `Esplora.writeGreen()` a `Esplora.writeBlue()` slouží každá ke generování jedné barvy, kdy parametrem je jejich jas v rozsahu 0 až 255. Funkce `Esplora.writeRGB()` je shrnutím předchozích funkcí do jedné. Má tři parametry, a to jas červené, zelené a modré (v tomto pořadí). Vyzkoušejme si mixování barev pomocí slideru a joysticku.

```
1  #include <Esplora.h>
2
3  int r,g,b;
4
5  void setup(){
6  }
7
8  void loop(){
9      r = map(Esplora.readJoystickX(), -512, 512, 0, 255);
10     g = map(Esplora.readJoystickY(), -512, 512, 0, 255);
11     b = map(Esplora.readSlider(), 0, 1023, 0, 255);
12
13     Esplora.writeRGB(r,g,b);
14 }
```

36.10 Neoficiální rozložení pinů (angl. *pinout*)

Oficiální dokumentace se nezmiňuje o jedné věci, a to o využití pinů na přední straně desky. Levá část je bohužel nepřipojená a slouží pouze k lepšímu mechanickému uchycení případných rozšíření. Pravá část však nabízí několik využitelných pinů. Velice dobře zpracovaný popis pinů desky nalezneme v tomto dokumentu[22]. Zde nás budou zajímat hnědě vyznačené piny. Pod jejich číslem je můžeme používat pomocí standardních funkcí jako `digitalRead()`, `digitalWrite()` a dalších. Nikoliv tedy `Esplora.read...`

Neoficiální popis pinů pochází z webu ***`pighixxx.tumblr.com`****. Zde naleznete graficky výborně zpracované rozložení pinů a součástek většiny desek Arduino a spoustu dalších informací včetně doporučeného zapojení dalších přídavných komponent.*

[22] Neoficiální Esplora rozložení pinů – `http://arduino.cz/wp-content/uploads/esplora.pdf`

Část X

Processing

V této části se budeme věnovat vývojovému prostředí Processing. Postupně se podíváme na hlavní části tohoto prostředí a ukážeme si, jak fungují.

Kapitola 37
Úvod

I když by se mohlo zdát, že spolu projekty Arduino a Processing moc nesouvisí, ve skutečnosti tomu tak není. Processing se sice nepoužívá na programování hardware, ale myšlenka obou prostředí je stejná. Má přiblížit programování i neprogramátorům jednoduchou cestou. Prostředí má vlastní stejnojmenný programovací jazyk, který vychází z jazyka Java a značně jej zjednodušuje. Dá se zde však použít i většina příkazů z tohoto jazyka. Vývojové prostředí stáhneme z oficiální stránky[23]. Po kliknutí na download se zobrazí nabídka s různými operačními systémy (Windows, Linux a Mac OS X), ve které si vybereme a stáhneme vhodnou verzi. Silnou stránkou jazyka Processing jsou jeho grafické aplikace. Jednoduše se v něm dají kreslit plošné a po nějakém čase i prostorové geometrické útvary.

Velmi zajímavá instruktážní videa s ukázkami programů vytvořených v tomto prostředí naleznete na jeho oficiálních stránkách `http://hello.processing.org/`.

Kapitola 38
Seznámení

Po rozbalení staženého archivu je prostředí plně funkční. Ve Windows jej spustíme pomocí programu `processing.exe`. Otevře se nám okno velmi podobné Arduino IDE. Nejvíce nás bude zajímat dvojice kulatých a čtveřice hranatých tlačítek v horní části. Jejich funkce je (zleva na obrázku č. 40.1): spuštění programu, zastavení programu, vytvoření nového programu, otevření existujícího programu, uložení programu a tlačítko pro export kódu v podobě spustitelné mimo Processing. I zde slouží velká bílá plocha uprostřed ke psaní kódu. Při programování se používají dvě základní funkce: `setup()` a `draw()`.

```
1 void setup() {}
2
3 void draw() {}
```

Funkce `setup()` funguje stejně jako u Arduina. Funkce `draw()` je funkčně ekvivalentí s `loop()`.

Kapitola 39
Základ

Než se pustíme do grafických aplikací, ukažme si, jakým způsobem se v Processing řídí chod programu.

39.1 Datové typy

Zde nás nečeká téměř nic nového. Stejně jako u Wiring i zde existují datové typy `boolean`, `byte`, `char`, `double`, `float`, `int` a `long`. Navíc se setkáme ještě s datovým typem `color` sloužícím k uchování informace o barvě.

39.2 Pole

I když se zde s poli pracuje prakticky stejně tak, jak známe, dají se nalézt malé odlišnosti, které však mohou způsobit problémy. Hned způsob vytváření polí je mírně jiný.

```
1 //misto
2 int a[5];
3 int b[5] = {1,2,3,4,5};
4 //se zde pouziva syntaxe
5 int[] a = new int[5]; //pro vytvoreni prazdneho pole o peti prvcich
6 int[] b = {1,2,3,8,20};
```

[23] Processing – `https://processing.org/download/`

Práce s buňkami pole je stejná jako ve Wiring.

```
1 //nastaveni hodnoty bunky s danym indexem
2 int[] a = new int[5];
3 a[3] = 10; //nastaveni hodnoty bunky
```

Processing navíc nabízí zajímavou funkci pro zjištění délky pole.

```
1 int[] a = new int[5];
2 int l;
3
4 l = a.length; //l obsahuje delku pole a - tj. 5
```

Tato část je také velmi podobná Wiringu. Používají se zde základní porovnávací operátory, jimiž jsou `!=`, `<`, `>`, `==`, `<=` a `>=`. Podmínky i cykly zde mají stejnou syntaxi, jakou již známe.

```
1  //podminky
2  if (podminka){
3      //prikazy
4  }
5  else if(podminka){
6      //dalsi prikazy
7  }
8  else {
9      //a dalsi prikazy
10 }
11
12 //cyklus
13 for(int i=0; i<10; i++){
14   //prikazy
15 }
```

39.3 Výpis hodnot

Pro výpis textových hodnot slouží panel pod částí pro psaní kódu. Zde se v Arduino IDE zobrazují informace o průběhu uploadu a podobně. K výpisu slouží funkce `print()` a `println()`, které fungují stejně jako nám známé `Serial.print()` a `Serial.println()`.

Kapitola 40
Kreslení

40.1 Vytvoření kreslicího plátna

Abychom měli kam kreslit, musíme pro program vytvořit kreslicí plochu. To se provádí pomocí příkazu `size(x,y)`. Je jasné, že `x` a `y` jsou rozměry v jednotlivých osách. Následující kód tedy vytvoří kreslicí plátno o rozměrech 500×500. Tím nám zároveň vznikne nová vztažná soustava pro kreslení objektů. Důležité je vědět, jakým způsobem jsou zde určené osy. Jejich průsečík (počátek) nalezneme v levém horním rohu plátna. Osa x je rovnoběžná s horním okrajem a její hodnota roste směrem doprava. Osa y je na ní kolmá. Je tedy rovnoběžná s levým okrajem a roste směrem dolů (což by mohlo zkušené matematiky mást).

```
1 void setup() {
2     size(500,500);
3 }
4
5 void draw() {
6 }
```

Zatím to není žádná sláva – šedý obdélník a nic víc. Pojďme si tedy ukázat, jak tento obdélník vyplnit. Mějme na paměti, že se objekty na plátně vrství ve stejném pořadí, v jakém je v programu vytváříme.

Obrázek 40.1: Hlavní okno programu v Processing

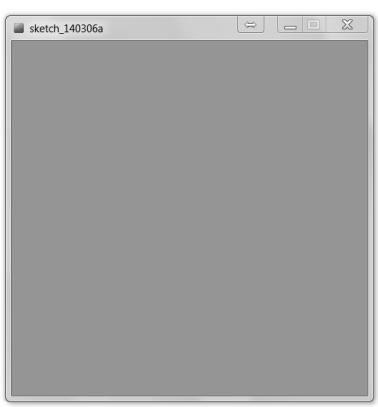
40.2 Barvy

K barvám lze přistupovat různými způsoby. Většinou se nastavují pomocí funkce `color()`, která může mít různý počet parametrů v závislosti na způsobu jejího použití.

```
1 //barva ve stupnich sede od cerne (0) po bilou (255)
2 color(gray);
3 //barva ve stupnich sede se sytosti od 0 do 255 (cim vetsi alpha, tim sytejsi barva)
4 color(gray, alpha);
5 //barva zadana pomoci RGB - cervena, zelena, modra v rozsahu 0 az 255
6 color(r, g, b);
7 //barva RGB se sytosti (0-255)
8 color(r, g, b, alpha);
```

Těmito funkcemi se ale ještě nic neprovede. Pouze vracejí hodnotu zadané barvy jako int. Pro nastavování barev jednotlivých částí kreslicí plochy slouží několik funkcí. První z nich je `background()`, která změní barvu celého pozadí. Jako její parametr můžeme použít právě představenou funkci `color()`.

```
1 void setup(){
2     size(500,500);
3     background(color(0,255,0)); //nastavi pozadi platna na zelenou
4 }
5
6 void draw(){}
```



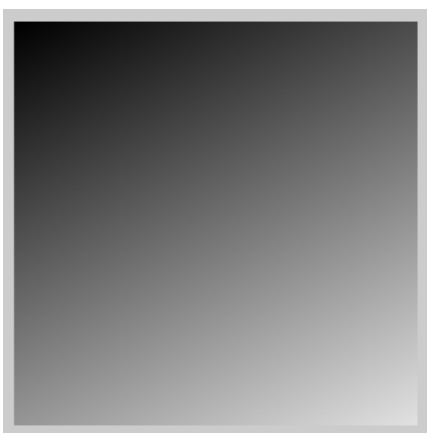
Obrázek 40.2: Funkce `background()`

Kreslené útvary mají většinou okraj a výplň. Pro nastavení barvy okraje se používá funkce `stroke()` a pro výplň funkce `fill()`. Také můžeme programu říct, že si nepřejeme, aby okraj či výplň zobrazoval. Pomocí funkcí `noStroke()` a `noFill()` se tyto části objektu zprůhlední. Pokud použijeme libovolnou z těchto čtyř funkcí, ovlivní všechny útvary, které vytvoříme po jejich volání.

40.3 Bod

S bodem se konečně dostáváme ke geometrickým útvarům. Na plátno jej nakreslíme pomocí funkce `point(x,y)`. Pomocí dvou cyklů a představených funkcí si můžeme jednoduše vytvořit barevnou škálu dvou barev.

```
1  int[] pocatek = {20,20}; //pocatecni souradnice x, y
2
3  void setup(){
4    size(296,296);
5  }
6
7  void draw(){
8      for(int i = 0; i<256; i++){
9          for(int j = 0; j < 256; j++){
10             stroke(color(i,j,0));
11             point(i+pocatek[0], j+pocatek[1]);
12         }
13     }
14 }
```



Obrázek 40.3: Funkce `stroke()` a `point()`

40.4 Úsečka

K nakreslení úsečky se používá funkce `line(x1, y1, x2, y2)` se souřadnicemi jejího počátečního a koncového bodu. Vykresleme si barevnou škálu stupňů šedé pomocí úseček.

```
1  int delka = 100;
2  int[] pocatek = {20,20}; //pocatecni souradnice x, y
3
4  void setup(){
5      size(296,140);
6  }
7
8  void draw(){
9      for(int i = 0; i<256; i++){
```

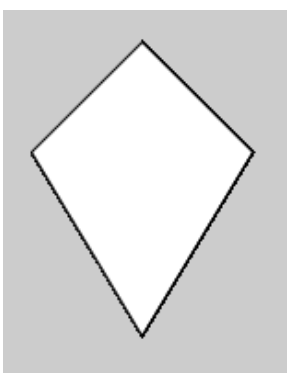
```
10          stroke(i);
11          line(pocatek[0] + i, pocatek[1], pocatek[0] + i, pocatek[1] + delka);
12      }
13  }
```

Obrázek 40.4: Funkce `stroke()` a `fill()`

40.5 Čtyřúhelník

Čtyřúhelník je geometrický útvar daný pomocí čtyř bodů. V Processing se nakreslí pomocí funkce `quad(x1, y1, x2, y2, x3, y3, x4, y4)`, kde `x1` – `y4` jsou souřadnice krajních bodů. Funkci si předvedeme na deltoidu (tvar klasického draka).

```
1  void setup(){
2      size(200, 300);
3  }
4
5  void draw(){
6      quad(40,100,100,40,160,100,100,200);
7  }
```

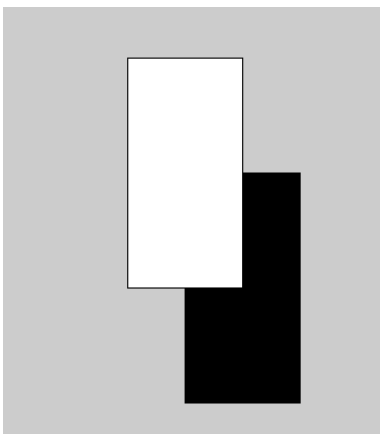


Obrázek 40.5: Funkce `quad()`

40.6 Zvláštní případy čtyřúhelníků

Pro čtyřúhelníky, jejichž dvě protější strany jsou stejně dlouhé (čtverec, obdélník), existuje speciální funkce `rect()`. Ta vytvoří čtyřúhelník z jeho rozměrů a souřadnic určujícího bodu. Tento bod může být dvojího druhu: buďto střed útvaru, nebo jeho levý horní roh. Ve výchozím stavu je jako určující bod nastaven právě roh útvaru, ale pomocí funkce `rectMode()` se dá změnit. Pokud ji voláme s parametrem `CORNER` nebo `CORNERS`, nastaví se určující bod na roh, pokud je parametr `CENTER` nebo `RADIUS`, bude jím střed útvaru.

```
1  void setup(){
2      size(500,500);
3  }
4
5  void draw(){
6      fill(0);
7      rectMode(CORNER);
8      rect(250,250,100,200);
9
10     fill(255);
11     rectMode(CENTER);
12     rect(250,250,100,200);
13 }
```



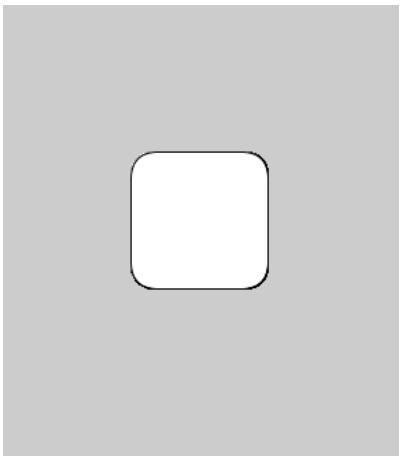
Obrázek 40.6: Funkce `fill()` a `rect()`

Funkce `rect()` může mít ještě další parametry, které určují zaoblení rohů. Přidáme-li funkci pátý parametr, všechny rohy se zaoblí tak, že jejich poloměr bude zadaný parametr.

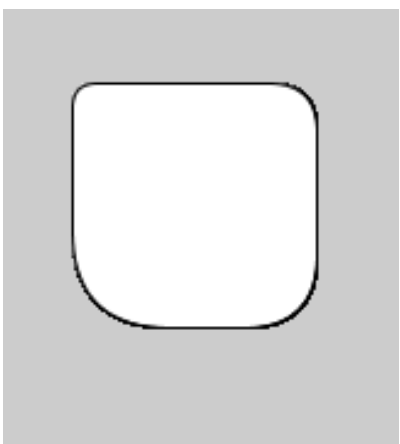
```
1  void setup(){
2      size(500,500);
3  }
4
5  void draw(){
6      fill(255);
7      rectMode(CENTER);
8      rect(250, 250, 100, 100, 20);
9  }
```

U čtyřúhelníku můžeme nastavit i poloměry jednotlivých zaoblených rohů zvlášť. Funkce tedy bude mít osm parametrů: `rect(a, b, c, d, hl, hp, dp, dl)`, kde parametry `a` – `d` jsou stejné jako při volání základní funkce `rect()` a `hl` je poloměr horního levého rohu, `hp` horního pravého, `dp` dolního pravého a `dl` dolního levého rohu.

```
1  void setup(){
2      size(500,500);
3  }
4
5  void draw(){
6      fill(255);
7      rectMode(CENTER);
8      rect(250, 250, 100, 100, 10, 20, 30, 40);
9  }
```



Obrázek 40.7: Funkce `rect()`



Obrázek 40.8: Funkce `rect()`

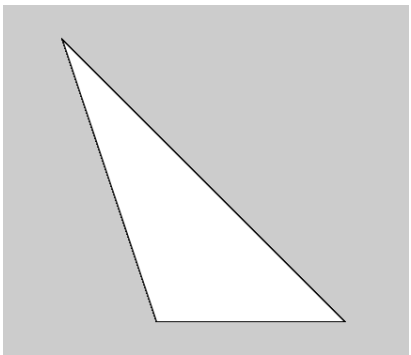
40.7 Trojúhelník

Dalším představeným objektem je trojúhelník. Používá se stejně jako `quad()`, jen má namísto osmi pouze šest parametrů (x a y tří bodů). K jeho nakreslení se používá funkce `triangle(x1,y1,x2,y2,x3,y3)`.

```
1 void setup(){
2     size(500,500);
3 }
4
5 void draw(){
6     fill(255);
7     triangle(100,100,400,400,200,400);
8 }
```

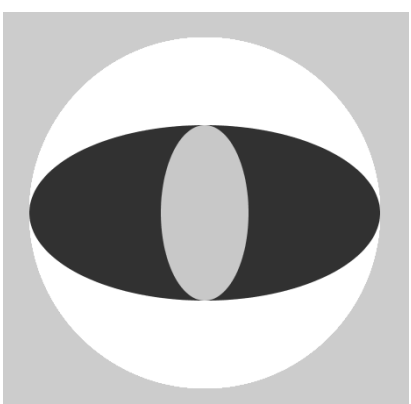
40.8 Elipsa a kružnice

Tímto jsme vyčerpali paletu základních n-úhelníků a dostáváme se k elipse, kružnici a kruhu a jejich částem. V Processing se využívá stejná funkce pro kružnici i elipsu a to `ellipse(x,y,sirka,vyska)`. Liší se pouze použitými parametry (když `sirka` = `vyska`, výsledný tvar je kružnice). Parametry `x` a `y` jsou souřadnice středu elipsy.



Obrázek 40.9: Funkce `triangle()`

```
1  void setup(){
2      size(500,500);
3  }
4
5  void draw(){
6      noStroke();
7
8      fill(color(255));
9      ellipse(250,250,400,400);
10
11     fill(color(30,50,100));
12     ellipse(250,250,400,200);
13
14     fill(color(200));
15     ellipse(250,250,100,200);
16 }
```

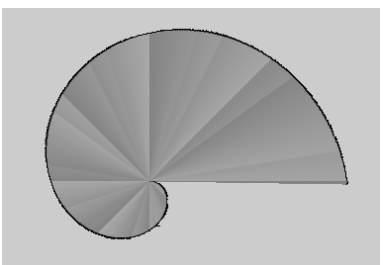


Obrázek 40.10: Funkce `ellipse()`

40.9 Část kružnice a elipsy

Pro práci s částmi kružnice slouží funkce `arc(x,y,sirka,vyska,p,k)`. První čtyři parametry jsou stejné, jako u `ellipse()`. Jsou to `x` a `y` středu, šířka, výška. Další dva parametry jsou počátek výřezu a konec výřezu v radiánech. Pro tento účel zde nalezneme konstanty `PI`, `HALF_PI` a `QUARTER_PI` odpovídající částem Ludolfova čísla π.

```
int casti = 400;

void setup(){
    size(500,500);
}

void draw(){
    for(int i=1; i<=casti; i++){
        fill(color(i%30,i%50+200,i%40+100));
        arc(250,250,400/casti*i, 400/casti*i, PI*2*i/casti,
            PI*2*i/casti+PI*2/casti);
    }
}
```



Obrázek 40.11: Funkce `arc()`

V referenci jazyka Processing[24] *nalezneme i funkce pro práci s křivkami (Curves). Těmi se ale v této knize nebudeme zabývat.*

Kapitola 41
Interakce s myší

Processing umí reagovat na akce myši i klávesnice. Může například zjišťovat polohu kurzoru, stisk tlačítek nebo rolování kolečka. K tomuto účelu slouží řada funkcí a proměnných. Některé z nich si nyní představíme. Ještě předtím si ale musíme říct, že se zde s funkcemi nepracuje úplně stejně jako u Arduina. Některé funkce musejí být volány stejným způsobem, jako bychom je znovu deklarovali. Mezi ně patří všechny funkce pro práci s myší.

Ve skutečnosti je mechanismus takový, že program čeká na akci myši či klávesnice a volá tyto funkce. Jedná se tedy o tzv. callback funkci.

41.1 Tlačítka myši

Základní funkce, která zjišťuje, jestli bylo vůbec nějaké tlačítko stisknuto, je funkce `mouseClicked()`. Jak už jsme naznačili, musí být tato funkce ve vlastním bloku kódu na úrovni void `draw()` a dalších. Mějme na paměti, že se funkce provádí při zmáčknutí a uvolnění tlačítka (nestačí tedy pouze stisknutí).

```
void setup(){}
void draw(){ }

void mouseClicked(){
    //kod, ktery se provede, kdyz je stisknute libovolne tlacitko mysi
}
```

Informace, jestli je stisknuté nějaké tlačítko, nám ale většinou nestačí. Ke zjištění, jaké tlačítko bylo zmáčknuto, slouží proměnná `mouseButton`. Ta obsahuje informaci o stisknutém tlačítku. Její hodnoty mohou být `LEFT`, `RIGHT` a `CENTER`.

[24] Reference jazyka Processing – `https://processing.org/reference/`

```
int casti = 400;
int vypln = 0;

void setup(){
    size(500,500);
}

void draw(){
    for(int i=1; i<=casti; i++){
        fill(vypln);
        noStroke();
        arc(250,250,400/casti*i, 400/casti*i, PI*2*i/casti,
            PI*2*i/casti+PI*2/casti);
    }
}

void mouseClicked(){
    if(mouseButton == LEFT){
        vypln = 255;
    }
    else{
        vypln = 0;
    }
}
```

Nalezneme zde další funkce, které s tlačítky myši souvisí. Jejich použití je stejné jako u `mouseClicked()`. První z nich je funkce `mouseDragged()`, která se vykoná v případě, že je stisknuté tlačítko myši a zároveň se myš pohybuje. Další funkcí je `mouseMoved()`, která se vykoná, pokud není zmáčknuto žádné tlačítko a myš se hýbe. Funkce `mousePressed()` se provede, pokud je stisknuto libovolné tlačítko a funkce `mouseRelased()` se provede, pokud je uvolněno. Poslední funkcí je `mouseWheel()`, která je volána, pokud rolujeme kolečkem myši.

41.2 Poloha kurzoru

Pro zjištění polohy kurzoru se zde nepoužívají funkce, ale proměnné, které obsahují jeho aktuální souřadnice v osách x a y. Jsou to `mouseX` a `mouseY`. Ukažme si program, který vykreslí úsečku danou jedním pevným bodem a kurzorem myši.

```
void setup(){
    size(500,500);
}

void draw(){
    background(200);
    line(250,250,mouseX,mouseY);
}
```

Možná si říkáte, proč se v knize o Arduinu bavíme o softwarovém prostředí, které s ním očividně moc nesouvisí. Vězte však, že Processing umí komunikovat se sériovou linkou a tím třeba i číst informace, které mu posílá Arduino. Celkem jednoduchým způsobem se tak dají vytvářet zajímavé grafy pomocí dat naměřených Arduinem v prostředí Processing.

Část XI

Arduino a Processing

Prostředí Processing jsme si již představili. I dále se jím budeme zabývat a ukážeme si, jak může komunikovat s Arduinem. Na příkladech si předvedeme funkce pro zpracování dat ze sériové komunikace.

Kapitola 42
Úvod

Jak už asi mnohé napadlo, komunikace mezi Processingem a Arduinem bude probíhat po sériové lince. Nabízí se nám ale dva možné způsoby ovládání Arduina. Pro tento účel totiž existuje speciální firmware `Firmata`. Ten se nahraje do Arduina a poté se stará o veškeré jeho ovládání. Následně stačí přidat do Processing knihovnu `Arduino` a začít programovat. Tato cesta má tedy výhodu v tom, že odpadá nutnost pracovat ve dvou prostředích najednou a tvořit tak souběžně dva programy pro Arduino a Processing. Druhou cestou je naprogramovat si Arduino podle svých představ, odesílat z něj data a ty poté zpracovávat v Processing. Tento způsob je sice pomalejší, ale neomezuje nás v použití celého arzenálu Arduina. My si obě cesty představíme.

Kapitola 43
`Firmata`

43.1 Nastavení

V první řadě musíme nastavit Arduino tak, aby mohlo s Processing komunikovat. Tato část je velmi jednoduchá – *Firmata* totiž IDE obsahuje už od stažení. V *Examples* rozbalíme nabídku *Firmata* a otevřeme program *StandardFirmata* a nahrajeme ho do Arduina. Tím jsme vyřešili veškeré nastavování na straně Arduina. Jednoduché, že?

Poté přichází na řadu knihovna pro Processing. Tu stáhneme zde[25] ve formátu ZIP. Stažený archiv rozbalíme a složku `Arduino` umístíme do *Dokumentů* → *Processing* → *Libraries*. Pokud právě prostředí běží, změny se projeví až po jeho restartu. Jako ukázkový příklad si v Processing otevřeme *Examples* → *Contributed Libraries Arduino (Firmata)* → *arduino_input*, popřípadě *arduino_input_mega* (pokud máme Arduino Mega). Tento příklad sleduje všechny digitální i analogové piny. Pokud je na nějakém hodnota `HIGH`, vybarví se jemu odpovídající čtvereček v Processing programu. U analogových pinů zkoumá na nich naměřenou hodnotu a vykresluje kružnice podle její velikosti.

43.2 Propojení

V Processing si vytvoříme nový program. Hned na začátku musíme určit, jaké doplňkové knihovny budeme potřebovat. V našem případě se jedná o knihovnu, která umí pracovat se sériovou linkou (ta je v Processing od začátku) a knihovnu pro práci s Arduinem (tu, kterou jsme před chvílí přidali). V dalším kroku vytvoříme prázdnou proměnnou pro naše Arduino, se kterým budeme dále pracovat. Všimněme si, že slovo Arduino při vytváření objektu zde funguje stejně jako u datových typů. Základní nastavení si ukažme na příkladu.

```
1  //vlozeni knihoven
2  import processing.serial.*;
3  import cc.arduino.*;
4
5  //vytvoreni promenne mojeArduino
6  Arduino mojeArduino;
7
8  void setup() {
9
10 }
11
12 void draw() {
13
14 }
```

[25] Processing Firmata – `http://playground.arduino.cc/uploads/Interfacing/processing2-arduino.zip`

Tímto jsme vytvořili základní objekt pro práci s Arduinem. V dalším kroku si vypíšeme všechna zařízení dostupná přes sériovou linku. Seznam těchto zařízení získáme funkcí `Arduino.list()`, která vrací aktivní sériové porty jako pole. Většinou však je v poli jediná hodnota a to právě naše připojené Arduino. Poté přiřadíme k proměnné `mojeArduino` nový objekt.

To provedeme pomocí `new Arduino(this, Arduino.list()[0], 57600)`, kdy první parametr je vždy `this`, druhým parametrem je sériový port s naším Arduinem (v tomto případě port s indexem 0) a třetím parametrem je rychlost sériové komunikace – zde 57 600 baud.

```
1 void setup() {
2     println(Arduino.list()); //vypise dostupna zarizeni
3     //vytvoreni noveho objektu
4     mojeArduino = new Arduino(this, Arduino.list()[0], 57600);
5 }
```

Nyní už máme Arduino i Processing připravené ke spolupráci a můžeme se pustit do programování. Sami zjistíte, že dostupné funkce jsou téměř totožné s těmi, které už známe.

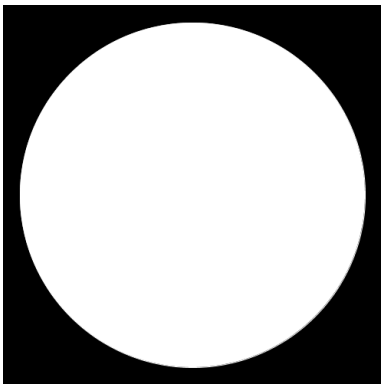
43.3 Programování

Knihovna `Arduino` přináší několik konstant. Patří tam: `Arduino.HIGH`, `Arduino.LOW`, `Arduino.OUTPUT` a `Arduino.INPUT`. Ty odpovídají konstantám `HIGH`, `LOW` a dalším v jazyce Wiring. Dále zde nalezneme nám známé funkce jako `pinMode()`, `digitalRead()`, `digitalWrite()`, `analogRead()` a `analogWrite()`. Ty se však musí psát s názvem vybraného Arduina a tečkou, například `mojeArduino.digitalRead(...)`. U analogových funkcí se navíc nepoužívá A při určování čísla pinu, ale pouze číslo. Jejich použití si ukažme na dvou příkladech.

První z nich pracuje s digitálními funkcemi. Při stisknutí tlačítka na pinu 2 se rozsvítí LED na 13 a zobrazí se bílý kruh na monitoru.

Zápis `mojeArduino.pinMode()` se musí použít, protože `mojeArduino` je instancí třídy `Arduino`. Příkaz `mojeArduino.pinMode()` je tedy volání metody objektu `mojeArduino`.

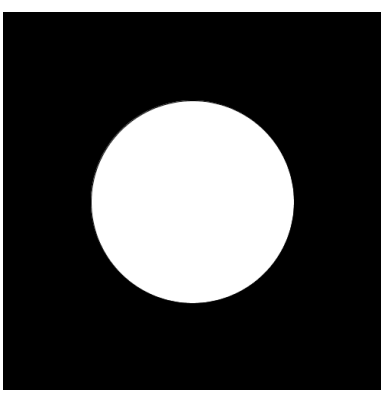
```
1  import processing.serial.*;
2  import cc.arduino.*;
3
4  //vytvoreni objektu mojeArduino
5  Arduino mojeArduino;
6  int led = 13;
7  int tlacitko = 2;
8
9  void setup() {
10     size(500,500);
11
12     println(Arduino.list()); //vypise dostupna zarizeni
13     //vytvoreni noveho objektu
14     mojeArduino = new Arduino(this, Arduino.list()[0], 57600);
15
16     mojeArduino.pinMode(tlacitko, Arduino.INPUT);
17     mojeArduino.pinMode(led, Arduino.OUTPUT);
18 }
19
20 void draw() {
21     background(0);
22     if(mojeArduino.digitalRead(tlacitko) == Arduino.HIGH){
23         fill(255);
24         stroke(255);
25         ellipse(250, 250, 450, 450);
26         mojeArduino.digitalWrite(led, Arduino.HIGH);
27     }
28     else{
29         mojeArduino.digitalWrite(led, Arduino.LOW);
30     }
31 }
```



Obrázek 43.1: Ovládání kruhu na obrazovce

V druhém programu si ukážeme použití analogových funkcí. Budeme číst hodnotu na pinu A0 a podle toho zapínat LED a také zvětšovat a zmenšovat kruh na monitoru.

```
1  import processing.serial.*;
2  import cc.arduino.*;
3
4  Arduino mojeArduino;
5  int led = 13;
6  int pot = 0;
7
8  void setup() {
9      size(500,500);
10
11     println(Arduino.list());
12     mojeArduino = new Arduino(this, Arduino.list()[0], 57600);
13 }
14
15 void draw() {
16     background(0);
17     int hodnota = mojeArduino.analogRead(pot);
18     ellipse(250, 250, hodnota/3, hodnota/3);
19     mojeArduino.analogWrite(led, hodnota/4);
20 }
```



Obrázek 43.2: Ovládání průměru kruhu pomocí potenciometru

Tímto jsme vyčerpali možnosti tohoto typu komunikace a dostáváme se ke zdlouhavějšímu, ale šikovnějšímu způsobu.

Schválně jsme nezmínili funkci `servoWrite()`. Servomotory jsme se totiž ještě nezabývali, ale vězte, že takováto funkce existuje. Více se o servech můžete dočíst v části Robotická ruka.

Asi vás už napadlo, že toto je cesta, jak se dá přesunout logická část programu z Arduina na výkonnější stroj (počítač). Podobné postupy se používají například při propojení Arduina a Raspberry Pi.

Kapitola 44
Vlastní způsob komunikace

Jak už jsme řekli v úvodu, druhý způsob je sice časově náročnější, zato nijak neomezuje funkce Arduina. Abychom mohli začít programovat, musíme se seznámit s funkcemi v Processing pro práci se sériovou linkou. Pro účely ukázek budeme používat Arduino Esplora, které jsme si představili na straně 79. Po malé úpravě ale budou použitelné pro jakékoliv Arduino.

44.1 Sériová komunikace

Stejně jako u příkladů s `Firmata`, i zde pracujeme s knihovnou `serial`, kterou vložíme pomocí příkazu `import processing.serial.*;`. Vytvoření proměnné pro port a přiřazení objektu je stejné jako u objektu `Arduino`. Také zde nalezneme funkci pro zobrazení dostupných zařízení, kterou je `Serial.list()`. Nový objekt vytvoříme příkazem `new Serial(this, Serial.list()[0], 57600)`.

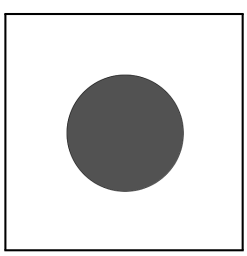
```
1  import processing.serial.*;
2
3  Serial port;
4
5  void setup() {
6      size(500,500);
7
8      println(Serial.list());
9      port = new Serial(this, Serial.list()[0], 57600);
10 }
11
12 void draw() {
13
14 }
```

Dále zde nalezneme funkci `available()`. Jedná se o velmi užitečnou funkci, která vrací počet čekajících bytů v zásobníku pro sériovou komunikaci. Používá se například, když chceme zjistit, jestli vůbec přišla nějaká data. Také se často používá funkce `clear()`, která zahodí všechna data v zásobníku. Tu použijeme, když se chceme pojistit, abychom nezačali číst nějakou delší informaci uprostřed. Pro přijímání dat použijeme `readStringUntil()`, která načte ze zásobníku řetězec od jeho začátku až po první výskyt ASCII znaku, který si zvolíme pomocí parametru funkce. Nejčastěji se používá znak zalomení řádku, který má ASCII hodnotu 10. Z Arduino Esplora si nechme posílat data ze slideru. Ty poté vypíšeme v konzoli a budeme vykreslovat kružnici s daným poloměrem. Musíme však pamatovat na to, že přijatá data jsou datového typu `string` a musíme je tedy před použitím převést na číslo. Do Esplory nahrajeme kód:

```
1  #include <Esplora.h>
2
3  void setup(){
4      Serial.begin(57600);
5  }
6
7  void loop(){
8      Serial.println(Esplora.readSlider());
9      delay(100);
10 }
```

Kód pro Processing může vypadat třeba takto:

```
import processing.serial.*;

Serial port;
String prijem;
int hodnota;
int eol = 10; //ASCII znak pro zalomeni radku

void setup() {
    size(512,512);

    println(Serial.list());
    port = new Serial(this, Serial.list()[0], 57600);
    port.clear();
}

void draw() {
    fill(color(100,50,200));
    while(port.available() > 0){
        prijem = port.readStringUntil(eol);
        if(prijem != null){ // pokud byla prijata nejaka data
            background(255);
            //funkce trim() orizne prebytecne bile znaky (mezery...)
            hodnota = int(trim(prijem));
            println(hodnota);
            ellipse(256,256,hodnota/2,hodnota/2);
        }
    }
}
```



Obrázek 44.1: Ovládání průměru kruhu sliderem

Tímto jsme si ukázali, jak číst data z Arduina a zpracovávat je v PC, a dostáváme se k odesílání informací z PC do Arduina. K tomu slouží funkce `write()`. Její parametr může být datového typu `byte`, `char`, `int`, `string` nebo pole bytů. Pokud si již nevzpomínáte, jak se se sériovou linkou pracuje u Arduina, můžete si to připomenout v sekci jí věnované na straně 47. V první ukázce Processing odešle Arduinu Esplora hodnotu červené barvy, podle které se poté rozsvítí LED na desce.

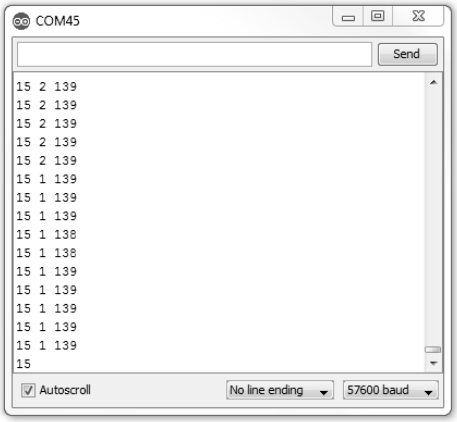
```
//Arduino program

#include <Esplora.h>

void setup(){
    Serial.begin(57600);
}

void loop(){
    if(Serial.available() > 0){
        Esplora.writeRGB(Serial.read(),0,0);
    }
}
```

```
1  //Processing program
2  import processing.serial.*;
3
4  Serial port;
5
6  void setup(){
7      size(512,512);
8
9      println(Serial.list());
10     port = new Serial(this, Serial.list()[0], 57600);
11
12     port.write(10);
13 }
14
15 void draw(){
16
17 }
```

Když potřebujeme odeslat více číselných informací najednou, je vhodné použít na straně Arduina funkci `Serial.parseInt()`, která vyhledá nejbližší vhodné číslo typu `integer` a vrátí jeho hodnotu. Poté stačí z Processing odeslat řetězec s čísly oddělenými například dvojtečkou a funkce se postará o zbytek. Na ukázce vidíte program, který podle přijatých hodnot bude nastavovat barvu RGB LED u Arduino Esplora.

```
1  //Arduino program
2  #include <Esplora.h>
3
4  int r=0, g=0, b=0;
5
6  void setup(){
7      Serial.begin(57600);
8  }
9
10 void loop(){
11     if(Serial.available() > 0){
12         r = Serial.parseInt();
13         g = Serial.parseInt();
14         b = Serial.parseInt();
15
16         Esplora.writeRGB(r,g,b);
17     }
18 }
```

```
1  //Processing program
2  import processing.serial.*;
3
4  Serial port;
5
6  void setup() {
7      size(512,512);
8
9      println(Serial.list());
10     port = new Serial(this, Serial.list()[0], 57600);
11
12     port.write("50:20:90");
13 }
14
15 void draw() {
16
17 }
```



Obrázek 44.2: Monitor sériové linky

Na závěr ještě zmíníme funkci `stop()`, která komunikaci se sériovou linkou ukončí.

```
1 port.stop()
```

Nepředstavili jsme si všechny funkce, ale jen ty nejužitečnější. Přehled všech funkcí naleznete v dokumentaci[26].

Kapitola 45
Příklad: Ovládání bodu joystickem

V prvním příkladu si ukážeme, jak se dá navigovat po plátně pomocí joysticku u Esplory. Budeme potřebovat:

- Arduino Esplora, nebo Arduino s připojeným joystickem či potenciometry.
- Processing.

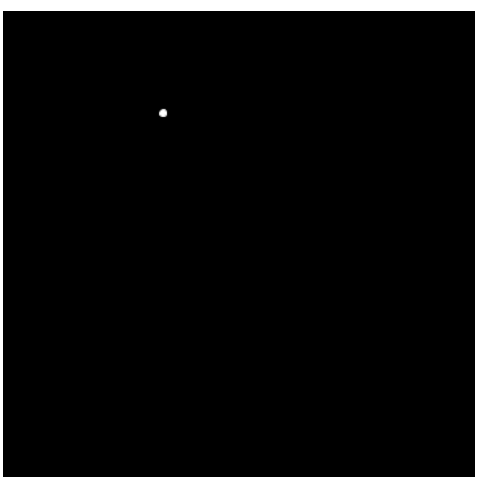
Z Arduina budeme odesílat dvě hodnoty oddělené mezerou – každou pro jednu osu. Poté budeme na obrazovce vykreslovat malou kružnici (kvůli viditelnosti není vhodné použít pouze jeden bod).

```
1  //Arduino program
2  #include <Esplora.h>
3
4  void setup(){
5      Serial.begin(57600);
6  }
7
8  void loop(){
9      Serial.print(Esplora.readJoystickX()+512);
10     Serial.print(' ');
11     Serial.println(Esplora.readJoystickY()+512);
12 }
```

```
1  //Processing program
2  import processing.serial.*;
3
4  Serial port;
```

[26] Dokumentace knihovny `Serial` – `https://www.processing.org/reference/libraries/serial/`

```
5  String x;
6  String y;
7  int intx, inty;
8  int eol = 10; //ASCII znak pro zalomeni radku
9  int sp = 32; //ASCII znak pro mezeru
10
11 void setup() {
12     size(512,512);
13
14     println(Serial.list());
15     port = new Serial(this, Serial.list()[0], 57600);
16     port.clear();
17 }
18
19 void draw() {
20    while(port.available() > 0){
21        fill(255);
22        stroke(255);
23        x = port.readStringUntil(sp); // prvni cislo konci mezerou
24        y = port.readStringUntil(eol); // druhe cislo konci zalomeni radku
25        if(x != null){ // pokud se povedlo nacist x
26            if(y != null){ //pokud se povedlo nacist y
27                background(0);
28                x = trim(x);
29                y = trim(y);
30                intx = 1024-int(x);
31                inty = int(y);
32                ellipse(intx/2,inty/2,5,5);
33            }
34        }
35    }
36 }
```



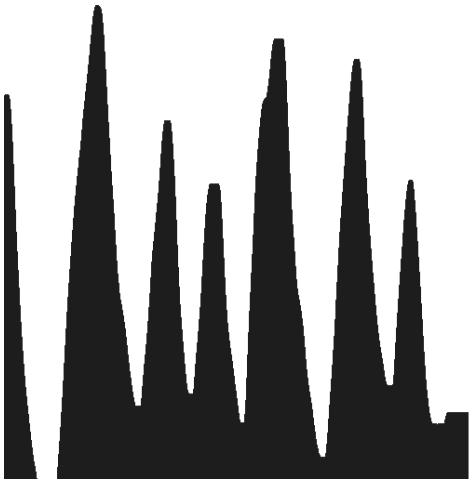
Obrázek 45.1: Pohyb bodu pomocí joysticku

Kapitola 46
Příklad: Graf hodnot ze slideru

V druhém příkladu si ukážeme, jak se dá v Processing vykreslovat graf z potenciometru. Potřebovat budeme stejné věci jako v předchozím příkladu.

```
1  //Arduino program
2  #include <Esplora.h>
3
4  void setup(){
5      Serial.begin(57600);
6  }
7
8  void loop(){
9      Serial.println(Esplora.readSlider());
10     delay(10);
11 }
```

```
1  //Processing program
2  import processing.serial.*;
3
4  Serial port;
5  String h;
6  int inth, i = 0;
7  int eol = 10; //ASCII znak pro zalomeni radku
8
9  void setup() {
10     size(512,512);
11
12     println(Serial.list());
13     port = new Serial(this, Serial.list()[0], 57600);
14     port.clear();
15
16     background(255);
17     fill(255);
18     stroke(color(0,0,255));
19 }
20
21 void draw() {
22     while(port.available() > 0){
23        if(i == 512){
24            background(255);
25            i = 0;
26        }
27        h = port.readStringUntil(eol);
28        if(h != null){
29           inth = int(trim(h));
30           line(i,512,i,512-inth/2);
31           i++;
32        }
33    }
34 }
```



Obrázek 46.1: Graf hodnot odeslaných ze slideru

Část XII

Sběrnice použitelné u Arduina

V této kapitole se budeme zabývat možnostmi připojení více desek Arduino k sobě a jejich vzájemné komunikace. Na začátku si předvedeme propojení přes sériovou linku, poté si ukážeme, jak k Arduinu připojit bluetooth modul. Nakonec si představíme sběrnici I2C.

Kapitola 47
Sériová linka

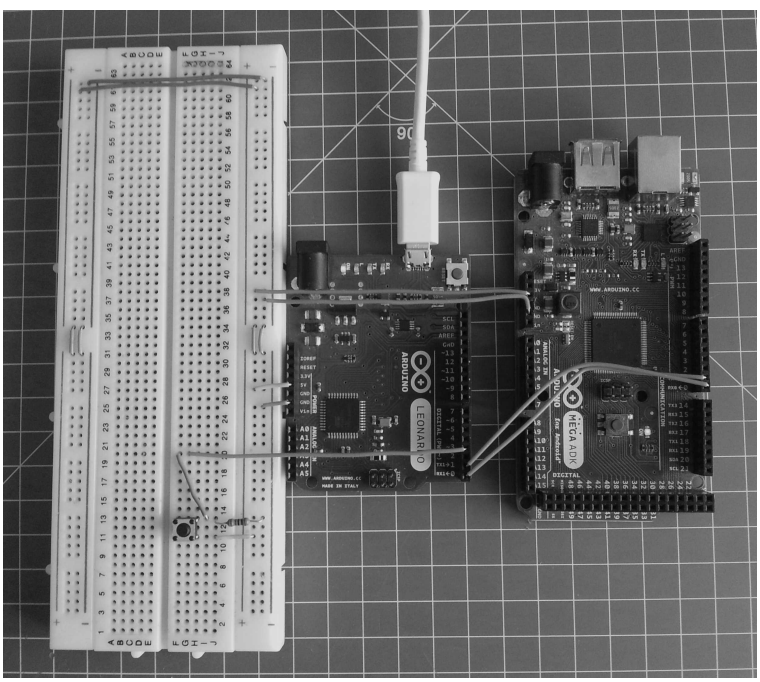
Většinu funkcí, které se pro sériovou komunikaci používají, jsme si popsali již dříve. V předchozí části jsme si je poté předvedli více prakticky ve spojení s prostředím Processing. Nyní si připojíme k sobě dvě Arduina (v mém případě Arduino Leonardo a Arduino Mega) a ukážeme si, jak spolu můžou komunikovat.

47.1 Propojení

V první řadě musíme Arduina správně spojit. To zde ale bude velmi jednoduché. Na obou deskách najdeme piny RX (anglicky *receive*, přijmout) a TX (anglicky *transmit*, odeslat). Tyto dva piny slouží právě pro sériovou komunikaci a za chvíli si ukážeme, jak sem připojit bluetooth modul. Desky k sobě propojíme tak, že RX jedné desky bude připojeno na TX druhé (jedna odesílá, druhá přijímá). Pozor! Jak už jsme zmínili dříve, některé typy desek mají více sériových portů. Ty jsou označeny RX0, RX1 atd. V tomto případě nezáleží na tom, jaké piny použijeme. Musíme ale použít vždy dvojici se stejným číslem. Také musíme mírně změnit kód. Pro piny RX0 a TX0 použijeme většinou funkce začínající `Serial.`, pro RX1 a TX1 to budou funkce `Serial1.` atd. V našem případě budeme využívat RX1 a TX1 u Leonarda a RX0 a TX0 u Arduino Mega. Většinou platí, že TX nalezneme na pinu 1 a RX na pinu 0. Pokud bude na zdroj připojena pouze jedna deska, musíme spojit 5V a GND obou desek.

Pozor! U Arduino Leonardo se situace ještě trochu komplikuje – má totiž od sebe oddělenou hardware a USB sériovou linku, která není fyzicky vyvedená. Pro komunikaci přes USB využijeme funkci začínající `Serial.`, pro hardware komunikaci využijeme `Serial1.` (náš případ).

Jelikož jsme si jednotlivé funkce popsali již v minulých dílech, přejdeme rovnou na příklad. Na Arduino Leonardo připojíme tlačítko. Pokud bude stisknuté, po sériové lince budeme odesílat hodnotu `H`, jinak odešleme hodnotu `L`. Jak už jsme napsali před chvílí – zapojení je velmi jednoduché. RX jedné desky připojíme na TX druhé a GND a 5V obou desek připojíme k sobě.



Obrázek 47.1: Propojení Arduina Leonardo s Arduinem Mega

Arduino, na kterém je připojeno tlačítko (u nás Leonardo), bude fungovat jako vysílač. Jeho úkolem bude zjišťovat, jestli je tlačítko stisknuté. Pokud bude, odešle po sériové lince funkcí `Serial.print()` znak `H`, v opačném případě odešle `L`.

```
byte tl = 3; //tlacitko

void setup(){
    pinMode(tl, INPUT);
    Serial1.begin(9600);
}

void loop(){
    if(digitalRead(tl) == HIGH){
        Serial1.print("H");
    }
    else{
        Serial1.print("L");
    }
    delay(100);
}
```

Druhé Arduino (zde Mega) bude mít za úkol přijímat zprávy ze sériové linky a pokud nalezne znak `H`, tak rozsvítí LED diodu. Ke zpracování dat použijeme funkce `Serial.available()` a `Serial.read()`.

```
void setup(){
    pinMode(13, OUTPUT);
    Serial.begin(9600);
}

void loop(){
    while(Serial.available()){
        char a = Serial.read();
        if(a == 'H'){
            digitalWrite(13, HIGH);
        }
        else if(a == 'L'){
            digitalWrite(13, LOW);
        }
    }
}
```

Další příklady by byly pouhé opakování věcí z předchozích částí. Pojďme se nyní podívat, jak může Arduino komunikovat s Bluetooth modulem.

47.2 Bluetooth

Naštěstí pro nás existuje celá řada modulů (a nejenom bluetooth), které jsou určeny pro ovládání přes sériovou linku. K jejich ovládání se většinou nepoužívá nic jiného, než nám známé funkce pro sériovou komunikaci. U některých si musíme dát pozor na napájení. Nemusí být totiž určeny na napětí 5 V a mohly by být poškozeny. To však neplatí o modulech přímo určených pro Arduino. Jedním z takovýchto bluetooth modulů je BlueSMiRF Silver, popřípadě o třídu vyšší BlueSMiRF Gold.

Modul zapojíme stejným způsobem jako Arduino v minulém příkladu.

Arduino	Modul
TX	RX
RX	TX
+5V	VCC
GND	GND

Tímto je modul zapojen a my se můžeme pustit do programování.

Obrázek 47.2: BlueSMiRF Silver [16]

47.2.1 Odeslání a zpracování dat z potenciometru

Na pin A0 si připojíme potenciometr. Na něm naměřené hodnoty budeme přes bluetooth odesílat do PC a pomocí Processing zpracovávat. PC identifikuje bluetooth zařízení stejným způsobem jako připojené Arduino. Náš modul tedy bude také připojen na nějaký COM port. Výchozí rychlost komunikace obou uvedených modulů je 115 200 bps. Program v Arduinu má velmi jednoduchou funkci – přečti hodnotu a pošli ji přes bluetooth do PC.

```
1  void setup(){
2      Serial.begin(115200);
3  }
4
5  void loop(){
6      Serial.println(analogRead(A0));
7      delay(10);
8  }
```

Před začátkem komunikace musíme PC s modulem spárovat. Zapneme bluetooth u PC a vyhledáme Bluetooth zařízení. Po nalezení našeho modulu jej přidáme. Pokud budeme vyzváni k zadání kódu, výchozí pin je 1234. Modul zabere v PC dva COM porty. V Processing musíme pomocí indexu u funkce `Serial.list()[]` vyzkoušet, jaký je ten správný. Kód pro vykreslování grafu je převzatý z příkladu na konci minulého dílu.

```
1  import processing.serial.*;
2
3  Serial port;
4  String h;
5  int inth, i = 0;
6  int eol = 10; //ASCII znak pro zalomeni radku
7
8  void setup(){
9      size(512,512);
10     println(Serial.list());
11     port = new Serial(this, Serial.list()[1], 115200);
12 }
13
14 void draw(){
15     while(port.available() > 0){
16         if(i == 512){
17             background(255);
18             i = 0;
19         }
20         h = port.readStringUntil(eol);
21         if(h != null){
22             inth = int(trim(h));
23             line(i,512,i,512-inth/2);
24             i++;
```

```
25              println(inth);
26          }
27      }
28  }
```

Může se stát, že program po připojení modulu k Arduinu nepůjde nahrát. Pokud totiž máme cokoliv připojeného na piny, pomocí kterých se Arduino programuje, může nám to rušit nahrávání programů. Stačí ale na dobu nahrávání odpojit modul od Arduina, čímž by se měl problém vyřešit.

Kapitola 48
Arduino a Android

Existuje celá řada aplikací pro Android, která je přímo určená pro práci s Arduinem. Velmi zajímavá je aplikace SensoDuino, která umí přes bluetooth odesílat data snad ze všech dostupných senzorů. Po instalaci aplikace se nám zobrazí následující rozhraní.

Zde si můžete vybrat, jaký senzor bude zapnutý a jestli se z něj budou odesílat informace. Data jsou vždy odeslána v pořadí: číslo senzoru (`integer`), číslo měření (`integer`) a tři naměřené hodnoty (`float`). Některé senzory odesílají pouze jednu hodnotu. V tom případě jsou zbylé dvě nastaveny na 99.99. Část senzorů odesílá tři hodnoty. Jde většinou o měřené hodnoty v jednotlivých osách. Tato čísla můžeme získat pomocí funkcí `Serial.parseInt()` a `Serial.parseFloat()` – tu jsme si ještě nepředstavili, ale funguje stejně jako `Serial.parseInt`, pouze čeká na číslo typu `float`. Ještě před tím ale musíme nastavit jinou rychlost komunikace s modulem. Výchozí rychlost přenosu je totiž s aplikací nekompatabilní. U představených modulů to provedeme příkazy `Serial.print("$")` a poté `Serial.println("U,9600,N")`. Pouhé vypsání hodnot zajistí program:

```
1  int a,b;
2  float c,d,e;
3
4  void setup(){
5      Serial.begin(115200);
6      // vychozi rychlost nasich modulu
7      // trikrat vypiseme dolar - tim se dostaneme do modu pro nastavovani
8      Serial.print("$");
9      Serial.print("$");
10     Serial.print("$");
11     delay(100);
12     // docasne nastavime rychlost modulu na 9600 bps
13     Serial.println("U,9600,N");
14     Serial.end();
15     Serial.begin(9600);
16 }
17
18 void loop(){
19     while(Serial.available() > 0){
20         a = Serial.parseInt(); //typ senzoru
21         b = Serial.parseInt(); //cislo mereni
22         c = Serial.parseFloat(); //prvni hodnota
23         d = Serial.parseFloat(); //druha hodnota
24         e = Serial.parseFloat(); //treti hodnota
25
26         Serial.println(a);
27         Serial.println(b);
28         Serial.println(c);
29         Serial.println(d);
30         Serial.println(e);
31     }
32 }
```

Obrázek 48.1: SensoDuino [40]

To není vskutku žádný div. Je to ale vše, co potřebujeme k dalšímu programování. Ještě si ale musíme uvést, jaké označení mají jaké senzory. V následujícím seznamu, převzatém z oficiální dokumentace aplikace, naleznete číslo typu senzoru, jednotky a další doplňkové informace.

1. ACCELEROMETER (m/s^2 – X, Y, Z).
2. MAGNETIC_FIELD (uT – X, Y, Z).
3. ORIENTATION (Yaw, Pitch, Roll).
4. GYROSCOPE (rad/sec – X, Y, Z).
5. LIGHT (SI lux).
6. PRESSURE (hPa millibar).
7. DEVICE TEMPERATURE (C).

8. PROXIMITY (Centimeters or 1, 0).
9. GRAVITY (m/s^2 – X, Y, Z).
10. LINEAR_ACCELERATION (m/s^2 – X, Y, Z).
11. ROTATION_VECTOR (Degrees – X, Y, Z).
12. RELATIVE_HUMIDITY (%).
13. AMBIENT_TEMPERATURE (C).
14. MAGNETIC_FIELD_UNCALIBRATED (uT – X, Y, Z).
15. GAME_ROTATION_VECTOR (Degrees – X, Y, Z).
16. GYROSCOPE_UNCALIBRATED (rad/sec – X, Y, Z).
17. SIGNIFICANT_MOTION (1, 0).
18. AUDIO (Vol.).
19. GPS1 (Lat., Long., Alt.).
20. GPS2 (Bearing, Speed, Date/Time).

Se získanými informacemi už je poměrně jednoduché například udělat z Arduina vozítko ovládané pomocí náklonu telefonu a podobně.

Pro hlubší nastavení modulu, jako je rychlost připojení, jméno nebo pin se používají tzv. `AT` *commands. Ty ale pro základní činnost nejsou potřeba. Jejich výpis naleznete například zde*[27].

Kapitola 49
Sběrnice I2C

I2C je sběrnice, která umožňuje mít na dvou vodičích (jeden pro udávání taktu a druhý pro data) připojeno až 128 zařízení (každé má svoji 7bitovou adresu). Vodič, který udává takt komunikace, se nazývá SCL (anglicky *clock line*). Druhý vodič přenášející data je označován SDA (anglicky *data line*). U tohoto typu komunikace je vždy jedno zařízení master (řídící) a ostatní jsou slave (řízená). Zařízení master nemusí mít definovanou adresu, kdežto slave musí. Tuto sběrnici je navíc možné provozovat jen na vyhrazených pinech. Piny, ke kterým se sběrnice připojuje, jsou u různých desek nastaveny různě. Přehled naleznete v následující tabulce:

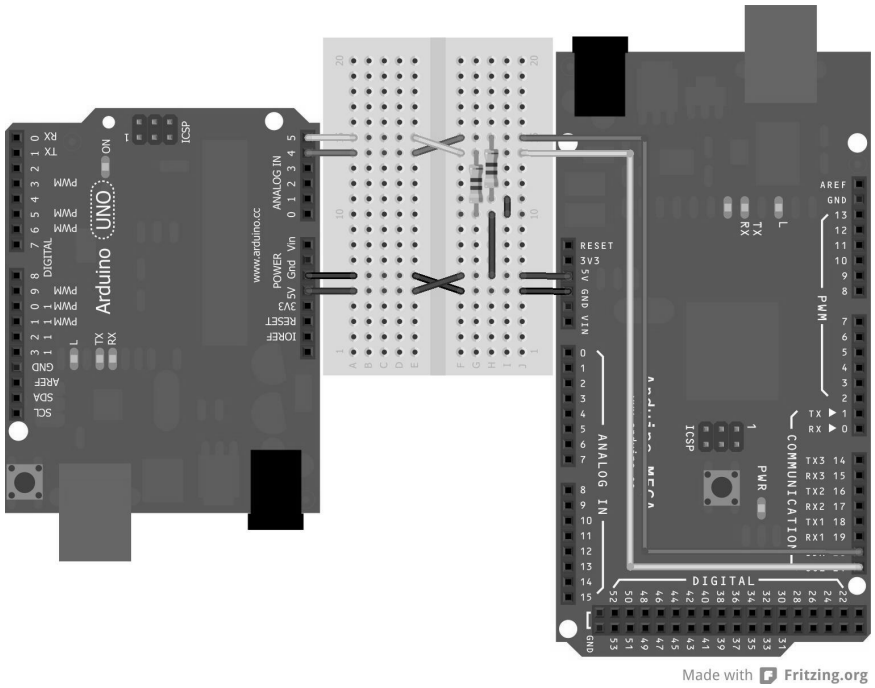
Deska	SDA pin	SCL pin
Uno, Ethernet	A4	A5
Mega 2560	20	21
Leonardo	2	3
Due	20, SDA1	21, SCL1

Ke správné funkci budeme potřebovat dva rezistory o odporu 4700 Ω. Jeden zapojíme mezi SDA a +5V a druhý mezi SCL a +5V.
Tím máme za sebou zapojení obou desek. Nyní už se můžeme podívat na software.

49.1 Funkce pro práci s I2C

Funkce, které si zde ukážeme, jsou velmi podobné těm, se kterými jsme se setkali u sériové komunikace. Na začátku programu musíme vložit knihovnu `Wire.h` pomocí příkazu `#include <Wire.h>`. Další funkce jsou pro přehlednost popsány v tabulce.

[27] `AT` příkazy – `https://www.sparkfun.com/datasheets/Wireless/Bluetooth/rn-bluetooth-um.pdf`



Obrázek 49.1: Zapojení I2C sběrnice

Funkce	Kdo používá	Popis
`Wire.begin(adr)`	master, slave	Připojí zařízení ke sběrnici. Pokud je zařízení master, nemusí mít žádný parametr. U slave je parametrem adresa zařízení (číslo mezi 0 a 127).
`Wire.beginTransmission(adr)`	master	Zahájí komunikaci se zařízením s adresou danou v parametru.
`Wire.endTransmission()`	master	Ukončí komunikaci se zařízením.
`Wire.requestFrom(adr,p)`	master	Touto funkcí si master vyžádá data od slave. Prvním parametrem je adresa slave zařízení. Druhým je počet bytů, které očekáváme.
`Wire.available()`	master, slave	Stejně jako u `Serial.available()` i tato funkce vrátí počet bytů čekajících na zpracování.
`Wire.write(data,delka)` `Wire.write(hodnota)`	master, slave	Tato funkce slouží k odeslání informací po I2C. Může mít buďto jeden parametr, a to číselnou hodnotu či řetězec, nebo dva parametry – pole bytů a jejich délku.
`Wire.read()`	master, slave	Používá se ke čtení hodnot z I2C.
`Wire.onReceive(fce)`	slave	Parametrem této funkce je jméno jiné (tzv. callback) funkce, která je volána, pokud jsou od zařízení master přijata nějaká data. Volaná funkce by měla mít jeden parametr a to počet přijatých bitů.
`Wire.onRequest(fce)`	slave	Zde je parametrem jméno funkce volané, když jsou od master požadována data.

Nyní si tyto funkce ukážeme v praxi.

49.2 Přenos master → slave

V tomto případě bude master číst data ze sériové linky. Pokud mu pošleme hodnotu `H` nebo `L`, pošle ji dále do slave zařízení (ale tentokrát přes I2C). Pokud slave obdrží hodnotu `H`, rozsvítí LED, v případě `L` ji zhasne.

Kód pro master zařízení.

```
1  //master
2
3  #include <Wire.h>
4
5  char a = ' ';
6
7  void setup(){
8      Wire.begin();
9      Serial.begin(9600);
10 }
11
12 void loop(){
13     while(Serial.available() > 0){
14         a = Serial.read();
15     }
16
17     if(a != ' '){
18         Wire.beginTransmission(100);
19         Wire.write(a);
20         Wire.endTransmission();
21         a = ' ';
22     }
23     delay(500);
24 }
```

Kód pro slave zařízení.

```
1  //slave
2
3  #include <Wire.h>
4
5  char a;
6
7  void setup(){
8      Wire.begin(100);
9
10     pinMode(13, OUTPUT);
11
12     Wire.onReceive(priPrijmu);
13 }
14
15 void loop(){
16     delay(100);
17     if(a == 'H'){
18         digitalWrite(13, HIGH);
19     }
20     else if(a == 'L'){
21         digitalWrite(13, LOW);
22     }
23 }
24
25 void priPrijmu(int b){
26     while(Wire.available() > 0){
27         a = Wire.read();
28     }
29 }
```

49.3 Přenos slave → master

V minulém příkladu bylo hlavním úkolem zařízení master odesílat data. V této ukázce bude mít úkol opačný, a to informace přijímat. Slave na vyžádání master zařízení změří hodnotu na A0. Tuto hodnotu ale musíme ještě před odesláním rozebrat na dva byty (odesílají se jednotlivě). Algoritmus rozkladu je stejný, jako kdybychom chtěli číslo převést z desítkové soustavy do „dvěstěpadesátšestkové“ soustavy. Dva odeslané byty by potom odpovídaly dvěma znakům z této soustavy reprezentující naši hodnotu. Více o číselných soustavách naleznete na Wikipedii[28]. Master si tedy vyžádá dva byty, které mu vzápětí přijdou. Následně je musí spojit, což probíhá obdobně jako u rozebírání, jen postup otočíme. Nakonec získanou hodnotu vypíšeme pomocí sériové linky.

Kód pro master zařízení.

```
1  //master
2
3  #include <Wire.h>
4
5  int hodnota;
6  byte mb[2];
7
8  void setup(){
9      Wire.begin();
10     Serial.begin(9600);
11 }
12
13 void loop(){
14     Wire.requestFrom(100, 2);
15     while(Wire.available() > 0){
16         mb[0] = Wire.read(); //nizsi byte
17         mb[1] = Wire.read(); //vyssi byte
18         //nyni hodnoty opet spojime dohromady
19         hodnota = mb[0] + mb[1]*256;
20         Serial.println(hodnota);
21     }
22     delay(1000);
23 }
```

Kód pro slave zařízení.

```
1  //slave
2
3  #include <Wire.h>
4
5  int mereni;
6  byte mb[2];
7
8  void setup(){
9      Wire.begin(100);
10     Wire.onRequest(odeslatData);
11 }
12
13 void loop(){
14     delay(100);
15 }
16
17 void odeslatData(){
18     mereni = analogRead(A0);
19     //nyni rozebereme hodnotu mereni na dva byty
20     //kolik celych 256 se vejde do namerene hodnoty
21     mb[1] = (mereni-(mereni%256))/256; //byte s vyssi hodnotou
22     //jaky je zbytek po deleni 256
23     mb[0] = mereni%256; //byte s nizsi hodnotou
```

[28] Wikipedia: Číselné soustavy – http://cs.wikipedia.org/wiki/Poziční_číselná_soustava

```
24      Wire.write(mb, 2);
25  }
```

Stejně jako u sériových modulů, i zde existuje celá řada součástek, vybavená možností komunikovat přes I2C. Patří sem například různé typy pamětí, převodníků, řadičů ale i displejů. Tím se nám otevírá celá škála dalších možností, co s Arduinem dělat.

Část XIII

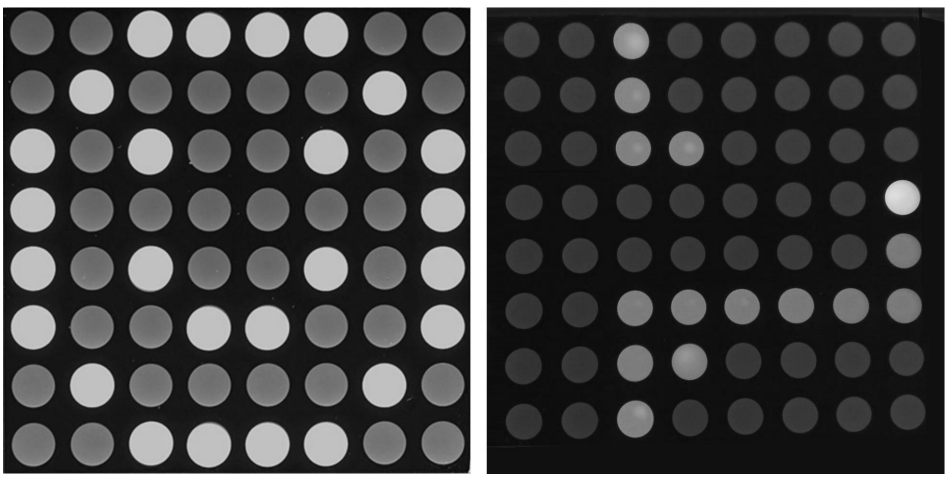
Arduino a displeje

V části o displejích si ukážeme různé způsoby grafického výstupu z Arduina. Nejdříve si předvedeme, jak používat maticové LED displeje. Také se zaměříme na platformu Rainbowduino a ukážeme si i pár příkladů využití. Nakonec se dostaneme i k LCD displejům.

Kapitola 50
Úvod

U velké části projektů se dostaneme do situace, kdy se nám hodí mít možnost něco zobrazit. Ať už jde o vykreslování grafů, zobrazení řetězce nebo celé grafické rozhraní programované aplikace či hry. Nepočítáme-li segmentové displeje, konstrukčně nejjednodušší jsou maticové LED displeje. Není to nic jiného, než vhodně rozmístěné a spojené LED.

Kapitola 51
Maticové LED displeje



Obrázek 51.1: Maticové displeje [35]

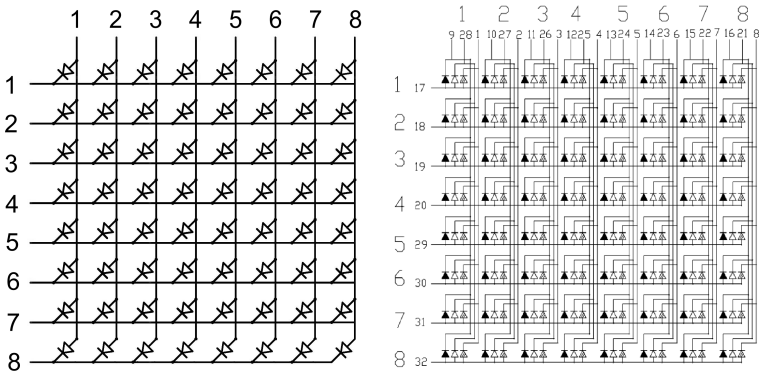
Tento typ displejů (anglicky *matrix display*) využívá k zobrazování klasické LED. Jedna dioda slouží většinou jako jeden pixel výsledného obrazu. Slovo matice v názvu napovídá, že budou srovnané do mřížky. V této mřížce mají vždy jeden vývod (anoda/katoda) společný pro řádek a druhý pro sloupec. Můžeme se také setkat s displeji, které mají více než jednu barvu. Potom může mít každá z LED diod i tři nebo více vývodů (většinou společná anoda, nebo katoda a pro každou barvu jeden zbývající vývod). Do matice jsou vývody diod spojeny stejným způsobem jako u jednobarevných, jenom buďto řádky, nebo sloupce mají více vývodů (pro každou barvu jeden). Také se můžeme setkat s různou velikostí displejů. Časté jsou k vidění velikosti 5×7, 8×8 a další.

51.1 Teorie řízení

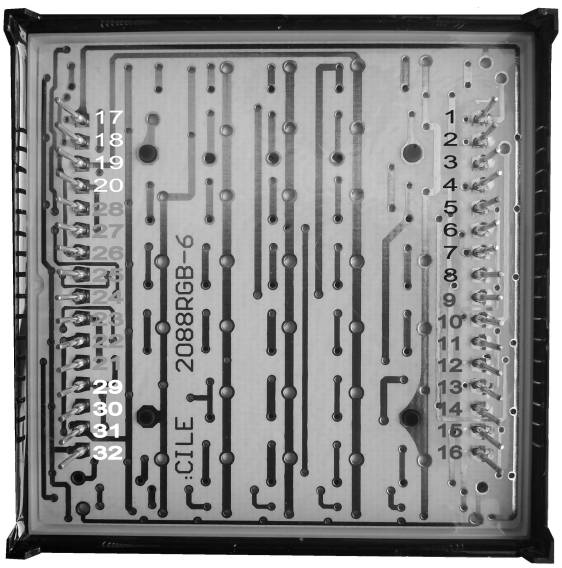
Schéma jednobarevného a RGB maticového displeje je na další straně. Rozložení LED je stejné jako při pohledu na displej zepředu.

My budeme používat tento RGB maticový displej (obrázek č. 51.1), jehož schéma můžete vidět na obrázku č. 51.2. Na začátek si ukážeme, jak pracovat pouze s jednou barvou a poté si vysvětlíme princip práce s celým RGB spektrem. Ze schématu je jasné, že náš displej má společnou anodu. Anody všech diod v jednom řádku jsou spojené. Sloupce tedy připadají na katody. Jelikož je náš displej RGB, má každý sloupec tři vodiče pro ovládání barev. V první části příkladu budeme pracovat pouze s jednou barvou (například zelenou). Na začátek se musíme podívat ještě na to, jak jsou piny rozmístěny ve skutečnosti. Na zadní straně našeho displeje nalezneme 32 pinů (8 pro každou barvu + 8 pro společnou anodu). Orientačním prvkem je číslo jedna natištěné u jednoho z rohových pinů.

Na obrázku můžete vidět popis pinů podle dokumentace.



Obrázek 51.2: Schéma mono a RGB maticového displeje



Obrázek 51.3: Piny RGB displeje

Než se pustíme do programování, musíme si trošku objasnit princip řízení. Pokud chceme rozsvítit LED na pozici [1,1], musíme připojit pin 17 na + a 1 na GND. Pokud bychom ale chtěli rozsvítit zároveň bod [1,1] a [2,2], je situace trošku komplikovanější. Kdybychom totiž připojili 17 a 18 na + a 1 i 2 na GND současně, rozsvítil by se nám celý čtverec ([1,1], [2,1], [1,2], [2,2]). Z tohoto důvodu probíhá ovládání tak, že se vždy pracuje jen s jednou řadou (ať už jde o řádek, nebo o sloupec), rozsvítí se všechny body, které se mají zobrazit, poté se napájení řady vypne a to samé se opakuje se všemi dalšími řadami. Pokud toto „překreslování“ probíhá dostatečně rychle, lidské oko si ničeho nevšimne (kvůli jeho setrvačnosti). Anody jsou vypnuté, pokud je na jejich pinu stav `LOW`, u katod je tomu naopak – vypnuté jsou při stavu `HIGH`.

51.2 Zapojení

Nyní se už můžeme pustit do spojení Arduina s displejem. V této části pracuji s Arduinem Leonardo. Budeme používat zelenou barvu RGB displeje. Také musíme použít 330 Ω rezistor na řádky, nebo sloupce (jedno z nich). Jaký pin zapojíme kam, je pouze na nás, jen tomu musíme přizpůsobit kód. My použijeme následující zapojení.

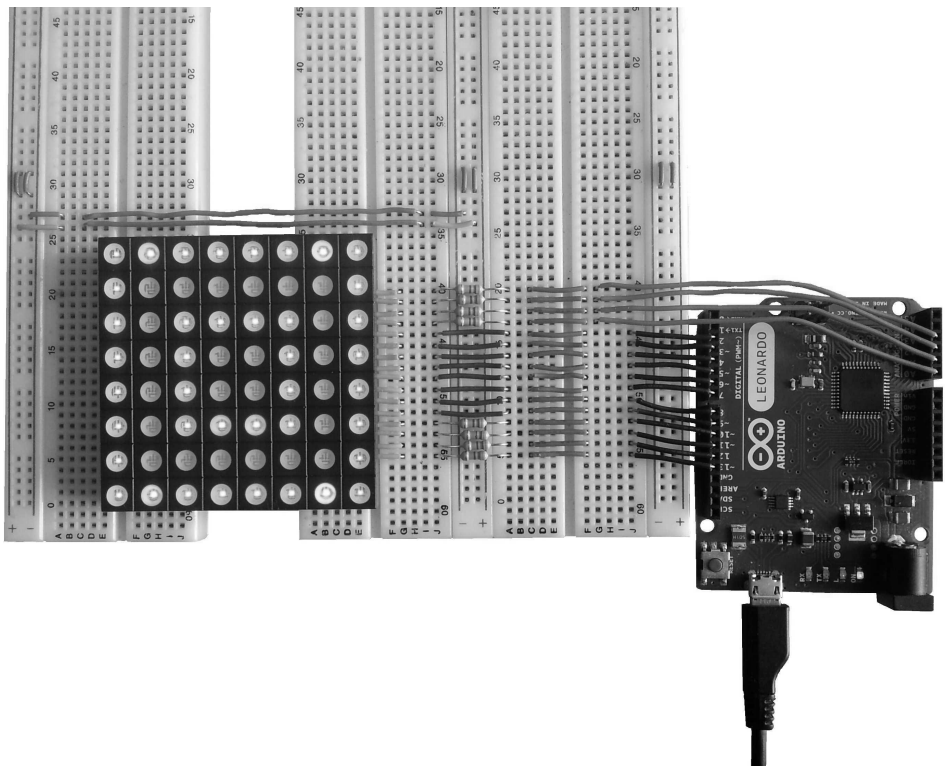
Číslo sloupce	Pin Arduina	Pin displeje	Číslo řádku	Pin Arduina	Pin displeje
1	2	28	1	A3	17
2	3	27	2	A2	18
3	4	26	3	A1	19
4	5	25	4	A0	20
5	6	24	5	10	29
6	7	23	6	11	30
7	8	22	7	12	31
8	9	21	8	13	32

51.3 Programování

Začneme tím, že si do prvních dvou polí (radky, sloupce) vypíšeme piny, na které jsou připojeny. Mějme na paměti, že se zde posunou souřadnice pinů kvůli indexování v poli od nuly. Bodu [1,1] na displeji budou tedy odpovídat `sloupce[0]` a `radky[0]`. Také si můžeme všimnout dvojrozměrného pole `obrazek[][]`, ve kterém jsou informace o obrázku, který budeme chtít zobrazit. Pokud má vybraná LED svítit, bude na jí odpovídajícím políčku 1. Princip by měl být zřejmý z komentářů v kódu.

```
byte sloupce[] = {2,3,4,5,6,7,8,9};
byte radky[] = {A3,A2,A1,A0,10,11,12,13};

//dvojrozmerne pole pro obrazek
byte obrazek[8][8] = {{1,1,1,1,1,1,1,1},
                      {1,0,0,0,0,0,0,1},
                      {1,0,1,1,1,1,0,1},
                      {1,0,1,0,0,1,0,1},
                      {1,0,1,0,0,1,0,1},
                      {1,0,1,1,1,1,0,1},
                      {1,0,0,0,0,0,0,1},
                      {1,1,1,1,1,1,1,1}};

void setup(){
    for(int i = 0; i < 8; i++){
        //nastavime piny
        pinMode(sloupce[i], OUTPUT);
        pinMode(radky[i], OUTPUT);

        //zajistime vypnuti displeje
        digitalWrite(sloupce[i], HIGH);
        digitalWrite(radky[i], LOW);
    }
}

void loop(){
    for(int i = 0; i < 8; i++){
        //zapneme radek i
        digitalWrite(radky[i], HIGH);

        //dale pracujeme s jednotlivymi sloupci
        for(int j = 0; j < 8; j++){
            //pokud je ve vybranem policku 1, rozsviti se LED
            if(obrazek[i][j] == 1){
                digitalWrite(sloupce[j], LOW);
            }
        }
        delay(1); //chvili pockame, aby byl obraz dostatecne jasny

        //vypneme vsechny sloupce
```

```
41          for(int j = 0; j < 8; j++){
42              digitalWrite(sloupce[j], HIGH);
43          }
44
45          //vypneme radek i
46          digitalWrite(radky[i], LOW);
47      }
48  }
```



Obrázek 51.4: Ukázka zapojení

Tímto jsme si představili práci s libovolným jednobarevným displejem. Co když ale chceme použít všechny dostupné barvy RGB displeje?

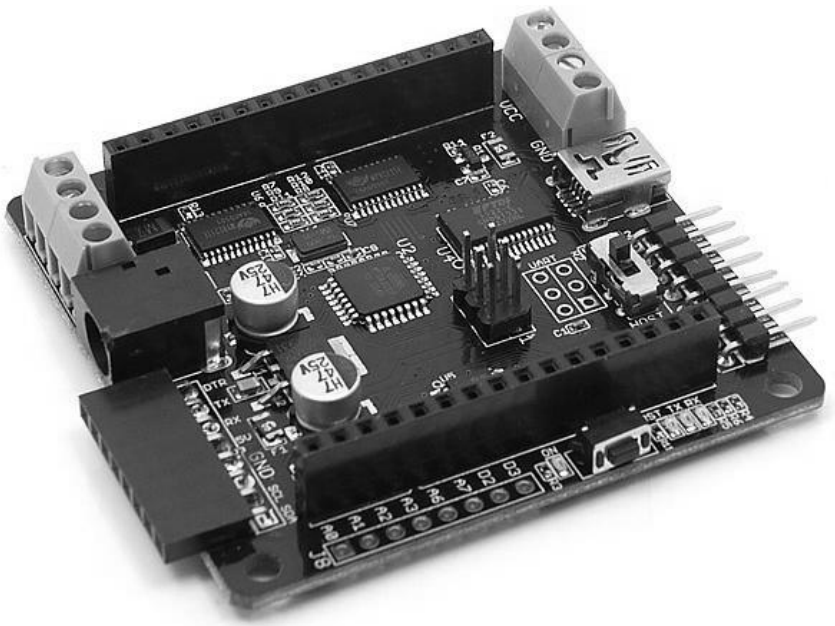
Kapitola 52
RGB teoreticky

Princip řízení zůstává stále stejný. Teď asi zklamu majitele menších desek. K ovládání totiž potřebujeme celkem 32 pinů. Takový počet ale najdeme jenom u větších desek (Mega, Due). Menší desky to zvládnou pouze s použitím nějakého přídavného hardware (shift registr, řadiče…). Pokud bychom k ovládání používali jenom funkce `digitalWrite()`, namícháme celkem 7 různých barev (dá se říci, že osm, když započítáme i stav, kdy LED nesvítí vůbec). Při použití funkce `analogWrite()` je teoretické maximum 256^3 barev, což je úctyhodných 16 777 216 možností. Musíme ale brát v úvahu možné nepřesnosti.

Také existuje celá řada displejů s řadičem, který umožňuje ovládání po sériové lince. Jedním z nich je například tento[29].

[29] Displej s řadičem – `https://www.sparkfun.com/products/760`

Kapitola 53
Rainbowduino



Obrázek 53.1: Rainbowduino [34]

S tématem maticových displejů velmi úzce souvisí Rainbowduino. Jedná se o klon Arduina, který je přímo určen k ovládání LED diod. Může ovládat buďto maticový displej 8×8, nebo dokonce RGB kostku 4×4×4 (oboje se společnou anodou). My si jej předvedeme se stejným displejem jako v předchozí části. V tomto případě nepotřebujeme žádný jiný hardware než Rainbowduino a displej. Nastavíme přepínač na desce do stavu USB a poté najdeme zdířku s označením 1 BLUE. Displej zasuneme do zdířky tak, aby byl v této zdířce zasunut pin displeje s označením 1.

53.1 Funkce

V první řadě musíme Arduino IDE naučit s deskou Rainbowduino pracovat. To uděláme jednoduše – stáhneme knihovnu[30] ze stránek výrobce a umístíme ji do složky `Libraries`. Do kódu ji poté vložíme známým příkazem `#include <Rainbowduino.h>`. Abychom nemuseli dělat vše manuálně, nabízí se nám několik funkcí.

Název	Popis
`Rb.init()`	Inicializuje driver pro Rainbowduino. Většinou je příkaz umístěn v `setup()`.
`Rb.setPixelXY(x,y,r,g,b)`	Nastaví barvu pixelu určeného souřadnicemi. Hodnoty souřadnic `x` a `y` mohou být v rozmezí 0 až 7. Když si desku natočíme tak, aby USB bylo dole, je bod `[0,0]` v levém spodním rohu.
`Rb.blankDisplay()`	Vypne všechny LED diody.

[30] Rainbowduino knihovna – `http://www.seeedstudio.com/wiki/images/4/43/Rainbowduino_for_Arduino1.0.zip`

Název	Popis
`Rb.drawChar(znak, x, y, RGBbarva)`	Vypíše na displej znak na daných souřadnicích. Barva musí být zadána ve 32bitovém formátu. To může být například 0xFF00FF, což je číslo v hexadecimální soustavě, kdy první dva znaky za 0x udávají hodnotu pro červenou, druhé dva pro zelenou a poslední dva pro modrou. Každý znak zabere obvykle pole 6×8, kdy souřadnice jsou udány pro pravý horní roh. V šířce znaku je započítána i jednosloupcová mezera před každým z nich. Pozor! Tato funkce má otočenou soustavu souřadnic o 90° proti směru hodinových ručiček.
`Rb.drawCircle(x,y, pol,RGBbarva)`	Nakreslí „kruh“ se středem v `[x,y]` o poloměru pol. Barva se zde zadává stejně jako u funkce pro zobrazení znaku.
`Rb.drawLine(x0, y0, x1, y1, RGBbarva)`	Vykreslí úsečku zvolené barvy z bodu `[x0,y0]` do bodu `[x1,y1]`.
`Rb.drawRectangle(x, y, vyska, sirka, RGBbarva)`	Zobrazí obdélník (nebo čtverec), kde bod `[x,y]` je pravý horní roh.
`Rb.fillRectangle(x, y, vyska, sirka, RGBbarva)`	Funguje stejně jako předchozí funkce, pouze bude výsledný tvar vyplněný.

Po připojení k PC spustíme IDE a v menu *Boards* nastavíme typ desky na *Arduino Duemilanove w/ ATmega 328*.

Jako ukázku použijeme příklad, který zobrazí na displeji znak zaslaný po sériové lince. Jeho barvu budeme generovat náhodně.

```
1  #include <Rainbowduino.h>
2
3  char a;
4
5  void setup(){
6      Rb.init();
7      Serial.begin(9600);
8  }
9
10 void loop(){
11     while(Serial.available() > 0){
12         a = Serial.read();
13         Rb.blankDisplay();
14         Rb.drawChar(a,0,1,random(0xFFFFFF));
15         delay(500);
16     }
17 }
```

Možná jste si také všimli, že se na jednom z okrajů desky nachází několik neosazených pinů (konkrétně D2, D3, A1, A2, A3, A6 a A7). To jsou plnohodnotné piny, které můžeme používat pomocí nám dobře známých funkcí. Další užitečnou věcí jsou piny určené k propojování více desek Rainbowduino k sobě. Z jejich popisků zjistíme, že se jedná o napájení, piny pro sériovou komunikaci a také piny I2C sběrnice (SDA, SCL).

53.2 Propojujeme Rainbowduina

I když by bylo možné komunikovat mezi deskami po sériové lince, my použijeme I2C sběrnici. Pro náš účel je vhodnější, protože umožňuje přímé adresování. Tentokrát nepotřebujeme jiný hardware než dvě (nebo více) desek. Princip si ukážeme na příkladu. V něm propojíme dvě desky Rainbowduino. Jedna z nich (master) bude přijímat zprávy po sériové lince a rozsvítí diodu s danými souřadnicemi a barvou. Displeje si umístíme tak, aby master byl dole a slave nad ním. Osa x bude mít tedy maximální hodnotu 7 a osa y 15. Adresu slave zařízení si nastavíme například na 100. Master bude po sériové lince čekat příkaz ve tvaru: `reset:x:y:r:g:b`, kdy `reset` nabývá hodnot 0 nebo 1, tedy například: `0:1:0:255:0:0`. V příkladu jsou použity uživatelem definované funkce `clearDisplay(adresa)` a `setPixel(adresa)`. První z nich odešle

příkaz ke zhasnutí všech diod na displeji dané adresy a druhá na displeji dané adresy nastaví barvu a pozici pixelu.

Kód pro master zařízení.

```
1  //master
2  #include <Wire.h>
3  #include <Rainbowduino.h>
4
5  int c,x,y,r,g,b;
6
7  void setup(){
8      Rb.init();
9      Serial.begin(9600);
10     Wire.begin();
11 }
12 void loop(){
13     while(Serial.available() > 0){
14         c = Serial.parseInt(); //maji se zhasnout dosud svitici LED?
15         x = Serial.parseInt();
16         y = Serial.parseInt();
17         r = Serial.parseInt();
18         g = Serial.parseInt();
19         b = Serial.parseInt();
20         if(x >= 8 || y >= 16 || r >= 256 || g >= 256 || b >= 256){
21             Serial.println("Chybna data, nebo neplatny rozsah.");
22         }
23         else{
24             if(c == 1){
25                 Rb.blankDisplay();
26                 clearDisplay(100);
27             }
28             if(y < 8){
29                 //toto se zobrazi na master zarizeni
30                 Rb.setPixelXY(x,y,r,g,b);
31             }
32             else if(y >= 8){
33                 //toto se musi pred zobrazenim odeslat do slave zarizeni
34                 //pred odeslanim jeste prevedeme souradnice na rozsah 0 - 7
35                 y = y%8;
36                 setPixel(100);
37             }
38         }
39     }
40 }
41
42 void clearDisplay(int addr){
43     Wire.beginTransmission(addr);
44     Wire.write(1); //pouze zhasne svitici LED
45     Wire.write(0);
46     Wire.write(0);
47     Wire.write(0);
48     Wire.write(0);
49     Wire.write(0);
50     Wire.endTransmission();
51 }
52
53 void setPixel(int addr){
54     Wire.beginTransmission(addr);
55     Wire.write(c);
56     Wire.write(x);
57     Wire.write(y);
```

```
58      Wire.write(r);
59      Wire.write(g);
60      Wire.write(b);
61      Wire.endTransmission();
62  }
```

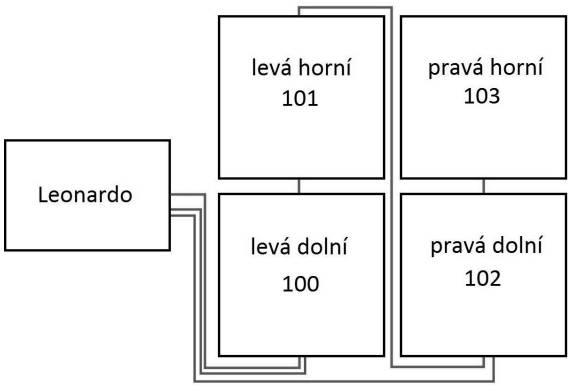
Kód pro slave zařízení.

```
1   //slave
2
3   #include <Wire.h>
4   #include <Rainbowduino.h>
5
6   byte c,x,y,r,g,b;
7
8   void setup(){
9       Rb.init();
10      Wire.begin(100);
11
12      Wire.onReceive(priPrijmu);
13  }
14
15  void loop(){
16      if(x >= 8 || y >= 16 || r >= 256 || g >= 256 || b >= 256){
17          Serial.println("Chybna data, nebo neplatny rozsah.");
18      }
19      else{
20          if(c == 1){
21              Rb.blankDisplay();
22          }
23          if(y < 8){
24              Rb.setPixelXY(x,y,r,g,b);
25          }
26      }
27  }
28
29  void priPrijmu(int c){
30      while(Wire.available() > 0){
31          c = Wire.read();
32          x = Wire.read();
33          y = Wire.read();
34          r = Wire.read();
35          g = Wire.read();
36          b = Wire.read();
37      }
38  }
```

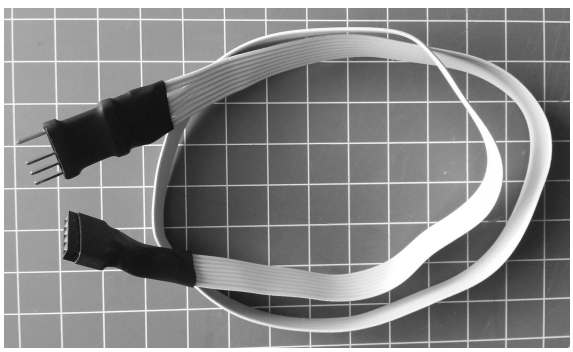
53.3 Zobrazení obrázku z PC

V této části propojíme Arduino Leonardo se čtyřmi displeji Rainbowduino. Na nich si poté zobrazíme obrázek, který odešleme z PC. Zpracování obrázku provedeme v prostředí Processing, kde ho rozkouskujeme na jednotlivé pixely a informace o nich odešleme po sériové lince do Arduina Leonardo. To bude mít za úkol informace zpracovat a rozeslat je do jednotlivých displejů po I2C sběrnici. Na obrázku č. 53.2 vidíte rozmístění displejů a jejich adresy.

V tomto příkladu budeme potřebovat ještě několik vodičů na spojení jednotlivých komponent. Nabízí se nám více způsobů propojení – první z nich je naznačený červenými liniemi (horní z Leonarda a mezi částmi). Leonardo je zde připojeno k jednomu sloupci a ten je připojen k dalšímu. Ve druhém způsobu (modře; dvě spodní linky z Leonarda) jsou s Arduinem spojeny oba sloupce. Tyto dva způsoby jsou funkčně ekvivalentní, ale praktičtější je ten druhý. Asi nejjednodušší způsob je si propojovací vodiče vyrobit. Nám se osvědčil plochý vícežilový kabel se zdířkami na jedné straně a piny na druhé. Pro I2C



Obrázek 53.2: Propojení Rainbowduin (modré jsou dvě spodní linie z Leonarda; ostatní jsou červené)



Obrázek 53.3: Propojovací kabel

komunikaci potřebujeme čtyři vodiče – SDA, SCL, +5V a GND. Výsledek může vypadat třeba takto, obrázek č. 53.3 – takovéto propojky budeme potřebovat dvě.

Teď už se můžeme pustit do programu pro Processing. Ten bude mít za úkol pomocí funkcí pro práci s obrázkem načíst vybraný obrázek a rozebrat ho na jednotlivé pixely. Z nich zjistit jejich barvu a souřadnice a vše odeslat po sériové lince. Odesílaná data mají stejnou podobu jako v minulém příkladu. Obrázek musí být 16×16 pixelů a může to být třeba obyčejný smajlík:

Princip jednotlivých částí je popsán v kódu.

```
1  import processing.serial.*;
2
3  PImage obr; //objekt pro ulozeni obrazku
4  int r,g,b,x,y, last;
5  Serial port;
6
7  void setup(){
8      if(Serial.list().length > 0){
9          println(Serial.list());
10         port = new Serial(this, Serial.list()[0], 19200);
11
12         obr = loadImage("7.jpg"); //nacte obrazek
13         obr.loadPixels(); //vytvori jednorozmerne pole pixelu
14
15         //pole obr.pixels obsahuje jednotlive pixely obrazku
16         if(obr.pixels.length != 256){ //test rozmeru
17             size(200,20);
18             text("Obrazek neni 16x16px.",10, height/2);
```

```
19          }
20          else{
21              while(millis() - last < 100){ //chvilku pocka
22              }
23              size(16,16);
24              last = millis();
25              for(int i = 0; i < obr.pixels.length; i++){
26                  //tyto tri funkce (red()...) nactou informace o jednotlivych barvach pixelu
27                  r = int(red(obr.pixels[i]));
28                  g = int(green(obr.pixels[i]));
29                  b = int(blue(obr.pixels[i]));
30
31                  //z indexu pixelu v poli urcime souradnice
32                  x = i % 16; //sloupec
33                  y = (i - i%16)/16; //radek
34
35                  y = 15 - y; //prevraceni osy y
36
37                  while(millis() - last < 5){ //chvilku pocka
38                  }
39
40                  port.write(0+":"+x+":"+y+":"+r+":"+g+":"+b+":"); //nakonec
                        odesleme retezec
41
42                  println(x+","+y+","+r+","+g+","+b);
43                  last=millis();
44              }
45              fill(0);
46              image(obr,0,0); //zobrazi obrazek i na PC
47          }
48      }
49      else{
50          println("Nenalezeno zadne zarizeni");
51      }
52  }
53
54  void draw(){
55
56  }
```

Data nám tedy už z PC chodí. Teď musíme zajistit, aby byla správně zpracována a rozeslána – to má u nás na starost Arduino Leonardo. Jistě si povšimnete, že je princip skoro stejný jako u master zařízení v minulém příkladu.

```
1   //master – Leonardo
2
3   #include <Wire.h>
4
5   int c,x,y,r,g,b;
6
7   void setup(){
8       Serial.begin(19200);
9       Wire.begin();
10  }
11
12  void loop(){
13      while(Serial.available() > 0){
14          c = Serial.parseInt(); //maji se zhasnout dosud svitici LED?
15          x = Serial.parseInt();
16          y = Serial.parseInt();
17          r = Serial.parseInt();
18          g = Serial.parseInt();
```

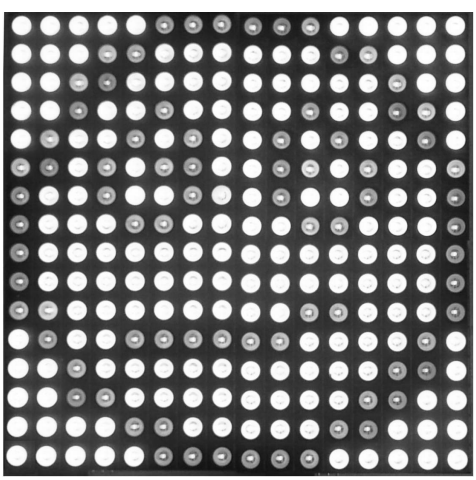
```
        b = Serial.parseInt();

        if(x >= 16 || y >= 16 || r >= 256 || g >= 256 || b >= 256){
            Serial.println("Chybna data, nebo neplatny rozsah.");
        }
        else{
            if(c == 1){
                clearDisplay(100);
                clearDisplay(101);
                clearDisplay(102);
                clearDisplay(103);
            }
            if(x < 8 && y < 8){
                //levy spodni
                setPixel(100);
            }
            else if(x < 8 && y >= 8){
                //levy horni
                y = y%8;
                setPixel(101);
            }
            else if(x >= 8 && y < 8){
                //pravy spodni
                x = x%8;
                setPixel(102);
            }
            else if(x >= 8 && y >= 8){
                //pravy horni
                x = x%8;
                y = y%8;
                setPixel(103);
            }
        }
    }
}
void clearDisplay(int addr){
    Wire.beginTransmission(addr);
    Wire.write(1); //pouze zhasne svitici LED
    Wire.write(0);
    Wire.write(0);
    Wire.write(0);
    Wire.write(0);
    Wire.write(0);
    Wire.endTransmission();
}

void setPixel(int addr){
    Wire.beginTransmission(addr);
    Wire.write(c);
    Wire.write(x);
    Wire.write(y);
    Wire.write(r);
    Wire.write(g);
    Wire.write(b);
    Wire.endTransmission();
}
```

Poslední a nejdůležitější částí jsou displeje. Kód všech bude téměř totožný – bude se lišit pouze v I2C adrese, kterou jim přidělíme podle předchozího obrázku.

```
//slave – Rainbowduino

```

```
3  #include <Wire.h>
4  #include <Rainbowduino.h>
5
6  int c,x,y,r,g,b;
7
8  void setup(){
9      Rb.init();
10
11     //tuto adresu musime nastavit zvlast pro kazdou desku
12     Wire.begin(100);
13
14     Wire.onReceive(priPrijmu);
15 }
16
17 void loop(){
18
19 }
20
21 void priPrijmu(int c){
22     while(Wire.available() > 0){
23         c = Wire.read();
24         x = Wire.read();
25         y = Wire.read();
26         r = Wire.read();
27         g = Wire.read();
28         b = Wire.read();
29     }
30     if(c == 1){
31         Rb.blankDisplay();
32     }
33     if(x < 8 && y < 8){
34         Rb.setPixelXY(x,y,r,g,b);
35     }
36 }
```

A výsledek by mohl vypadat třeba takto.

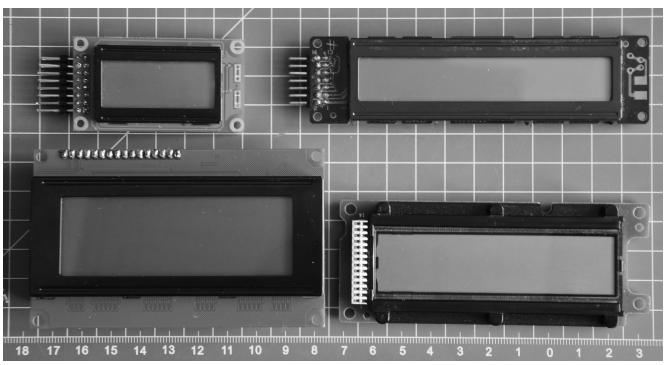


Obrázek 53.4: Výsledný obrázek

Kapitola 54
LCD displeje

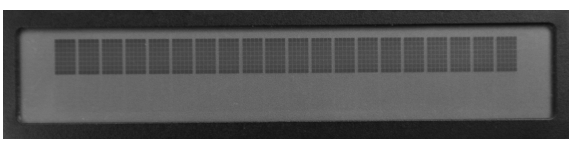
Jak už jsme zmínili v minulé kapitole, existuje celá řada LCD displejů. Všeobecně se užívají dva typy dělení. První z nich rozděluje LCD displeje do dvou skupin na znakové a grafické. Druhý z nich je dělí na barevné a monochromatické (jednobarevné). My si nyní jednotlivé skupiny představíme.

LCD displeje jsou většinou snazší na ovládání. Ovladač jim totiž zasílá pouze informace o tom, jaký znak mají kde zobrazit. Tyto znaky jsou předem definované – displej obsahuje základní „slovník“ znaků. Existuje celá řada velikostí displejů, které se však neudávají v počtech pixelů, ale v počtu řádků a míst pro znaky, kdy každé místo má na displeji přesně dané umístění. Můžeme se tedy běžně setkat s displeji 8×1 (jeden řádek s osmi znaky) až 40×4 (čtyři řádky po čtyřiceti znacích). Ovládání poté probíhá tak, že nastavíme kurzor na místo, na které chceme znak napsat a poté ho odešleme. Znakové displeje jsou většinou monochromatické. Můžeme se ale setkat s různými barvami podsvícení displeje.



Obrázek 54.1: LCD displeje

Pro lepší představu níže vidíme umístění míst pro znaky na displeji – každý obdélník odpovídá jednomu místu pro znak.



Obrázek 54.2: Maximální kontrast

54.1 Znakové LCD displeje

Pro práci se znakovými LCD displeji je Arduino vybaveno knihovnou, která je zahrnutá již v základním balíku IDE. Ta umožňuje ovládání všech znakových displejů, které jsou kompatibilní s řadičem Hitachi HD44780, což většina současných znakových displejů je. Tyto displeje mají většinou šestnáct pinů. V tomto příkladu budeme pracovat s tímto 20×4 LCD displejem[31]. Pokud se podíváme na jeho zadní stranu, nalezneme zde piny popsané 1 až 16. Pojďme si je nyní představit.

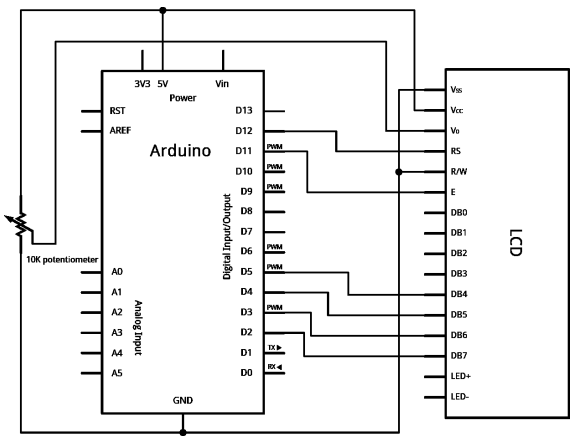
Číslo pinu	Symbol	Popis
1	VSS/GND	GND napájení displeje
2	VDD/VCC	+5V napájení displeje
3	V0	Pin pro nastavení kontrastu LCD (bude vysvětleno později)

[31] 20×4 LCD displej – `http://www.hwkitchen.com/products/a4x20-characters-lcd-display-kit/`

Číslo pinu	Symbol	Popis
4 – 6	RS, R/W, E	Řízení řadiče
7 – 14	DB0 – DB7	Datové piny
15	LED+	Anoda podsvícení displeje
16	LED−	Katoda podsvícení displeje

Některé displeje nemusejí mít podsvícení – piny 15 a 16 u nich tedy nenajdeme.

Displej zapojíme podle schématu. Také můžeme přes 10 kΩ rezistor připojit napájení k podsvícení. Potenciometr zde slouží k nastavení kontrastu displeje. Rezistor i potenciometr jsou součástí výše zmíněného LCD displeje.



Obrázek 54.3: Zapojení znakového LCD displeje [46]

Jak už jsme řekli dříve, obsahuje Arduino IDE pro komunikaci se znakovými LCD knihovnu. Ta má pro ovládání displeje několik funkcí. Použití některých z nich si ukážeme na příkladu.

Funkce	Popis
`LiquidCrystal lcd()`	Vytvoří objekt s názvem `lcd` pro práci s displejem. Jako parametry se udávají piny, na které je připojen displej. Více informací o různých kombinacích parametrů nalezneme v dokumentaci[32].
`lcd.begin(s,b)`	Zahájí práci s displejem. Parametry jsou: počet sloupců a počet řádků.
`lcd.clear()`	Tato funkce smaže všechny zobrazené znaky na displeji a nastaví kurzor do levého horního rohu.
`lcd.home()`	Nastaví kurzor do levého horního rohu.
`lcd.setCursor(s,r)`	Nastaví kurzor na danou pozici – sloupce, řádky.
`lcd.write(znak)`	Vypíše na displej jeden znak. Pozice kurzoru se posune o jedno místo doprava (v základním nastavení).
`lcd.print(data)`	Vypíše na displej řetězec nebo číslo. Pozice kurzoru se posune o počet zobrazených znaků doprava (v základním nastavení).
`lcd.cursor()`	Zobrazí na displeji pozici kurzoru podtržením znaku, na kterém je nastaven.
`lcd.noCursor()`	Skryje zobrazený kurzor.
`lcd.blink()`	Zobrazí blikající kurzor.
`lcd.noBlink()`	Skryje blikající kurzor.

[32] Dokumentace knihovny `LiquidCrystal` – `http://arduino.cc/en/Reference/LiquidCrystalConstructor`

Funkce	Popis
`lcd.noDisplay()`	Skryje všechny zobrazené znaky, ale nesmaže je. Komunikace s displejem nadále probíhá. Můžeme zapisovat znaky, které si displej pamatuje, jen je nezobrazí.
`lcd.display()`	Zobrazí vše, co bylo skryto funkcí `.noDisplay()` pokud mezitím došlo ke změně znaků na displeji, zobrazí se stav po změně.
`lcd.scrollDisplayLeft()`	Posune všechny zobrazené znaky o jedno místo doleva.
`lcd.scrollDisplayRight()`	Posune všechny znaky doprava.
`lcd.leftToRight()`	Nastaví automatický posun kurzoru po vypsání znaku doprava (což je výchozí stav).
`lcd.rightToLeft()`	Nastaví automatický posun kurzoru po vypsání znaku doleva.
`lcd.createChar(cislo, data)`	Tato funkce přináší možnost vytvoření vlastního znaku. Parametr `data` obsahuje informace o znaku (bude vysvětleno v příkladu). `Cislo` nám říká, pod jaké číslo se uloží do „slovníku“ znaků. To může nabývat hodnot 0 až 15. Pod tímto číslem jej poté můžeme pomocí funkce `.write()` zobrazit.

V prvním příkladu si vypíšeme na displeji počet vteřin od začátku běhu programu.

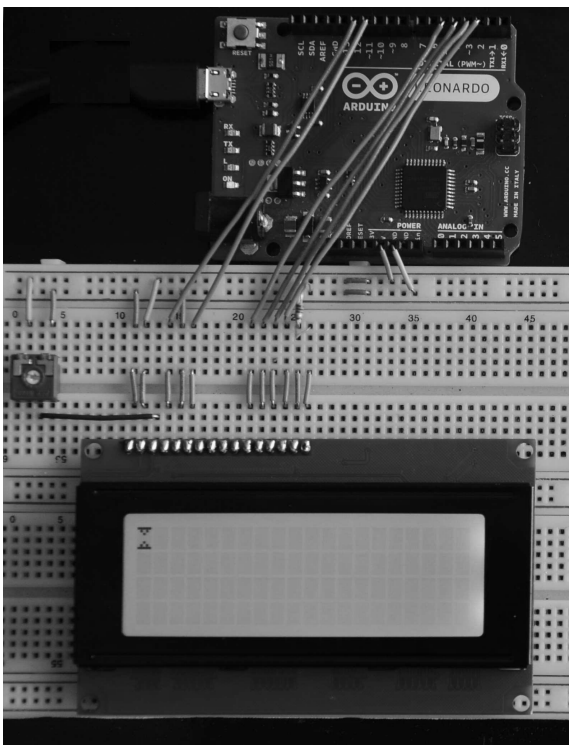
```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); //nastaveni pinu

long lastTime = 0;

void setup() {
    lcd.begin(20, 4); //inicializace displeje

    lastTime = millis();
}

void loop() {
    if(millis()-lastTime > 1000){
        lcd.clear();
        lcd.setCursor(9,1);
        lcd.print(millis()/1000);

        lastTime = millis();
    }
}
```

Nyní si ukážeme, jak se používá funkce `.createChar()`. Pod hodnotu 1 si uložíme znak dvou trojúhelníků. Námi definovaný znak se funkci předá jako jednorozměrné pole čísel (pro jednoduchou editaci v binární soustavě). V některých programovacích prostředích se pro zápis čísla ve dvojkové soustavě používá syntaxe 0b1111 (což odpovídá desítkové hodnotě 15). Arduino IDE však tento zápis neumožňuje. Pro pohodlnější práci s nižšími binárními čísly však v prostředí nalezneme několik předdefinovaných konstant začínajících velkým písmenem B. Konkrétně se jedná o rozsah hodnot od B00000000 (= 0) až B11111111 (= 255). Konstanta B00000010 tedy odpovídá hodnotě 2.

Můžeme si představit, že každý znak je vlastně malý maticový displej o rozměrech 5×8 bodů. My mu pomocí čísel ve tvaru Bxxxxxxxx posíláme vždy informaci o stavu jednoho celého řádku.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte znak[8] = {B11111,
                B01010,
                B00100,
                B00000,
```

```
                B00000,
                B00100,
                B01010,
                B11111};

void setup() {
    lcd.createChar(1, znak);
    lcd.begin(20, 4);
    lcd.write(1);
}

void loop() {}
```



Obrázek 54.4: Znakový LCD displej

Ve třetím příkladu se na displej vypíše text, který mu pošleme po sériové lince. K určení pozice na displeji slouží pomocná proměnná `i`. Pokud je celý displej zaplněn, smaže se a kurzor se vrátí do levého horního rohu.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int i = 0;
int x,y;

void setup() {
    Serial.begin(9600);
    lcd.begin(20,4);
}

```

```
13 void loop() {
14     while(Serial.available() > 0){
15         x = i%20;
16         y = (i-i%20)/20;
17 
18         lcd.setCursor(x, y);
19         lcd.write(Serial.read());
20         i++;
21         if(i == 80){
22             i = 0;
23             lcd.clear();
24         }
25     }
26 }
```

54.2 Grafické monochromatické LCD

Další skupinou displejů jsou grafické displeje, z nichž jednodušší jsou ty jednobarevné. My se jimi budeme zabývat pouze zběžně. Více se zdržíme až u barevných displejů. V této části si ukážeme použití displeje ATM12864D (128×64 pixelů) s Arduinem Mega. Tento displej obsahuje řadič KS0107B, k jehož ovládání je pro Arduino napsaná knihovna (i pro typ A a C). Tu stáhneme z archivu knihovny[33]. Dalším důležitým dokumentem je datasheet displeje[34], ve kterém najdeme funkce jednotlivých pinů. Posledním dokumentem, který budeme potřebovat, je dokumentace knihovny.

Displej s Arduinem propojíme podle následující tabulky. Pro jiné typy desek je zapojení popsáno v dokumentaci knihovny na straně 2.

Displej	Arduino Mega	Displej	Arduino Mega	Displej	Arduino Mega
1	GND	8	23	15	33
2	+5V	9	24	16	34
3	nepřipojeno	10	25	17	RESET
4	36	11	26	18	nepřipojeno
5	35	12	27	19	nepřipojeno
6	37	13	28	20	GND
7	22	14	29		

Piny 3, 18 a 19 nejsou připojeny k Arduinu. Zapojení pinů 3 a 18 a potenciometru je naznačeno na následujícím obrázku. Pin 19 je připojen na +5V přes 220 Ω rezistor.

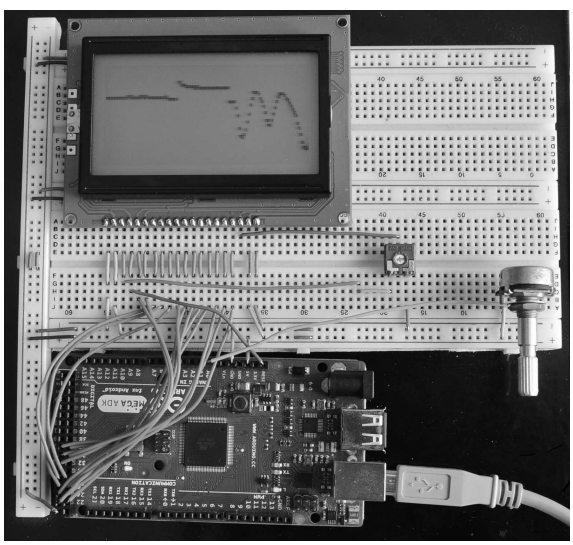
Obrázek 54.5: Zapojení potenciometru

Všechny funkce jsou přehledně popsány v již zmíněné dokumentaci. My si ukážeme, jak jednoduše na displej vykreslit graf hodnot naměřených na pinu A0.

```
1 #include <glcd.h>
2 
3 int data[128];
4 
5 void setup(){
```

[33] Knihovna displeje – `https://code.google.com/p/glcd-arduino/downloads/list`
[34] Datasheet displeje – `http://www.gme.cz/img/cache/doc/513/118/atm12864d-fl-ybw-datasheet-1.pdf`

```
6       GLCD.Init(); //inicializace displeje
7   }
8
9   void loop(){
10      data[0] = map(analogRead(A0), 0, 1023, 0, 63);
11
12      for(int i = 127; i > 0; i--){
13          bod(i,data[i]); // vykresli bod
14          data[i] = data[i-1]; // posune namerena data v poli
15      }
16
17      delay(40);
18
19      GLCD.ClearScreen();
20  }
21
22  void bod(int x, int y){
23      y = 63 - y; //prevraceni osy y
24      GLCD.SetDot(x,y, BLACK); //nastaveni bodu
25      GLCD.SetDot(x+1,y, BLACK);
26      GLCD.SetDot(x+1,y-1, BLACK);
27      GLCD.SetDot(x,y-1, BLACK);
28  }
```



Obrázek 54.6: Graf naměřených hodnot

54.3 Barevné grafické LCD

Nyní už se konečně dostáváme k barevným displejům. Představíme si dotykový displej s úhlopříčkou 2,8 palce a 320×240 pixely. Nebudeme se tedy zabývat pouze zobrazováním, ale i interakcí s dotykovou plochou. Ta nás bude zajímat jako první. Pro její funkci budeme potřebovat knihovnu Touch Screen Driver[35].

Vytvoříme si program, který nám odešle souřadnice právě zmáčknutého bodu. Na začátek programu vložíme kód, který za nás určí typ Arduina (v tomto případě Leonardo) a nastaví piny použité pro dotykovou vrstvu. Část kódu společně s vložením knihoven, vytvořením objektu displeje a kalibrací dotykové plochy vypadá následovně.

[35] Knihovna `TouchScreenDriver` – `https://github.com/Seeed-Studio/Touch_Screen_Driver`

Obrázek 54.7: Arduino a TFT shield

```
#include <stdint.h>
#include <SeeedTouchScreen.h>

#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__) // mega
#define YP A2
#define XM A1
#define YM 54
#define XP 57

#elif defined(__AVR_ATmega32U4__) //leonardo
#define YP A2
#define XM A1
#define YM 18
#define XP 21

#else
#define YP A2
#define XM A1
#define YM 14
#define XP 17

#endif

TouchScreen ts = TouchScreen(XP, YP, XM, YM);
```

Funkcí definic se nemusíme zabývat. Hlavní je, že nám tato část kódu zajistí správný chod displeje. Dalším krokem je kalibrace dotykové plochy. Na displeji je použita rezistivní technologie dotykové membrány. Její princip by se dal zjednodušeně popsat tak, že tlak na pružnou membránu se projeví určitým elektrickým odporem měřeným na této membráně. V ideálním případě by se například při tlaku v bodě `[10,20]` naměřil odpor na membráně měřící odpor v ose x $10\,\Omega$ a u y $20\,\Omega$. Naše plocha ale není takto ideální a musíme ji tedy nějakým způsobem zkalibrovat. Přibližné hodnoty maximální a minimální hodnoty odporu na osách vidíte v tabulce.

	Minimum	Maximum
x	232	1780
y	166	1826

V další části tedy musíme do kódu zahrnout údaje pro kalibraci.

```
int min_x = 232;
int max_x = 1780;
int min_y = 166;
int max_y = 1826;

int x, y;

void setup(){
    Serial.begin(9600);
}
```

V dalším kroku vytvoříme objekt pro bod, se kterým budeme dále pracovat. Jak se zachází s objekty, jsme si vyzkoušeli už při programování v Processing, takže to pro nás nebude žádné překvapení. Bod má tři vlastnosti: `x`, `y` odpovídající odporu na souřadnicích a `z`, která uchovává informaci o tlaku (s ní se dá poté do programu zakomponovat i citlivost dotyku). Pro získání souřadnic x a y musíme porovnat naměřené hodnoty s údaji pro kalibraci. Poté je už můžeme po sériové lince vypsat.

```
void loop(){
    Point p = ts.getPoint();

    x = map(p.x, min_x, max_x, 0, 239);
    y = map(p.y, min_y, max_y, 0, 319);

    if(p.z > 10){
        Serial.print(x);
        Serial.print(':');
        Serial.println(y);
    }

    delay(500);
}
```

Kalibrace dotykové plochy displeje se dá provést získáním naměřených hodnot pomocí funkce `.getPoint` *a postupným přejížděním os z minima do maxima. Poté stačí najít maximální a minimální hodnotu pro obě osy a změnit kalibrační proměnné.*

Už jsme zjistili, jak se dá z displeje zjistit souřadnice stlačeného bodu. Druhá (a asi důležitější) část je práce se samotnou zobrazovací plochou. I k tomuto účelu existuje pro náš displej knihovna, kterou stáhneme zde[36]. Ta nám přináší funkce pro jednodušší práci s displejem. Níže vidíte některé z nich.

Funkce	Popis
`Tft.TFTinit()`	Inicializuje displej.
`Tft.setPixel(x, y, barva)`	Vykreslí bod na daných souřadnicích.
`Tft.drawCircle(x, y, r, barva)`	Nakreslí kruh dané barvy se středem v `[x,y]` o poloměru `r`. Více o barvách dále.
`Tft.fillCircle(x, y, r, barva)`	Stejné jako předchozí funkce, pouze se zobrazí místo kruhu kružnice.
`Tft.drawLine(x1, y1, x2, y2, barva)`	Nakreslí úsečku dané barvy z bodu `[x1,y1]` do `[x2,y2]`.

[36] Knihovna `TFT Touch shield` – `https://github.com/Seeed-Studio/TFT_Touch_Shield_V2`

Funkce	Popis
`Tft.drawVerticalLine(x, y, d, barva)`	Nakreslí vertikální úsečku dané barvy s počátkem v bodu `[x,y]` o délce `d`.
`Tft.drawHorizontalLine(x, y, d, barva)`	Stejné jako u předchozí funkce, pouze bude výsledná úsečka horizontální.
`Tft.drawNumber(cislo, x, y, v, barva)`	Vypíše číslo typu `int` dané barvy na souřadnicích `x`, `y` o velikosti `v`.
`Tft.drawFloat(cislo, x, y, v, barva)`	Stejné jako u předchozí funkce. Zobrazí číslo typu `float`.
`Tft.drawRectangle(x,y,delka,vyska,barva)`	Zobrazí okraje obdélníku dané barvy s levým horním rohem v souřadnicích `x` a `y` o délkách hran `delka` a `vyska`.
`Tft.fillRectangle(x,y,delka,vyska,barva)`	Stejné jako předchozí funkce, pouze s tím rozdílem, že bude výsledný útvar vyplněný.
`Tft.fillScreen(x_levo, x_pravo, y_dole, y_nahore, barva)`	Vyplní displej mezi souřadnicemi danou barvou.
`Tft.drawTraingle(x1, y1, x2, y2, x3, y3, barva)`	Zobrazí trojúhelník dané barvy s vrcholy `[x1,y1]`, `[x2,y2]`, `[x3,y3]`.
`Tft.drawChar(znak, x, y, v, barva)`	Zobrazí znak dané barvy na souřadnicích `[x,y]` o velikosti `v`.
`Tft.drawString(retezec, x, y, v, barva)`	Vypíše řetězec dané barvy na souřadnicích `[x,y]` velikosti `v`.
`Tft.setDisplayDirect(smer)`	Nastaví směr textu. Parametr `smer` může mít následující hodnoty: `LEFT2RIGHT`, `RIGHT2LEFT`, `DOWN2UP` a `UP2DOWN`.

Displej umí pracovat s 2^{16} (65 536) různými barvami. Informace o barvě jednoho pixelu tedy zabere 16 bitů. Červená a modrá barva mají každá pět bitů, zelená má bitů šest, lidské oko je na ni více citlivé, tudíž rozezná více jejích odstínů. Tento způsob míchání barev se nazývá 16-bit high color a více se o něm můžete dočíst na anglické Wikipedii[37]. Červená a modrá barva tedy můžou mít 32 různých sytostí, na zelenou jich připadá 64. Knihovna obsahuje několik předdefinovaných konstant barev. Jsou to:

Barva	Název konstanty	HEX kód
Červená	`RED`	0xf800
Zelená	`GREEN`	0x07e0
Modrá	`BLUE`	0x001f
Černá	`BLACK`	0x0000
Žlutá	`YELLOW`	0xffe0
Bílá	`WHITE`	0xffff
Azurová	`CYAN`	0x07ff
Purpurová	`BRIGHT_RED`	0xf810
Šedá	`GRAY1`	0x8410
Šedá	`GRAY2`	0x4208

Pokud by nám nestačila výchozí paleta, můžeme si vytvořit i barvy vlastní. Jeden ze způsobů vidíte na příkladu. S čísly se zde pracuje na úrovni jednotlivých bitů. Důležité ale je to, že funkce barva má tři parametry – `r`, `g` a `b` pro jednotlivé barvy a vrací číslo barvy. Parametry `r` a `b` mohou nabývat hodnot od 0 do 31. Parametr `g` 0 až 63. Následující program zobrazí na displeji tři kružnice tří základních barev.

[37] Wikipedia: 16bitové barevné schéma – `http://en.wikipedia.org/wiki/High_color#16-bit_high_color`

```
#include <stdint.h>
#include <TFTv2.h> //knihovna displeje
#include <SPI.h> //knihovna pro komunikaci s displejem

void setup(){
    Tft.TFTinit();

    Tft.drawCircle(120, 60, 20, barva(31,0,0));
    Tft.drawCircle(120, 160, 20, barva(0,63,0));
    Tft.drawCircle(120, 260, 20, barva(0,0,31));
}

void loop(){

}

uint16_t barva(int r, int g, int b){
    r = r % 32;
    g = g % 64;
    b = b % 32;

    r = r << 11;
    g = g << 5;

    return r | g | b;
}
```

Jednotlivě jsme již vyřešili dotykovou plochu i displej. Nyní pojďme dát vše dohromady. Vytvoříme si aplikaci pro jednoduché malování černou barvou na bílé pozadí. Aplikace bude obsahovat také tlačítko RESET.

```
#include <stdint.h>
#include <SeeedTouchScreen.h>
#include <TFTv2.h>
#include <SPI.h>

#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__) // mega
#define YP A2
#define XM A1
#define YM 54
#define XP 57

#elif defined(__AVR_ATmega32U4__) //leonardo
#define YP A2
#define XM A1
#define YM 18
#define XP 21

#else
#define YP A2
#define XM A1
#define YM 14
#define XP 17

#endif

TouchScreen ts = TouchScreen(XP, YP, XM, YM);

int min_x = 232;
int max_x = 1780;
int min_y = 166;
int max_y = 1826;
```

```
32
33 int x, y;
34
35 void setup(){
36     Tft.TFTinit();
37
38     Tft.fillScreen(10,230,10,230, WHITE);
39     Tft.fillScreen(10,230,240,310, GRAY2);
40
41     Tft.drawString("RESET", 45, 255, 5, BLACK);
42 }
43 void loop(){
44     Point p = ts.getPoint();
45
46     x = map(p.x, min_x, max_x, 0, 239);
47     y = map(p.y, min_y, max_y, 0, 319);
48
49     if(p.z > 10){ // pokud tlak presahne urcitou hodnotu
50         if(x >= 10 && x <= 220 && y >= 250 && y <= 310){ //kdyz zmackneme na
               RESET
51             Tft.fillScreen(10,230,10,230, WHITE);
52         }
53         else if(x >= 10 && x <= 220 && y >= 10 && y <= 230 ){
54             Tft.fillRectangle(x,y,2,2,BLACK);
55         }
56     }
57
58     delay(1);
59 }
```

Obrázek 54.8: Výsledný program

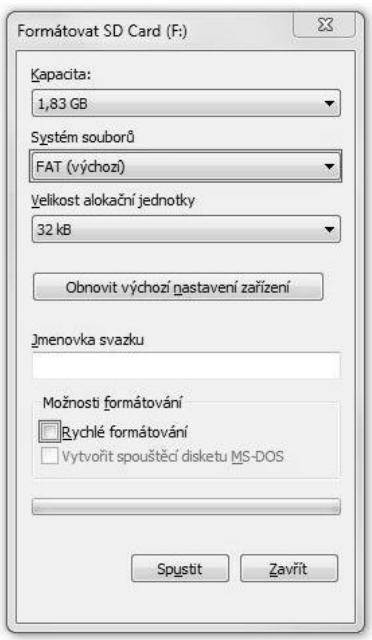
Část XIV

Projekt: 2048

Nyní se více než na teorii zaměříme na použití Arduina v praxi. Využijeme TFT shield popsaný v minulé části, předvedeme si, jak pracovat s SD kartou a nakonec si naprogramujeme hru 2048.

Kapitola 55
SD karta

Arduino IDE obsahuje již od základu knihovnu pro obsluhu SD karet. Ta nám umožňuje pracovat s kartami, které mají formáty souborového systému FAT16 a FAT32. Název souboru nesmí mít více než 8 znaků a přípona musí mít znaky tři. Pokud má karta, se kterou chceme pracovat, jiný než požadovaný typ, musíme ji před použitím zformátovat. To se ve Windows 7 udělá velmi jednoduše. Kartu vložíme do čtečky a otevřeme *Počítač*. Pravým tlačítkem myši otevřeme nabídku ikony karty, kterou chceme formátovat a vybereme možnost *Formátovat*. Po otevření dialogového okna pro formátování vybereme z nabídky *Systém souborů* možnost FAT, nebo FAT32. Také odškrtneme možnost *Rychlé formátování* (není potřeba vždy, ale máme jistotu, že se karta opravdu správně naformátuje).



Obrázek 55.1: Formátovací dialog

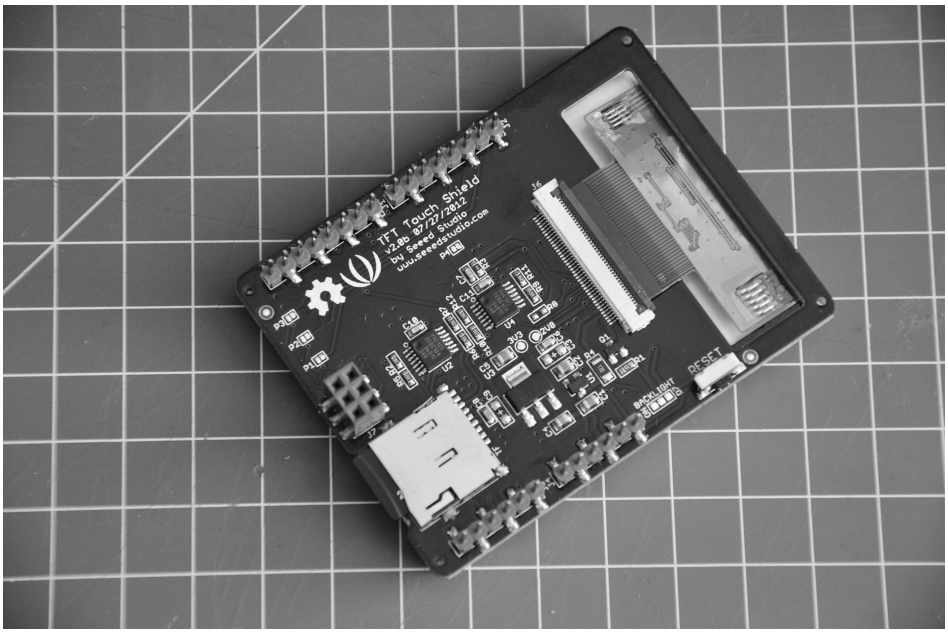
55.1 Příprava Arduina

Aby mohlo Arduino s SD vůbec komunikovat, musíme mít k němu připojenou vhodnou čtečku. Taková čtečka nemusí být vůbec drahá ani náročná na výrobu. Udělat se dá třeba jen i s pár piny (jak popisuje tento návod[38]). Také existuje celá řada shieldů, které SD čtečku obsahují. Jsou jimi například SD Card shield V4.0, Arduino Ethernet shield SD a další. Nyní si předvedeme, jak použít micro SD čtečku, kterou obsahuje TFT Touch shield popsaný v předchozí části. Čtečku nalezneme na spodní straně shieldu poblíž jednoho z rohů.

55.2 Funkce

Arduino komunikuje se čtečkou přes SPI rozhraní (piny MISO, MOSI, SCK, SS). Tyto piny jsou u každého Arduina jinde (u verze Mega je nalezneme na 50, 51, 52, 53, u Uno jim pak odpovídají 12, 11, 13, 10 atd.). Bližší informace nalezneme v dokumentaci jednotlivých desek. I když pin SS nepoužíváme, musí být nastavený jako `OUTPUT` (Uno 10, Mega 53...). Pro práci s SD potřebujeme knihovnu `SD.h`, kterou do kódu vložíme známým příkazem `#include <SD.h>`.

[38] DIY SD čtečka – `http://www.instructables.com/id/Cheap-DIY-SD-card-breadboard-socket/`



Obrázek 55.2: TFT shield – spodní strana

Zasuneme kartu, připojíme shield a můžeme programovat. Funkce obsažené v knihovně se dají rozdělit do dvou skupin. První skupina funkcí slouží k práci s umístěním souborů a složek. Druhá skupina umí měnit samotný obsah souborů. Přehled funkcí nalezneme v dokumentaci knihovny[39]. My si představíme ty nejužitečnější.

Funkce pro práci s umístěním složek a souborů.

Název	Funkce
`SD.begin(pin)`	Inicializuje SD kartu. Při úspěchu vrátí `true`, jinak `false`. Parametr `pin` slouží k nastavení linky pro výběr čipu. Nejčastěji má hodnotu 4.
`SD.exists(jmeno)`	Pokud zadaná složka nebo soubor existuje, vrátí hodnotu `true`, jinak `false`. Parametr `jmeno` může obsahovat i cestu k souboru (např. `SLOZKA1/SLOZKA2/SOUBOR.TXT`).
`SD.mkdir(jmeno)`	Vytvoří zadanou složku. Pokud má proměnná `jmeno` formát například `sl1/sl2/a`, vytvoří se i nadřazené složky (pokud neexistují).
`SD.open(jmeno, mod)`	Otevře vybraný soubor (parametr `jmeno` se řídí stejnými pravidly jako u `SD.exists()`). Parametr `mod` je nepovinný a je defaultně nastavený na `FILE_READ`. V tomto stavu je soubor otevřený pouze ke čtení a kurzor pozice je umístěn na začátku. Pokud nastavíme mod na `FILE_WRITE`, otevře se soubor i pro zápis s kurzorem pozice na jeho konci. Pokud soubor neexistuje, funkce ho vytvoří. Všechny nadřazené složky však musí existovat. Tato funkce je důležitá tím, že vrací objekt obsahující informace o otevřeném souboru, se kterým poté pracuje druhá skupina funkcí.
`SD.remove(jmeno)`	Odstraní vybraný soubor (ne složku!).
`SD.rmdir(jmeno)`	Odstraní vybranou složku – složka však musí být prázdná.

Funkce `SD.open()` vrací objekt souboru, se kterým pracujeme. Ten je datového typu `File`. Následující funkce tedy použijeme pro práci s objektem `soubor` získaným takto:

```
1  File soubor = SD.open("abc.txt", FILE_WRITE);
```

[39] Dokumentace SD knihovny – `http://arduino.cc/en/Reference/SD`

Funkce pro práci s obsahem souboru.

Název	Funkce
`soubor.available()`	Funguje stejně jako u sériové komunikace – vrátí počet nepřečtených bytů v souboru.
`soubor.close()`	Zavře aktivní soubor.
`soubor.read()`	Přečte jeden byte ze souboru.
`soubor.write(data)`	Zapíše jeden byte do souboru.
`soubor.print(data)`	Zapíše data do souboru jako text (ASCII kódování). Čísla rozdělí na jednotlivé číslice a ty poté zapíše jednotlivě.
`soubor.println(data)`	Funguje stejně jako `.print()`, jen na konec informace přidá zalomení řádku a návrat posuvníku.

Na ukázku si dovolíme použít příklady z dokumentace.

55.3 Příklad 1: Zápis hodnot

Tento příklad měří hodnoty na A0, A1 a A3 a zapisuje je do složky na SD kartě.

```
1  #include <SD.h> //vlozeni knihovny
2
3  const int chipSelect = 4; //vyber pinu pro SD.begin()
4
5  void setup(){
6      Serial.begin(9600);
7      while (!Serial) {
8          ; //pocka na zahajeni komunikace – pouze pro desky s AT32u4
9      }
10
11     Serial.print("Initializing SD card...");
12     pinMode(10, OUTPUT); //SS pin desky, se kterou pracujeme
13
14     //je karta pritomna a v poradku?
15     if (!SD.begin(chipSelect)) {
16         Serial.println("Card failed, or not present");
17         return;
18     }
19     Serial.println("card initialized.");
20 }
21
22 void loop(){
23     //vytvori retezec pro shromazdovani dat
24     String dataString = "";
25
26     //precti 3 vstupy a vysledek zapis do retezce
27     for (int analogPin = 0; analogPin < 3; analogPin++) {
28         int sensor = analogRead(analogPin);
29         dataString += String(sensor);
30         if (analogPin < 2) {
31             dataString += ",";
32         }
33     }
34
35     //otevre soubor
36     File dataFile = SD.open("datalog.txt", FILE_WRITE);
37
38     // pokud se otevreni povedlo, data se zapisi do slozky
```

```
39      if (dataFile) {
40          dataFile.println(dataString);
41          dataFile.close();
42          Serial.println(dataString);
43      }
44      else {
45          Serial.println("error opening datalog.txt");
46      }
47  }
```

55.4 Příklad 2: Výpis dat ze souboru

V druhém příkladu přečteme data uložená na SD v předchozí části a vypíšeme je po sériové lince.

```
1  #include <SD.h>
2
3  const int chipSelect = 4;
4
5  void setup(){
6      Serial.begin(9600);
7      while (!Serial) {
8          ;
9      }
10
11     Serial.print("Initializing SD card...");
12     pinMode(10, OUTPUT);
13
14     if (!SD.begin(chipSelect)) {
15         Serial.println("Card failed, or not present");
16         return;
17     }
18     Serial.println("card initialized.");
19
20     File dataFile = SD.open("datalog.txt");
21
22     if (dataFile) {
23         while (dataFile.available()) {
24             Serial.write(dataFile.read());
25         }
26         dataFile.close();
27     }
28     else {
29         Serial.println("error opening datalog.txt");
30     }
31 }
32
33 void loop(){}
```

Tímto jsme si v rychlosti předvedli práci s SD kartou a můžeme se pustit do programování hry.

Kapitola 56
Hra 2048

V březnu roku 2014 zveřejnil na svém webu devatenáctiletý italský vývojář hru 2048. To rázem strhlo lavinu jejích klonů. A nešlo jen o verze pro prohlížeče, ale i všemožné platformy a programovací jazyky. Základní princip je jednoduchý. Na poli 4×4 máme dlaždice s hodnotami mocnin 2. Pohybem do čtyř stran můžeme dlaždice se stejnou hodnotou sečíst. Výsledkem sečtení je dlaždice o hodnotě součtu dvou předchozích (a nebo mocnina dvou s o jedno vyšším exponentem). V našem případě využijeme TFT dotykový displej pro zobrazení a ovládání hry a čtečku karet pro uložení postupu.

56.1 Hodnoty

Z praktických důvodů nebudeme pracovat s mocninami dvou, ale pouze s exponenty (mocniny by se na displej špatně vešly). Základní dlaždice bude mít tedy hodnotu 1. Při součtu dvou stejných dlaždic vznikne jedna dlaždice s o jedna větší hodnotou. Tímto způsobem lze na poli 4×4 dosáhnout maximální hodnoty 17. Princip zjištění maximální hodnoty je vidět na obrázku.

2	9	10	17
3	8	11	16
4	7	12	15
5	6	13	14

Obrázek 56.1: Maximální hodnoty

V programu budeme mít uloženy hodnoty jednotlivých dlaždic v poli `plocha[y][x]`. Pokud bude mít prvek hodnotu nula, bude se chovat, jako by tam žádná dlaždice nebyla. Další hodnoty, kterých může nabýt, jsou 1 až 17.

56.2 Jdeme na to

Nyní si celý program rozebereme část po části.

Na začátek si musíme vložit všechny potřebné knihovny a nadefinovat používané proměnné.

```
1  #include <stdint.h>
2  #include <SeeedTouchScreen.h>
3  #include <TFTv2.h>
4  #include <SPI.h>
5  #include <SD.h>
6
7  #if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__) // mega
8  #define YP A2
9  #define XM A1
10 #define YM 54
11 #define XP 57
12
13 #elif defined(__AVR_ATmega32U4__) //leonardo
14 #define YP A2
15 #define XM A1
16 #define YM 18
17 #define XP 21
18
19 #else
20 #define YP A2
21 #define XM A1
22 #define YM 14
23 #define XP 17
24
25 #endif
26
```

```
27 //konfigurace
28
29 TouchScreen ts = TouchScreen(XP, YP, XM, YM);
30 byte SDpin = 4;
31
32 int min_x = 232;
33 int max_x = 1780;
34 int min_y = 166;
35 int max_y = 1826;
36
37 Point p;
38
39 //promenne pouzivane ve hre
40
41 //0 - uvodni obrazovka, 1 - normalni hra, 2 - prohra, 3 - vyhra
42 byte stat = 0;
43
44 //pole pro ulozeni hodnot jednotlivych dlazdic plochy
45 byte plocha[4][4];
46
47 uint16_t barvy[18]; //pole s hodnotami barev
48 boolean rewrite_all = true; //prepis cele obrazovky
49 boolean rewrite_game; //prepis herni plochy
50 boolean moved; //mlelo se s dlazdicemi?
51 boolean gEnd; //konec hry?
52 boolean gWin; //vyhra?
53
54 //pomocne promenne
55 byte added = 0; //kolikrat doslo k secteni dlazdic
56 byte max_added;
57 byte len;
58
59 File soubor;
```

V dalším kroku nastavíme vše podstatné. Inicializujeme SD kartu a displej, nastavíme SS pin na `OUTPUT` (10 pro Uno, 53 pro Mega...). Dále si připravíme do pole `barvy[]` paletu barev. Barvy si můžeme zvolit libovolně. V našem případě jsou rovnoměrně rozložené po celé škále.

```
60 void setup(){
61     //inicializace SD karty a displeje
62     Tft.TFTinit();
63     SD.begin(SDpin);
64     pinMode(53, OUTPUT);
65
66     //prirazeni hodnot barvam
67     barvy[0] = GRAY1;
68     for(int b = 1; b < 18; b++){
69         barvy[b] = b * (65535 / 18);
70     }
71
72     randomSeed(analogRead(0)); //seed pro nahodny generator
73 }
```

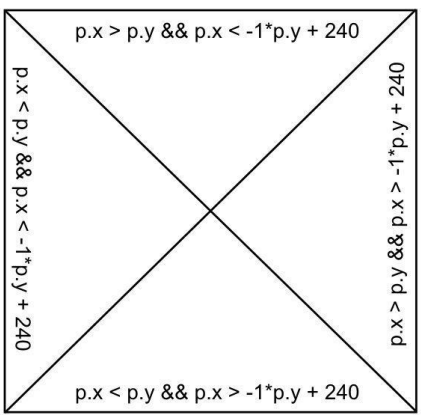
Vše máme nastaveno. Nyní už se budeme zabývat blokem `loop()`. To, v jaké části zrovna hra je, je uloženo v proměnné stat (0 – úvodní obrazovka, 1 – normální hra, 2 – prohra, 3 – výhra). Pokud je proměnná `rewrite_all true`, dojde k přepsání celé obrazovky. Na úvodní obrazovce nalezneme dvě tlačítka: LOAD a NEW. V této části tedy také ověříme, jestli uživatel na některé z nich zmáčkl. Pokud dojde ke zmáčknutí tlačítka LOAD, nahraje se uložená hra z SD. V našem případě jsou na kartě informace o dlaždicích uloženy jako jeden byte pro jednu dlaždici. Tyto byty jsou uloženy bez mezer na jednom řádku. Při zmáčknutí tlačítka NEW se všechny dlaždice nastaví na 0 a poté se pomocí námi definované funkce `addTile()` (bude přidána na konci) přidají dvě dlaždice o hodnotě 1 nebo 2. Funkce `checkP()` aktualizuje bod `p`.

```
74 void loop(){
75     if(rewrite_all){
76         Tft.fillRectangle(0,0,239,319, barvy[0]);
77     }
78 
79     //uvodni obrazovka
80     if(stat == 0){
81         if(rewrite_all){
82             Tft.fillRectangle(10,10,220,145, barvy[2]);
83             Tft.fillRectangle(10,165,220,145, barvy[2]);
84             Tft.drawString("LOAD", 35, 55, 7, BLACK);
85             Tft.drawString("NEW", 55, 210, 7, BLACK);
86             rewrite_all = false;
87         }
88 
89         checkP();
90 
91         if(p.z > 10){
92             //tlacitko LOAD
93             if(p.y < 160){
94                 rewrite_all = true;
95                 rewrite_game = true;
96                 if(!SD.exists("G2048.TXT")){
97                     stat = 0;
98                     Tft.fillRectangle(20,95,200,100, barvy[17]);
99                     Tft.drawString("ERROR", 28, 123, 6, BLACK);
100 
101                    delay(500);
102                }
103                else{
104                    stat = 1;
105                    soubor = SD.open("G2048.TXT");
106                    int i = 0;
107                    byte p;
108 
109                    for(int i=0; i < 4; i++){ //vycistime plochu
110                        for(int j = 0; j < 4; j++){
111                            plocha[i][j] = 0;
112                        }
113                    }
114 
115                    while(soubor.available()){
116                        p = soubor.read();
117                        plocha[(i - i%4) /4][i % 4] = p;
118                        i++;
119                    }
120                    soubor.close();
121                }
122            }
123 
124            //tlacitko NEW
125            else if(p.y >= 160){
126                rewrite_all = true;
127                rewrite_game = true;
128                moved = false;
129                added = 0;
130                gEnd = false;
131                gWin = false;
132                stat = 1; //pokracovat budeme hrou
133 
```

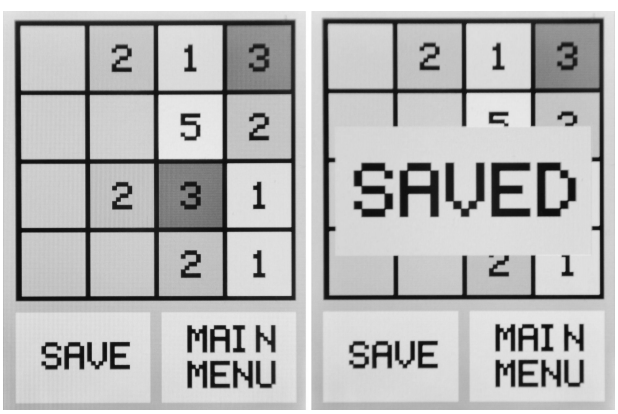
```
134                 //cela plocha se nastavi na 0
135                 for(int i = 0; i < 4; i++){
136                     for(int j = 0; j < 4; j++){
137                         plocha[i][j] = 0;
138                     }
139                 }
140 
141                 addTile();
142                 addTile();
143             }
144         }
145     }
```

Pokračovat budeme případem normální hry. Pokud je proměnná `rewrite_game true`, dojde k přepsání herní plochy. Dále program detekuje, jestli došlo k dotyku v oblasti herní plochy a tlačítek. Ovládání směru je nastaveno tak, jak vidíte na obrázku. Detekce dotyku v oblasti směrových šipek vychází z funkce $y = x$, jejímž grafem je přímka, která svírá úhel $45°$ s oběma osami.

Také zkontrolujeme, jestli jsou možné další tahy. Pokud ano, musí být buďto alespoň jedna dlaždice 0, nebo musí být alespoň jedna dvojice dlaždic se stejnou hodnotou. Výhru poznáme tak, že se na poli vyskytne dlaždice o hodnotě 17 (nebo jiná nižší hodnota, kterou si pro konec hry zvolíme).



Obrázek 56.2: Segmenty displeje



Obrázek 56.3: Herní plocha

Pokud se během hry pohnulo s bloky (tah byl úspěšný), přidáme pomocí funkce `addTile()` novou dlaždici. Také nadefinujeme tlačítka pro uložení a návrat do hlavního menu.

```
146     //normalni hra
147     else if(stat == 1){
148         if(rewrite_all){
149             rewrite_all = false;
150 
151             Tft.fillRectangle(10,240,105,70,barvy[10]);
152             Tft.fillRectangle(125,240,105,70,barvy[10]);
153 
154             Tft.drawString("SAVE", 27, 263, 3, BLACK);
155             Tft.drawString("MAIN", 142, 250, 3, BLACK);
156             Tft.drawString("MENU", 142, 278, 3, BLACK);
157         }
158         if(rewrite_game){
159             Tft.fillRectangle(10,10,220,220,BLACK);
160 
161             //zobrazeni jednotlivych dlazdic
162             for(int x = 0; x < 4; x++){
163                 for(int y = 0; y < 4; y++){
164                     Tft.fillRectangle(15+(x*54),15+(y*54), 49, 49,
                            barvy[plocha[y][x]]);
165                     if(plocha[y][x] != 0){
166                         if(plocha[y][x] >= 10){
167                             len = 0;
168                         }
169                         else{
170                             len = 10;
171                         }
172 
173                         Tft.drawNumber(plocha[y][x], 19+(x*54)+len, 29+(y*54),
                                3, BLACK);
174                     }
175                 }
176             }
177             rewrite_game = false;
178         }
179 
180         checkP();
181 
182         if(p.z > 10){
183 
184             //detekce dotyku v oblasti herni plochy
185             if(p.x > 10 && p.x < 220 && p.y > 10 && p.y < 230){
186                 //smer nahoru
187                 if(p.x > p.y && p.x < -1*p.y + 240){
188                     goUp();
189                 }
190 
191                 //smer dolu
192                 else if(p.x < p.y && p.x > -1*p.y + 240){
193                     goDown();
194                 }
195 
196                 //smer doprava
197                 else if(p.x > p.y && p.x > -1*p.y + 240){
198                     goRight();
199                 }
200 
201                 //smer doleva
202                 else if(p.x < p.y && p.x < -1*p.y + 240){
203                     goLeft();
```

```
204                 }
205                 do{
206                     checkP();
207                     delay(10);
208                 }while(p.z > 10);
209
210                 //kontrola konce hry
211                 gEnd = true; //nastavime na true a pokud nebude pravda,
                        v nasledujicim cyklu to zmenime
212                 for(int x = 0; x <= 3; x++){
213                     for(int y = 0; y <= 3; y++){
214                         if(plocha[y][x] == 0){
215                             gEnd = false;
216                         }
217                         else if(x == 3 && y == 3){
218                             //nic
219                             //u dolni prave dlazdice nic nekontrolujeme
220                         }
221                         else if(x == 3){
222                             if(plocha[y][3] == plocha[y+1][3] ){
223                                 gEnd = false;
224                             }
225                         }
226                         else if(y == 3){
227                             if(plocha[3][x] == plocha[3][x+1]){
228                                 gEnd = false;
229                             }
230                         }
231                         else{
232                             //zbyla cast pole
233                             if(plocha[y][x] == plocha[y][x+1] || plocha[y][x]
                                    == plocha[y+1][x]){
234                                 gEnd = false;
235                             }
236                         }
237                     }
238                 }
239
240                 //kontrola vyhry
241                 gWin = false;
242                 for(int x = 0; x <= 3; x++){
243                     for(int y = 0; y <= 3; y++){
244                         if(plocha[y][x] == 17){
245                             gWin = true;
246                         }
247                     }
248                 }
249
250                 //pokud se hnulo s bloky
251                 if(moved){
252                     moved = false;
253                     rewrite_game = true;
254
255                     if(gEnd == false){
256                         addTile();
257                     }
258                 }
259
260                 //rozhodnuti dalsiho postupu
261                 if(gEnd == true){
```

```
262                     gEnd = false;
263                     stat = 2;
264 
265                     rewrite_all = true;
266                     rewrite_game = true;
267                 }
268                 else if(gWin == true){
269                     gWin = false;
270 
271                     stat = 3;
272 
273                     rewrite_all = true;
274                     rewrite_game = true;
275                 }
276 
277                 rewrite_game = true;
278             }
279 
280             //dolni cast herni plochy s tlacitky
281             else if(p.y > 240 && p.y < 310){
282 
283                 //tlacitko SAVE
284                 if(p.x > 10 && p.x < 115){
285                     Tft.fillRectangle(20,95,200,100, barvy[17]);
286 
287                     //ukladani
288                     if(SD.exists("G2048.TXT")){
289                         SD.remove("G2048.TXT"); //vycistime pripadny stary soubor
290                     }
291                     soubor = SD.open("G2048.TXT", FILE_WRITE);
292 
293                     //pokud se otevreni povedlo, zapiseme hodnoty
294                     if(soubor){
295                         for(int y = 0; y <=3; y++){
296                             for(int x = 0; x <= 3; x++){
297                                 soubor.write(plocha[y][x]);
298                             }
299                         }
300                         soubor.close();
301                         Tft.drawString("SAVED", 28, 123, 6, BLACK);
302                     }
303                     else{
304                         Tft.drawString("ERROR", 28, 123, 6, BLACK);
305                     }
306 
307                     delay(1000);
308 
309                     rewrite_all = true;
310                     rewrite_game = true;
311                 }
312 
313                 //tlacitko MAIN MENU
314                 else if(p.x > 125 && p.x < 210){
315                     stat = 0;
316                     rewrite_all = true;
317                 }
318             }
319         }
320     }
```

Obrazovky pro výhru a prohru jsou prakticky stejné, liší se pouze v nápisu. V dolní části mají tlačítko MAIN MENU.

```
321     //prohra
322     else if(stat == 2){
323         if(rewrite_all){
324             Tft.fillRectangle(10,10,220,300,barvy[2]);
325             Tft.fillRectangle(20,200,200,100, barvy[10]);
326 
327             Tft.drawString("GAME", 32, 35, 7, BLACK);
328             Tft.drawString("OVER", 32, 115, 7, BLACK);
329             Tft.drawString("MAIN MENU", 37, 238, 3, BLACK);
330 
331             rewrite_all = false;
332         }
333 
334         checkP();
335 
336         //tlacitko MAIN MENU
337         if(p.z > 10){
338             if(p.x > 20 && p.x < 220 && p.y > 200 && p.y < 320){
339                 rewrite_all = true;
340                 stat = 0;
341             }
342         }
343     }
344 
345     //vyhra
346     if(stat == 3){
347         if(rewrite_all){
348             Tft.fillRectangle(10,10,220,300,barvy[2]);
349             Tft.fillRectangle(20,200,200,100, barvy[10]);
350 
351             Tft.drawString("YOU", 50, 35, 7, BLACK);
352             Tft.drawString("WON", 50, 115, 7, BLACK);
353             Tft.drawString("MAIN MENU", 37, 238, 3, BLACK);
354 
355             rewrite_all = false;
356         }
357 
358         checkP();
359 
360         //tlacitko MAIN MENU
361         if(p.z > 10){
362             if(p.x > 20 && p.x < 220 && p.y > 200 && p.y < 320){
363                 rewrite_all = true;
364                 stat = 0;
365             }
366         }
367     }
368 }
```

Nyní ještě musíme nadefinovat použité funkce. Funkce `checkP()` aktualizuje pozici dotyku (bod `p`). Funkce `addTile()` přidá dlaždici o hodnotě 1 nebo 2 na volné místo.

```
369 //nastavi aktualni pozici dotyku uzivatele
370 void checkP(){
371     p = ts.getPoint();
372     p.x = map(p.x, min_x, max_x, 0, 240);
373     p.y = map(p.y, min_y, max_y, 0, 320);
374 }
375 
```

```
376 //prida novou dlazdici
377 void addTile(){
378     byte r1, r2;
379 
380     do{
381         r1 = random(4);
382         r2 = random(4);
383     }while(plocha[r1][r2] != 0);
384 
385     plocha[r1][r2] = random(1,3);
386 }
387 
388 void goUp(){
389     for(int x = 0; x <= 3; x++){
390         if(plocha[0][x] == plocha[1][x] && plocha[2][x] == plocha[3][x]){
391             max_added = 2;
392         }
393         else{
394             max_added = 1;
395         }
396 
397         for(int y = 0; y <= 2; y++){
398             for(int y_p = y; y_p >= 0; y_p--){
399                 if(plocha[y_p][x] == plocha[y_p+1][x] && plocha[y_p][x] != 0
                        && added < max_added){
400                     plocha[y_p][x]++;
401                     plocha[y_p+1][x] = 0;
402                     added++;
403                     moved = true;
404                 }
405                 if(plocha[y_p][x] == 0 && plocha[y_p+1][x] != 0){
406                     plocha[y_p][x] = plocha[y_p+1][x];
407                     plocha[y_p+1][x] = 0;
408                     moved = true;
409                 }
410             }
411         }
412         added = 0;
413     }
414 }
```

V poslední části vytvoříme funkce pro pohyb dlaždic do stran. To je logicky asi nejsložitější část programu. My si ji vysvětlíme na pohybu doprava – ostatní pohyby jsou pouze modifikací.

Při zmáčknutí tlačítka doprava dojde ke kontrole dlaždic v řádku, pokud `plocha[y][0]==plocha[y][1]` a zároveň `plocha[y][2]==plocha[y][3]` (například pro {2,2,3,3} nebo {3,3,3,3}), může dojít k sečtení dvakrát. V ostatních případech pouze jednou. Kdybychom tuto část vynechali, docházelo by ke špatnému sčítání – {1,1,2,0} by při pohybu doprava skončilo takto: {0,0,0,3}. Námi požadovaný stav je však {0,0,2,2}. Poté pokračujeme kontrolou sousedících dvojic a přesunem hodnot jiných než 0 na prázdné dlaždice (s hodnotou 0).

```
415 void goUp(){
416     for(int x = 0; x <= 3; x++){
417         if(plocha[0][x] == plocha[1][x] && plocha[2][x] == plocha[3][x]){
418             max_added = 2;
419         }
420         else{
421             max_added = 1;
422         }
423 
424         for(int y = 0; y <= 2; y++){
425             for(int y_p = y; y_p >= 0; y_p--){
```

```
426                 if(plocha[y_p][x] == plocha[y_p+1][x] && plocha[y_p][x] != 0
                        && added < max_added){
427                     plocha[y_p][x]++;
428                     plocha[y_p+1][x] = 0;
429                     added++;
430                     moved = true;
431                 }
432                 if(plocha[y_p][x] == 0 && plocha[y_p+1][x] != 0){
433                     plocha[y_p][x] = plocha[y_p+1][x];
434                     plocha[y_p+1][x] = 0;
435                     moved = true;
436                 }
437             }
438         }
439         added = 0;
440     }
441 }
442
443 void goDown(){
444     for(int x = 0; x <= 3; x++){
445         if(plocha[0][x] == plocha[1][x] && plocha[2][x] == plocha[3][x]){
446             max_added = 2;
447         }
448         else{
449             max_added = 1;
450         }
451
452         for(int y = 3; y >= 1; y--){
453             for(int y_p = y; y_p <= 3; y_p++){
454                 if(plocha[y_p-1][x] == plocha[y_p][x] && plocha[y_p][x] != 0
                        && added < max_added){
455                     plocha[y_p-1][x]++;
456                     plocha[y_p][x] = 0;
457                     added++;
458                     moved = true;
459                 }
460                 if(plocha[y_p][x] == 0 && plocha[y_p-1][x] != 0){
461                     plocha[y_p][x] = plocha[y_p-1][x];
462                     plocha[y_p-1][x] = 0;
463                     moved = true;
464                 }
465             }
466         }
467         added = 0;
468     }
469 }
470 void goRight(){
471     for(int y = 0; y <= 3; y++){
472         if(plocha[y][0] == plocha[y][1] && plocha[y][2] == plocha[y][3]){
473             max_added = 2;
474         }
475         else{
476             max_added = 1;
477         }
478         for(int x = 3; x >= 1; x--){
479             for(int x_p = x; x_p <= 3; x_p++){
480                 if(plocha[y][x_p-1] == plocha[y][x_p] && plocha[y][x_p] != 0
                        && added < max_added){
481                     plocha[y][x_p-1]++;
482                     plocha[y][x_p] = 0;
```

```
483                     added++;
484                     moved = true;
485                 }
486                 if(plocha[y][x_p] == 0 && plocha[y][x_p-1] != 0){
487                     plocha[y][x_p] = plocha[y][x_p-1];
488                     plocha[y][x_p-1] = 0;
489                     moved = true;
490                 }
491             }
492         }
493         added = 0;
494     }
495 }
496 
497 void goLeft(){
498     for(int y = 0; y <= 3; y++){
499         if(plocha[y][0] == plocha[y][1] && plocha[y][2] == plocha[y][3]){
500             max_added = 2;
501         }
502         else{
503             max_added = 1;
504         }
505         for(int x = 0; x <= 2; x++){
506             for(int x_p = x; x_p >= 0; x_p--){
507                 if(plocha[y][x_p] == plocha[y][x_p+1] && plocha[y][x_p] != 0
                        && added < max_added){
508                     plocha[y][x_p]++;
509                     plocha[y][x_p+1] = 0;
510                     added++;
511                     moved = true;
512                 }
513                 if(plocha[y][x_p] == 0 && plocha[y][x_p+1]){
514                     plocha[y][x_p] = plocha[y][x_p+1];
515                     plocha[y][x_p+1] = 0;
516                     moved = true;
517                 }
518             }
519         }
520         added = 0;
521     }
522 }
```

Celý program je možné stáhnout ze stránky GitHub[40].

[40] GitHub projekt – `https://github.com/VodaZ/g2048`

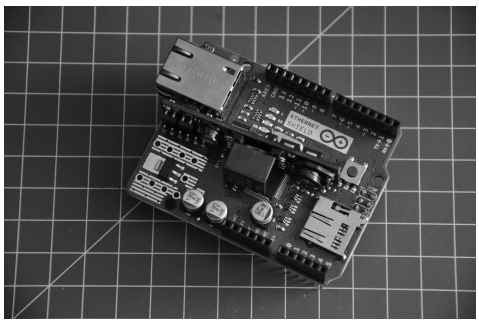
Část XV

Arduino a Ethernet shield

Nyní si ukážeme, jak pracovat s Ethernet shieldem, což je zajímavé rozšíření pro Arduino, které nám přináší nové možnosti interakce Arduina se sítí i internetem.

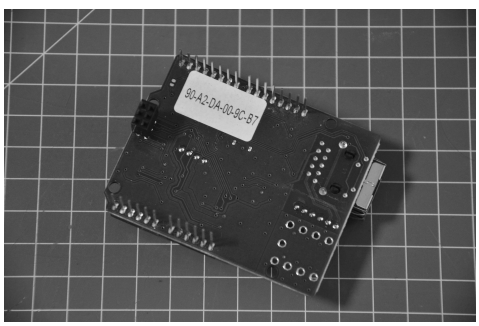
Kapitola 57
Ethernet shield

Ethernet shield si (stejně jako celé Arduino) prošel poměrně dlouhým vývojem, proto se setkáme hned s několika jeho verzemi. My budeme pracovat s verzí Arduino Ethernet Rev3 with PoE.



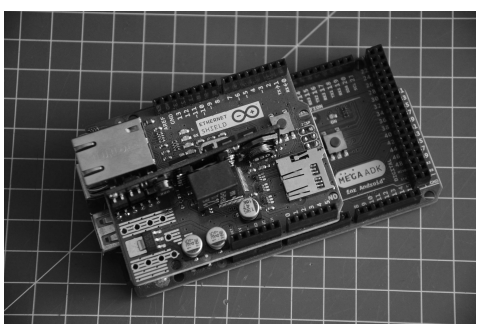
Obrázek 57.1: Ethernet shield

Dominantním prvkem desky je RJ45 konektor pro připojení Ethernet kabelu. Mimo něj ale na shieldu nalezneme i slot pro SD kartu, jejíž používání jsme si popsali na straně 146. Ovládání čipu (W5100) i SD karty probíhá přes SPI rozhraní. Rychlost síťové komunikace 10/100 Mb/s není v dnešní době zrovna strhující, ale pro naše účely je zcela dostačující. Než shield připojíme k Arduinu, otočíme jej.



Obrázek 57.2: MAC adresa

Na jeho spodní straně nalezneme nálepku s MAC adresou (unikátní identifikační číslo síťového zařízení). Tu si někam poznamenáme pro pozdější použití. Když máme MAC adresu zapsanou, můžeme shield připojit na Arduino (v tomto případě Arduino Mega).



Obrázek 57.3: Arduino Mega s připojeným Ethernet shieldem

57.1 Funkce

Pro programování Ethernet shieldu se používá hned několik tříd. Třída `Ethernet` slouží k základnímu nastavení.

Název	Zápis	Funkce

`Ethernet.begin(mac)`	`Ethernet.begin(mac)`
	`Ethernet.begin(mac, ip)`
	`Ethernet.begin(mac, ip, dns)`
	`Ethernet.begin(mac, ip, dns, gateway)`
	`Ethernet.begin(mac, ip, dns, gateway, subnet)`
		Slouží k zahájení komunikace shieldu s okolím (většinou router či switch). Vlevo vidíte použití různých parametrů. Nejčastěji se používají pouze `mac` a `ip`. Když parametr `ip` nepoužijeme, je IP adresa přidělena automaticky DHCP serverem.

`Ethernet.localIP()`	`Ethernet.localIP()`	Vrátí IP adresu shieldu. Tato funkce se nepoužívá, pokud IP adresu přiřazujeme manuálně, ale když ji necháme přes DHCP přidělit automaticky.
`Ethernet.maintain()`	`Ethernet.maintain()`	Pokud má shield přiřazenou adresu automaticky, může touto funkcí požádat o její obnovení. Přidělená adresa může být v závislosti na nastavení routeru stejná i nová.

Jakousi pomocnou třídou je třída `IPAddress`. Ta slouží k uchování IP adresy.

Název	Zápis	Funkce
`IPAddress()`	`IPAddress jmeno(a,b,c,d)`	Tato funkce uloží IP adresu. Zápis adresy je ve tvaru `a.b.c.d` (např. `10.0.0.1`) se jménem `mojeIP` se provede vytvořením následujícího objektu `IPAddress mojeIP(a,b,c,d)`.

Třída `Server` slouží k odesílání a přijímání dat mezi shieldem a připojenými klienty (programy na jiných zařízeních, které se připojují k serveru).

Název	Zápis	Funkce
`Server()`	`EthernetServer mujSvr = EthernetServer(port)`	Vytvoří server, který naslouchá příchozím připojením na vybraném portu (80 pro HTTP, 23 pro telnet…).
`mujSvr.begin()`	–	Spustí vybraný server.
`mujSvr.available()`	`EthernetClient client = server.available()`	Vrátí objekt `Client`, který je připojen k našemu serveru a odesílá data ke čtení.
`mujSvr.write()`	`mujSvr.write(val)` `mujSvr.write(buf, len)`	Pošle data všem klientům připojeným k serveru. Data mohou být typu `char`, `byte` nebo pole těchto typů (`buf` je poté délka tohoto pole).
`mujSvr.print()`	`mujSvr.print(data)` `mujSvr.print(data,BASE)`	Stejné jako `.write()`, jen `data` převádí na text. Parametr `BASE` může nabývat hodnot: `BIN` (dvojková soustava), `OCT` (osmičková soustava), `DEC` (dekadická soustava), `HEX` (šestnáctková soustava).
`mujSvr.println()`	`mujSvr.println(data)` `mujSvr.println(data,BASE)`	Stejné jako `.print()`, jen na konec přidá zalomení řádku.

Ke zpracování dat ze serveru slouží třída `Client`. Tato třída vytvoří ze shieldu klienta, který se může připojit k jiným serverům.

Název	Zápis	Funkce
`EthernetClient ja`	`EthernetClient ja`	Vytvoří klienta s názvem `ja`.
`ja`	`while(!ja)delay(1)`	Počká, dokud není klient připojen.
`ja.connected()`	–	Vrátí `true`, pokud jsou od klienta dostupná data ke zpracování (v době zpracování už nemusí být přítomen, ale musí být od něj přijata data).
`ja.connect()`	`ja.connect(ip,port)` `ja.connect(URL,port)`	Připojí se k vybrané `ip`, nebo URL přes zadaný `port`.
`ja.write()`	`ja.write(val)` `ja.write(buf,len)`	Pošle data serveru, ke kterému je shield připojen.
`ja.print()`	`ja.print(data)` `ja.print(data,BASE)`	Pošle data serveru jako text.
`ja.println()`	`ja.println(data)` `ja.println(data,BASE)`	Pošle data serveru jako text končící zalomením řádku.
`ja.available()`	–	Vrátí počet bytů přijatých klientem od serveru.
`ja.read()`	–	Přečte a vrátí hodnotu přijatého bytu.
`ja.flush()`	–	Vyprázdní frontu přijatých a nepřečtených dat.
`ja.stop()`	–	Odpojí se od serveru.

Poslední třída `EthernetUDP` slouží k přijímání a odesílání UDP zpráv. My se jí však nebudeme zabývat. Pro více informací navštivte dokumentaci[41].

57.2 Vytváříme server

Na úvod si naprogramujeme jednoduchý server. Než ale začneme, musíme pochopit, jak spolu komunikuje server s klientem. Komunikace zde probíhá na principu dotaz–odpověď. Tyto dotazy a odpovědi mají formát pouhého textu. Jakým způsobem musí být text zapsán, definuje HTTP protokol. Nejjednodušší bude si vše předvést na ukázce komunikace. Když se chce klient připojit na server, zašle mu požadavek přibližně v tomto tvaru:

```
1  GET / HTTP/1.1
2  Host: 10.0.0.15
3  Connection: keep-alive
4  Cache-Control: max-age=0
5  Accept: text/html,application/xhtml+xml, application/xml;q=0.9,
       image/webp,*/*;q=0.8 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
       AppleWebKit/537.36 (KHTML, like Gecko)
6  Chrome/35.0.1916.114 Safari/537.36
7  Referer: http://10.0.0.15/
8  Accept-Encoding: gzip,deflate,sdch
9  Accept-Language: cs,en;q=0.8,de;q=0.6,sk;q=0.4
10
11 --konec--
```

Tato zpráva obsahuje vše potřebné pro server pro odeslání vhodných dat klientovi. Obsahuje informaci o tom, s jakou verzí HTTP pracujeme, jaký je pro klienta přijatelný formát, o jakého klienta se jedná, jaké kódování používá a jaký jazyk očekává. Prázdný desátý řádek zde není z nepozornosti. Tímto způsobem se označuje, že je požadavek ukončen.

Server poté musí požadavek zpracovat a odeslat zpět odpověď v HTML tvaru s vhodnou HTTP hlavičkou. Hlavička obsahuje informace o verzi HTTP, o stavu připojení po ukončení přenosu nebo o automatickém obnovování. Tyto informace nám nyní stačí. Existuje ale samozřejmě celá řada dalších HTTP příkazů,

[41] Dokumentace Ethernet shieldu – `http://arduino.cc/en/Reference/Ethernet`

jejichž seznam nalezneme například na Wikipedii[42]. Hlavička odpovědi tedy vypadá takto (opět s volným řádkem na konci).

```
HTTP/1.1 200 OK
Content-Type: text/html
Connection: close
Refresh: 1

--konec--
```

Po hlavičce a volném řádku následuje kód stránky ve formátu HTML. Představme si nyní základní strukturu HTML stránky.

```
<!DOCTYPE HTML> - rikame prohlizeci, ze pracujeme s HTML
<html>
    <head>
        mezi znacky head se pisi zakladni informace o strance (kodovani, CSS
            styly...)
        <title>Titulek stranky </title> - zobrazi se v zalozce prohlizece
    </head>
    <body>
        sem patri obsah stranky (ten, ktery vidime v prohlizeci)
    </body>
</html>
```

Se všemi získanými informacemi už můžeme poskládat program jednoduchého serveru, který bude měnit barvu pozadí pomocí CSS stylů podle toho, jestli je nebo není stisknuto tlačítko připojené k Arduinu na pinu 45. Tlačítko budeme kontrolovat každou vteřinu. Mimo tlačítka budeme potřebovat ještě 10 kΩ rezistor a několik vodičů.

V programu použijeme také jednoduché CSS stylování. My budeme chtít, aby pozadí celé stránky bylo zelené, nebo červené.

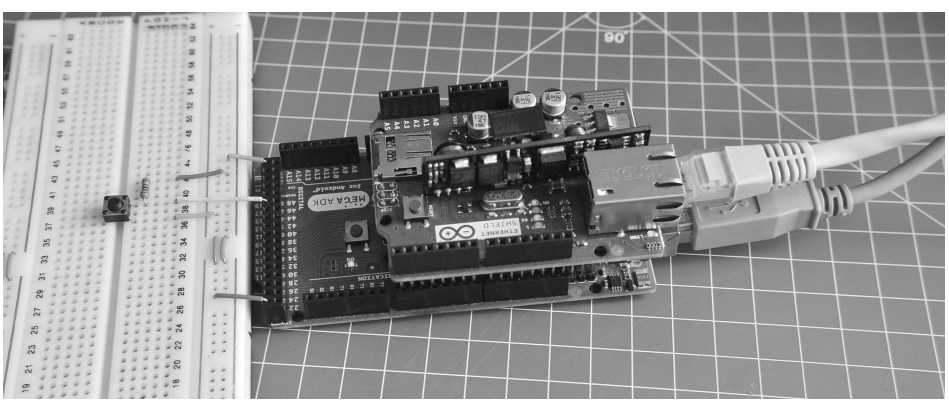
To se v HTML provede tak, že se k tagu `<body>` připojí `style="background: red/green"`. Dvojité uvozovky by ale program Arduina mohly mást, proto se používá tzv. escapování, kdy se před vybraný znak dá zpětné lomítko. Ten se poté projeví až při zpracování prohlížečem. Výsledný element body tedy bude vypadat třeba takto:

```
<body style="background: green">
```

```
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = {0x90, 0xA2, 0xDA, 0x00, 0x9C, 0xB7};
IPAddress ip(10,0,0,15); //ip serveru je 10.0.0.15

EthernetServer mujSvr(80); //vytvorime server na portu 80

void setup(){
    Ethernet.begin(mac);

    mujSvr.begin(); //spustime server

    pinMode(45, INPUT);
}

void loop(){
    EthernetClient client = mujSvr.available();
    if (client){
        boolean prazdnyRadek = true;
        while (client.connected() && client.available()){
            //dokud klient neco odesila (HTTP pozadavek)
```

[42] HTTP hlavička – `http://en.wikipedia.org/wiki/List_of_HTTP_header_fields`

```
23              char c = client.read(); //precti byte od klienta
24
25              if(c == '\n' && prazdnyRadek){
26                  client.println("HTTP/1.1 200 OK");
27                  client.println("Content-Type: text/html");
28                  client.println("Connection: close");
29                  client.println("Refresh: 1");
30                  client.println();
31                  client.println("<!DOCTYPE HTML>");
32                  client.println("<html>");
33                  client.println("<head>");
34                  client.println("<title>Zkoumame HTML a HTTP</title>");
35                  client.println("</head>");
36                  if(digitalRead(45) == HIGH){
37                      client.println("<body style=\"background:green\">");
38                  }
39                  else{
40                      client.println("<body style=\"background:red\">");
41                  }
42                  client.println("</body>");
43                  client.println("</html>");
44              }
45
46              if(c == '\n'){
47                  prazdnyRadek = true;
48              }
49              else if(c != '\r'){
50                  prazdnyRadek = false;
51                  /*dokud klient neposle dvakrat za sebou \r a \n
52                  znamena to, ze stale odesila data*/
53              }
54          }
55          delay(1); //dame klientovi cas na zpracovani
56          client.stop(); //komunikace je u konce
57      }
58  }
```



Obrázek 57.4: Arduino Mega s Ethernet shieldem a připojeným tlačítkem

57.3 Načítáme data

V druhém příkladu se připojíme k serveru a budeme po něm požadovat nějaká data, která si vypíšeme po sériové lince. Konkrétně to bude server `http://arduino.cz`, po kterém budeme chtít soubor `ahoj.txt`.

Ten je uložen v kořenovém adresáři serveru, tedy přímo na adrese `arduino.cz/ahoj.txt`. Na začátek si sestavíme HTTP požadavek.

```
GET /ahoj.txt HTTP/1.1
Host: arduino.cz
Connection: close
```

Slovy: Dej mi soubor `ahoj.txt` přes protokol HTTP verze 1.1 z `arduino.cz` a potom ukonči spojení.

```
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = {0x90, 0xA2, 0xDA, 0x00, 0x9C, 0xB7};

char server[] = "arduino.cz"; //server, kam se pripojujeme

EthernetClient client;

void setup(){
    Serial.begin(9600);
    Ethernet.begin(mac);
    delay(1000);

    Serial.println("Spojuji...");

    if(client.connect(server, 80)){ //pripoji se k serveru
        Serial.println("Pripojeno!");

        client.println("GET /ahoj.txt HTTP/1.1");
        client.println("Host: arduino.cz");
        client.println("Connection: close");
        client.println();
    }
    else {
        Serial.println("Spojeni se nepovedlo.");
    }
}

void loop(){
    if(client.available()) {
        char c = client.read();
        Serial.print(c); //vypise prijata data
    }

    if(!client.connected()) {
        Serial.println();
        Serial.println("Odpojuji.");
        client.stop();
        while(true){} //zastavi cinnost shieldu
    }
}
```

57.4 Ovládání přes síť

V posledním příkladu si ukážeme, jak Arduino ovládat pomocí prohlížeče či jiného síťového zařízení. Budeme ovládat čtyři LED připojené na pinech 3, 4, 5 a 6. Informaci o tom, která LED bude svítit, předáme shieldu pomocí parametru v URL (to co píšeme do adresního řádku). Parametr se píše za znak `?`. My tedy budeme v HTTP požadavku klienta hledat `?` a poté načítat čísla za ním následující. Náš program poté rozsvítí postupně LED diody daných čísel. Pokud napíšeme do adresního řádku prohlížeče: `10.0.0.15/?3456`, HTTP požadavek odeslaný na server vypadá nějak takto.

```
GET /?3456 HTTP/1.1
Host: 10.0.0.15
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml, application/xml;q=0.9,
    image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
    (KHTML, like Gecko) Chrome/35.0.1916.114 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: cs,en;q=0.8,de;q=0.6,sk;q=0.4

--konec--
```

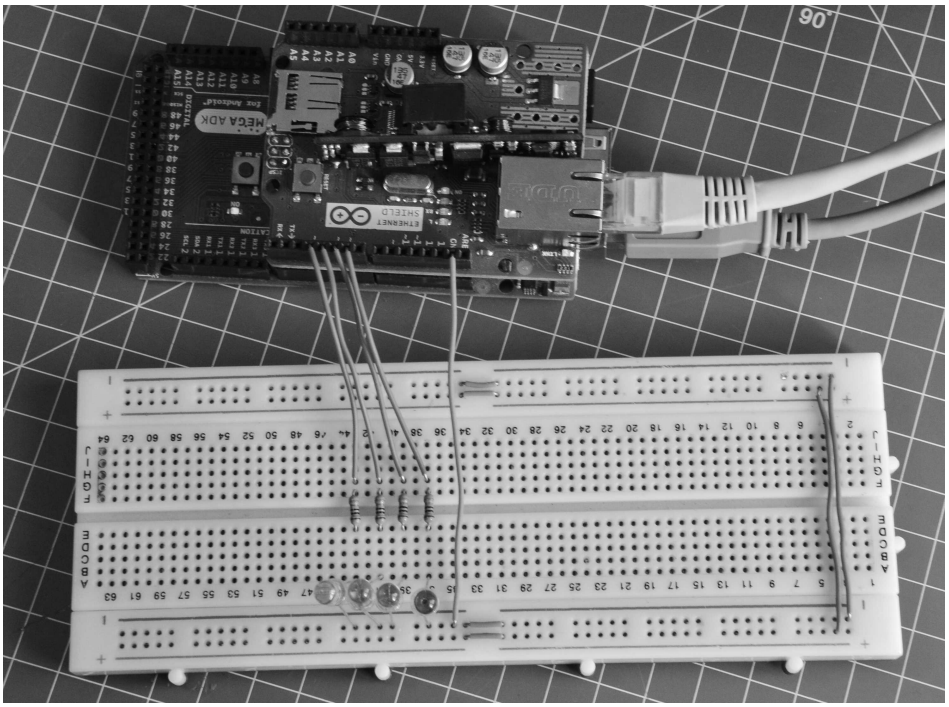
Jediná věc, která nás teď zajímá, je první řádek, a to až za otazníkem. Poté už nás další informace nezajímají. Data, která chceme, tedy začínají otazníkem a končí mezerou. Pozor na to, že jsou tu i číslice kódovány jako ASCII.

```
#include <Ethernet.h>
#include <SPI.h>

boolean zacatekCteni = false;

byte mac[] = {0x90, 0xA2, 0xDA, 0x00, 0x9C, 0xB7 };
IPAddress ip(10,0,0,15);

EthernetServer mujSvr = EthernetServer(80);

void setup(){
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);

  Ethernet.begin(mac, ip);

  mujSvr.begin();
}

void loop(){
    EthernetClient client = mujSvr.available();

    if(client){
        boolean prazdnyRadek = true;
        boolean hlavickaPoslana = false;

        while(client.connected() && client.available()){
            if(!hlavickaPoslana){ //jednou posleme hlavicku
                client.println("HTTP/1.1 200 OK");
                client.println("Content-Type: text/html");
                client.println();
                hlavickaPoslana = true;
            }

            char c = client.read();
            if(c == '?'){ //zacatek cteni
                zacatekCteni = true;
            }
            if(zacatekCteni && c == ' '){ //pokud uz nacita data a objevi se
                mezera, ukoncim nacitani
                zacatekCteni = false;
            }

```

```
45              if(zacatekCteni){
46                  if(c == '3'){
47                      blikni(3, client);
48                  }
49                  else if(c == '4'){
50                      blikni(4, client);
51                  }
52                  else if(c == '5'){
53                      blikni(5, client);
54                  }
55                  else if(c == '6'){
56                      blikni(6, client);
57                  }
58              }
59
60              if (c == 'n') {
61                  prazdnyRadek = true;
62              }
63              else if (c != 'r'){
64                  prazdnyRadek = false;
65              }
66          }
67          delay(1);
68          client.stop();
69      }
70  }
71
72  void blikni(int pin, EthernetClient client){
73      client.print("Sviti LED na pinu ");
74      client.print(pin);
75      client.print("<br>"); //zalomeni radku
76
77      digitalWrite(pin, HIGH);
78      delay(250);
79      digitalWrite(pin, LOW);
80      delay(250);
81  }
```

Tímto jsme získali základní přehled o tom, co Ethernet shield umí. To ale samozřejmě není vše. Můžeme ho například naučit komunikovat s Twitterem, zjišťovat čas podle atomových hodin a další. Několik zajímavých příkladů nalezneme v dokumentaci[43].

[43] Dokumentace Ethernet shieldu – `http://arduino.cc/en/Reference/Ethernet`



Obrázek 57.5: Ovládání LED diod přes Ethernet shield

Část XVI

Náš první klon Arduina

V této části si ukážeme, jak se dají pomocí Arduina programovat jiné čipy od Atmelu. Na začátek si předvedeme, jak pracovat s malými čipy z řady ATtiny, poté se dostaneme k větším čipům z řady ATmega a jak už název napovídá, vytvoříme vlastní klon Arduina.

Už na začátku knihy jsme se zmínili o neoficiálních deskách, tzv. klonech. Nejedná se jenom o sériově vyráběné desky. Takovýto klon si může každý udělat sám. Ještě než se pustíme do stavby vlastní desky, ukážeme si, jak se dají programovat menší čipy z řady ATtiny. A poté už se dostaneme ke tvorbě vlastní plnohodnotné desky.

Kapitola 58
Příprava Arduina

Než se pustíme do připojování a programování čipů, musíme z Arduina udělat ISP programátor. Ten slouží k nahrávání programů do připojených čipů. To provedeme velmi jednoduše – z *Examples* otevřeme program *ArduinoISP* a nahrajeme ho do našeho Arduina. V dalším kroku musíme „identifikovat“ důležité piny, které při programování použijeme. Jsou to: MISO, MOSI, SCK a SS. Jejich umístění se u různých desek může lišit. Příklad rozmístění pinů vidíte v tabulce.

Model	MISO	MOSI	SCK	SS
Mega	50	51	52	53
Uno	12	11	13	10

Informace o pinech jsou k nalezení v dokumentaci každé desky. Některé verze ale nemusejí mít tyto piny lehce dostupné. Například Leonardo je nemá vyvedeny standardně. O jeho použití jako ISP programátor pojednává tento článek[44].

Na vybraném čipu si podle dokumentace najdeme piny MISO, MOSI, SCK a RESET. Poté propojíme sobě odpovídající piny Arduina a čipu. Pin SS Arduina propojíme s pinem RESET čipu. Tímto se budeme podrobněji zabývat dále.

Kapitola 59
Čipy ATtiny

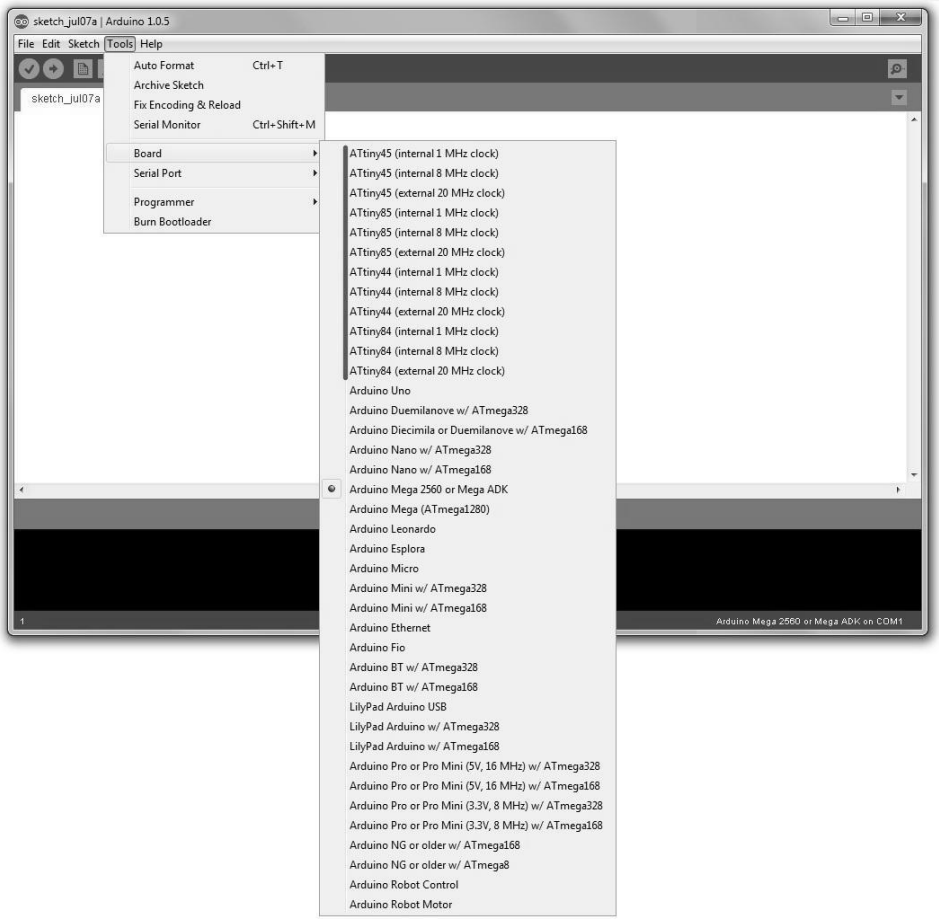
Jak už jsme zmínili dříve, v řadě ATtiny nalezneme menší a méně výkonnější čipy. To ale nevadí, protože nám v některých situacích budou plně dostačovat. Běžně se setkáme s použitím čipů ATtiny85, ATtiny45, ATtiny84, ATtiny44 a dalšími. Na internetu nalezneme i jiné čipy z této řady. Důležité však je, aby použitý čip měl k sobě odpovídající „popis“ – informaci v podobě textu, která programu říká informace o čipu, jako jsou rychlost, velikost paměti, rychlost komunikace a další. Soubor s informacemi o běžných deskách nalezneme ve složce s Arduino IDE pod `hardware/arduino/boards.txt`. Abychom si tento soubor nezahltili informacemi, umístíme data o čipech ATtiny do složky, kterou máme nastavenou pro ukládání programů Arduina. Tu většinou najdeme v `Dokumentech` ve složce `Arduino`. Stáhneme si tento archiv[45] a rozbalíme jej do složky `Arduino` v `Dokumentech`. Pokud máme spuštěné Arduino IDE, restartujeme jej. V nabídce *Board* by se nám nyní mělo objevit několik nových možností.

My si předvedeme, jak programovat čip ATtiny85. Ten má osm pinů – dva piny pro napájení, jeden pro restart a pět zbývá na libovolné použití. Popis funkcí pinů vidíme na obrázku. Orientaci čipu určíme buď podle půlkruhového symbolu na jedné ze stran (viz obr. 59.3) nebo podle označení pinu 1 kroužkem. Když budeme chtít v programu pracovat s pinem, dovoláme se ho podle číselného označení popsaného zvenku (čísla uvnitř čipu nás v tuto chvíli nezajímají).

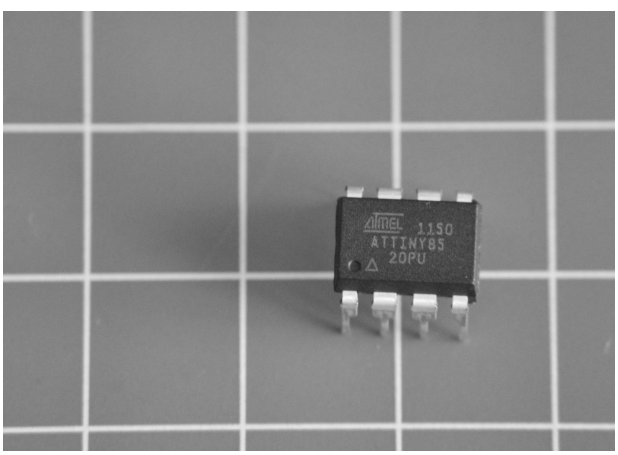
Jelikož se jedná o malé čipy s menším výkonem, i škála použitelných funkcí je zde omezena. Většinu základních funkcí zde ale najdeme. Jsou to:

- `pinMode()`
- `digitalWrite()`
- `digitalRead()`
- `analogRead()`
- `analogWrite()` – Připomeňme si, že je tato funkce použitelná jen u pinů PWM.
- `millis()`
- `micros()`
- `delay()`
- `delayMicroseconds()`

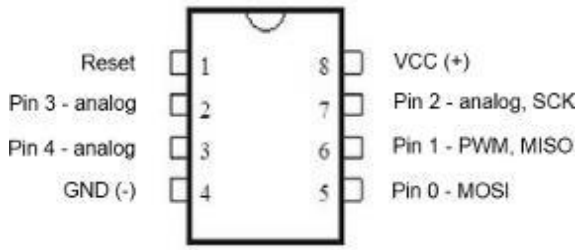
[44] Leonardo jako ISP – `http://www.instructables.com/id/Arduino-Leonardo-as-Isp/`
[45] `Boards.txt` – `http://arduino.cz/?attachment_id=976`



Obrázek 59.1: Menu *Board*



Obrázek 59.2: ATtiny 85

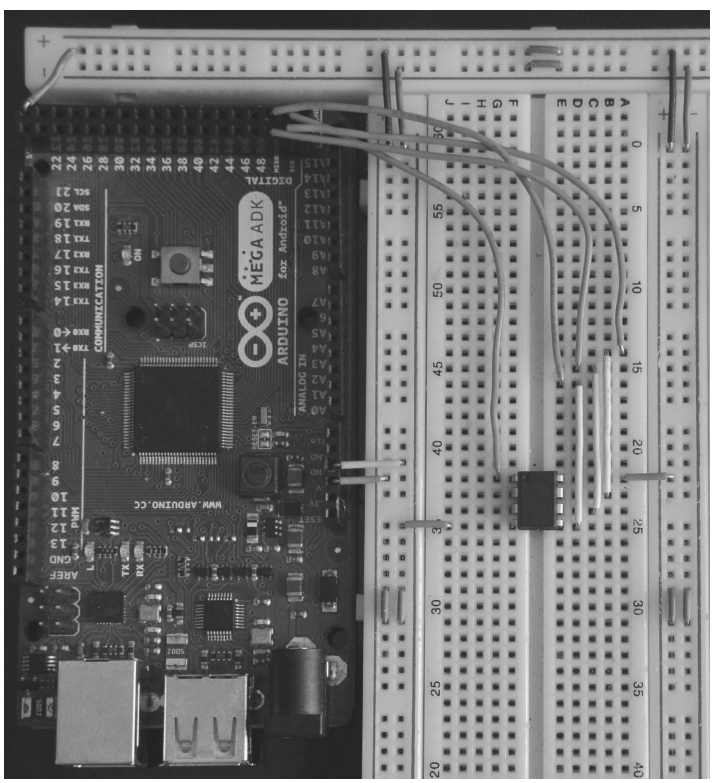


Obrázek 59.3: ATtiny 85 – rozložení pinů [38]

Také můžeme použít ještě neprobrané funkce `shiftOut()` a `pulseIn()`.

Základ použití máme vysvětlený a můžeme se pustit do propojení Arduina s čipem. Propojení provedeme následovně:

Arduino	**Čip**
MISO	MISO
MOSI	MOSI
SCK	SCK
SS	RESET
GND	GND
5V	5V



Obrázek 59.4: Zapojení

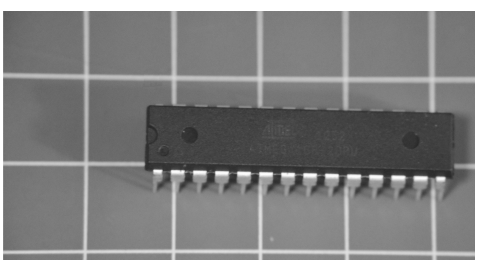
Po zapojení se ujistíme, že v Arduinu máme nahraný program ArduinoISP. Čipy ATtiny85 jsou ve výchozím nastavení taktované na frekvenci 1 MHz, proto z nabídky *Board* vybereme možnost *ATtiny85 (internal 1 MHz clock)*. Pro otestování funkčnosti připojíme mezi pin 0 (MOSI) a GND přes 330 Ω rezistor LED. Z *Examples* otevřeme program *Blink* a proměnnou `led` nastavíme na hodnotu 0.

```
1  int led = 0;
2
3  void setup() {
4    pinMode(led, OUTPUT);
5  }
6
7  void loop() {
8    digitalWrite(led, HIGH);
9    delay(1000);
10   digitalWrite(led, LOW);
11   delay(1000);
12 }
```

Změnu taktovací frekvence na 8 MHz provedeme jednoduše – z nabídky vybereme možnost ATtiny85 (internal 8 MHz clock) *a v menu* Tools *klikneme na* Burn Bootloader. *Stejným postupem čip nastavíme zpět na frekvenci 1 MHz, jen s výběrem* ATtiny85 (internal 1 MHz clock).

Kapitola 60
Čipy ATmega

Pokud se chceme od malých čipů ATtiny přesunout k něčemu většímu, nabízí se nám použití čipů ATmega. Ty jsou větší, výkonnější a mají více použitelných pinů. Stejně jako u ATtiny i zde platí, že můžeme použít všechny čipy, ke kterým nalezneme patřičnou modifikaci souboru `boards.txt`. Jako nejjednodušší se jeví použití těch čipů, na kterých jsou založeny standardní desky Arduina, tedy ATmega168, ATmega328 a další. My si ukážeme použití čipu ATmega168. Stejně jako u ATtiny, i zde musíme najít potřebné piny.

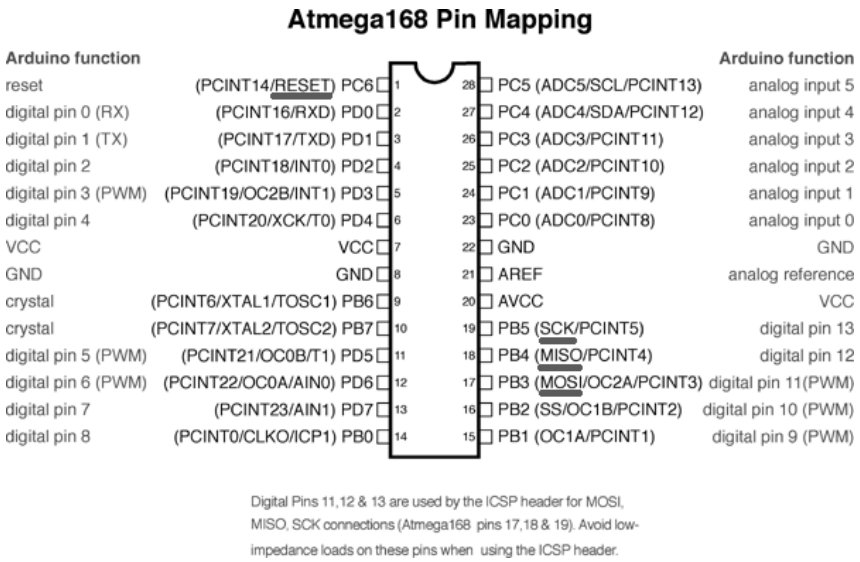


Obrázek 60.1: ATmega 168

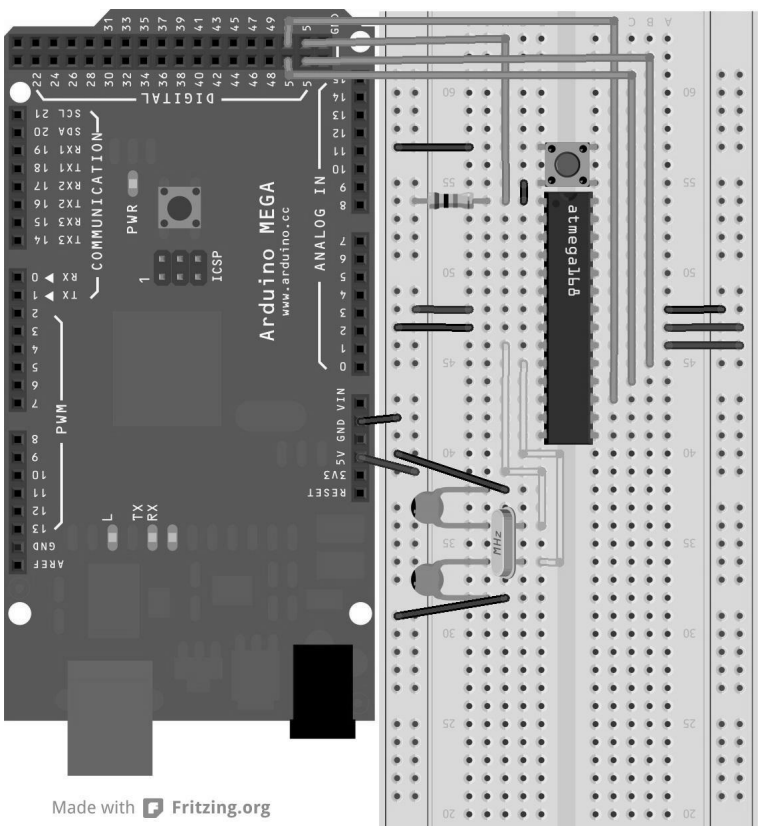
Abychom využili celý potenciál čipu, budeme potřebovat ještě 16MHz oscilátor, 10 kΩ rezistor, dva 22 pF kondenzátory a spínač (tlačítko). Také budeme potřebovat USB-serial převodník. Na výběr máme z více možností – FTDI Basic Breakout – 5V, USB-Serial Converter a další. Vše zapojíme podle schématu.

V našem Arduinu máme nahraný program `ArduinoISP`. Poté z nabídky *Board* vybereme *Arduino Diecimila or Duemilanove w/ ATmega168*. V *Tools/Programmer* vybereme možnost *Arduino as ISP* a spustíme *Burn Bootloader*. Po chvilce se ve stavovém řádku zobrazí „Done burning bootloader“. Tímto jsme nastavili čip tak, aby byl programovatelný pomocí USB-serial převodníku. V dalším kroku připojíme čip s převodníkem. To provedeme tak, že GND čipu propojíme s GND převodníku a stejně tak 5V. Dále propojíme RX pin čipu s TXD převodníku a TX pin čipu s RXD převodníku. Tím máme čip připravený k programování. Od čipu odpojíme Arduino, kterým jsme jej programovali a připojíme převodník k PC přes USB. V menu *Serial Port* by se nám měla objevit nová možnost, kterou vybereme. Poté se můžeme pustit do programování našeho nového klonu Arduina tak, jak jsme zvyklí.

Pokud nepůjde program do nového čipu nahrát, zkuste na chvíli před uploadem podržet tlačítko RESET.



Obrázek 60.2: ATmega 168 – rozložení pinů [37]



Obrázek 60.3: Zapojení Arduina na desce

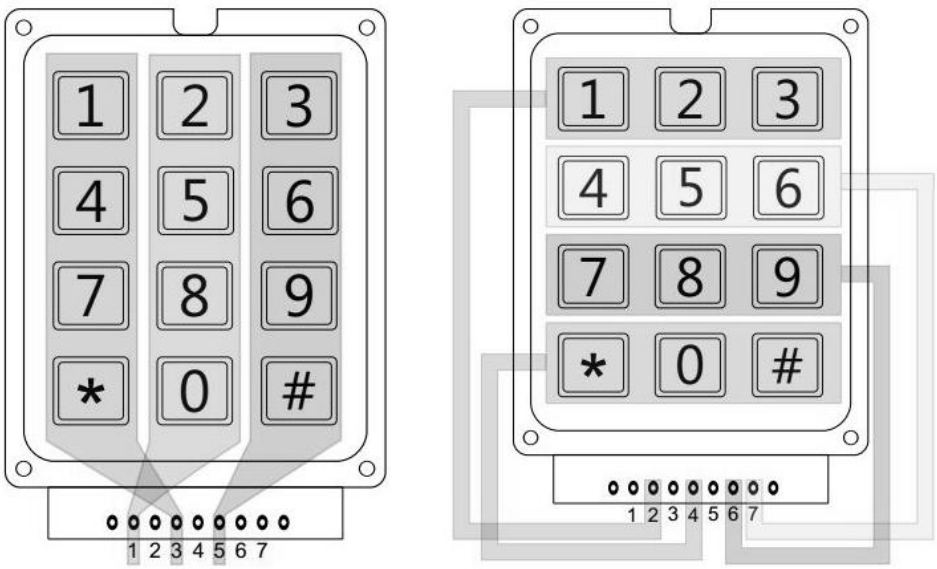
Část XVII

Projekt: Programátorská klávesnice

V této části si předvedeme, jak pracovat s keypadem s dvanácti tlačítky. Vytvoříme si velmi jednoduchý zabezpečovací systém a také si ukážeme, jak si při programování zkrátit čas (berte s rezervou) – vytvoříme si klávesnici pro programátory obsahující všechny potřebné závorky.

Kapitola 61
Keypad

Pro naše účely použijeme dvanáctitlačítkový keypad. Jeho tlačítka jsou uspořádána do matice 3×4. Jistě si vzpomínáte, jak jsme v kapitole věnované displejům ovládali maticový displej. Práce s keypadem je velmi podobná práci s displejem. Tlačítka jsou seřazena do třech sloupců a čtyř řádků, kdy každému je přiřazen jeden pin. Propojení je znázorněno na obrázku.



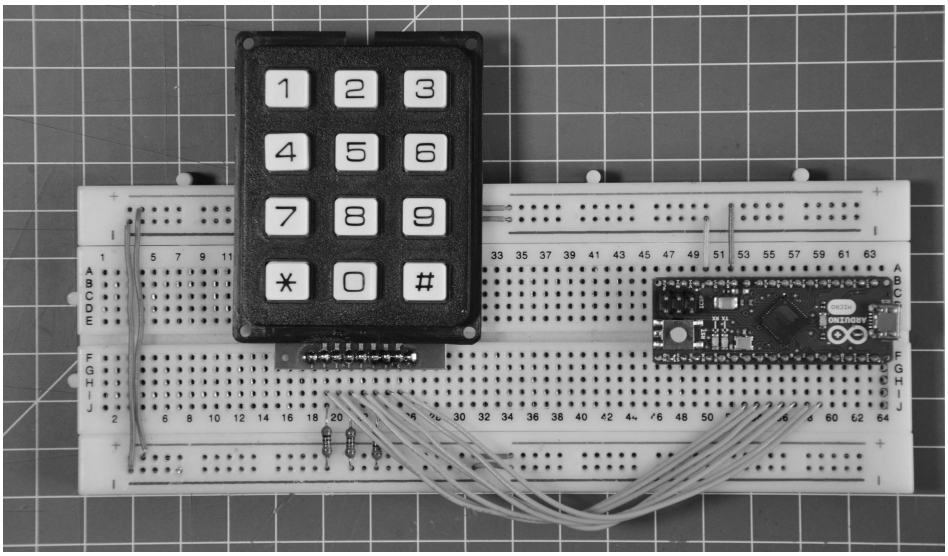
Obrázek 61.1: Vnitřní propojení keypadu [17]

Je tedy zřejmé, že při stisknutí tlačítka [1] jsou propojeny piny 2 a 3. Pokud bychom si chtěli připojit toto tlačítko k Arduinu, na pin 2 by bylo připojeno +5V a na pin 2 přes 10 kΩ rezistor GND a také digitální vstup (jako u standardně připojeného tlačítka). My ale potřebujeme zjistit stav všech dvanácti tlačítek. To provedeme tak, že si vybereme sloupce nebo řádky. Poté všechny piny jednoho typu připojíme na vstupy Arduina a přes 10 kΩ rezistor na GND. Druhou skupinu připojíme na výstupy. Na těch budeme stále dokola střídavě nastavovat napětí, které poté budeme detekovat na vstupech. K dispozici pak máme dvě informace: na jakém pinu nastavujeme jakou hodnotu a na kterém jsme jakou hodnotu detekovali. Z těchto informací už můžeme zjistit, jaké tlačítko je stisknuto.

61.1 Zapojení a programování

Jako první příklad si vytvoříme program, který po stisknutí tlačítka vypíše po sériové lince jemu odpovídající znak. Pokud tlačítko budeme držet stisknuté, znak se bude v určitém časovém intervalu opakovat. Můžeme si vybrat, jakým způsobem keypad zapojíme. Použijeme variantu, kdy řádky (2, 4, 6, 7) budou připojeny na výstupy Arduina a sloupce (1, 3, 5) na vstupy a také přes 10 kΩ rezistor na GND. V této části použijeme Arduino Micro, které budeme potřebovat při tvorbě klávesnice (také bychom mohli využít Arduino Leonardo). Pokud nevíte, proč budeme muset při tvorbě klávesnice použít právě tyto dvě desky, můžete si to připomenout v části věnované použití Arduina jako klávesnice či myši na straně 73. Keypad s Arduinem je možné propojit různými způsoby. My si zvolíme zapojení, kdy je pin n keypadu připojen na pin $n + 1$ Arduina. Pin 1 je tedy připojen na pin 2, pin 2 na pin 3 atd. Zapojení je vidět na obrázku č. 61.2.

Na začátku musíme programu říct, jaké piny jsou připojeny na co. To budeme mít uloženo ve dvou polích – `sloupce` a `radky`. Navíc si sloupce a řádky seřadíme zleva doprava a odshora dolů. Vytvoříme si dvojrozměrné pole se znaky odpovídajícími klávesám. Poté si piny odpovídající řádkům nastavíme jako výstupy a sloupce jako vstupy. V těle funkce `setup()` si ještě nastavíme aktuální čas. Ve funkci `loop()` budeme neustále dokola nastavovat napětí na jednotlivých řádcích a detekovat jeho velikost na sloupcích.

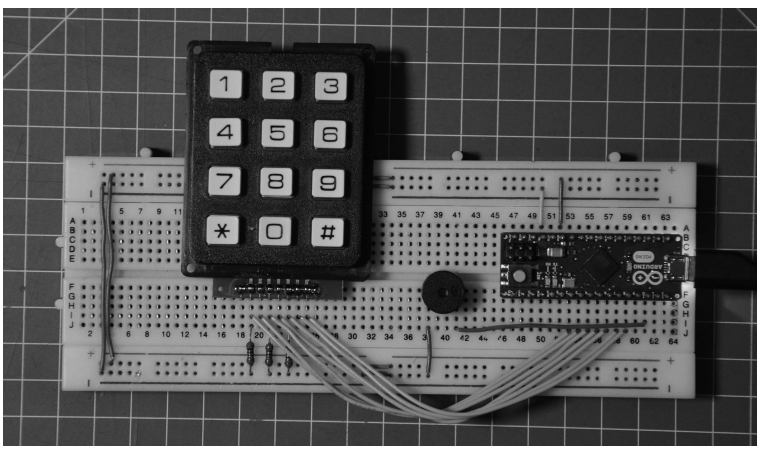


Obrázek 61.2: Zapojení keypadu

```
byte radky[4] = {3,8,7,5};
byte sloupce[3] = {4,2,6};

char znaky[4][3] = {{'1','2','3'},
                    {'4','5','6'},
                    {'7','8','9'},
                    {'*','0','#'}};

int cekej = 200;
long cas;

void setup(){
    for(int i = 0; i < 4; i++){
        pinMode(radky[i], OUTPUT);
    }
    for(int i = 0; i < 3; i++){
        pinMode(sloupce[i], INPUT);
    }

    cas = millis();
}

void loop(){
    for(int a = 0; a < 4; a++){
        digitalWrite(radky[a],HIGH);
        for(int b = 0; b < 3; b++){
            if(digitalRead(sloupce[b]) == HIGH && millis() - cekej > cas){
                Serial.println(znaky[a][b]);
                cas = millis();
            }
        }
        digitalWrite(radky[a],LOW);
    }
}
```

Kapitola 62
Bezpečnostní systém

V této části bude naším úkolem naprogramovat jednoduché bezpečnostní zařízení. To bude vybaveno keypadem pro zadávání kódu a piezzo bzučákem, který začne pípat při zadání špatného kódu. Ten bude připojený na pin 10. Stisknuté znaky z keypadu se budou nahrávat do řetězce. Znak `#` bude sloužit k vynulování řetězce a `*` k potvrzení zadání kódu. Vše je vysvětleno v programu níže. Vyjdeme z kódu z předchozího příkladu. Předpokládejme, že náš kód nebude delší než 100 znaků.



Obrázek 62.1: Keypad s bzučákem

```
byte radky[4] = {3,8,7,5};
byte sloupce[3] = {4,2,6};

char znaky[4][3] = {{'1','2','3'},
                    {'4','5','6'},
                    {'7','8','9'},
                    {'*','0','#'}};

int cekej = 200;
long cas;
byte pozice = 0; //pomocna promenna udavajici aktualni znak
byte chyby; //promenna pro ukladani neshod kodu
char spravnyKod[100] = "12321";
char kod[100];

void setup(){
    for(int i = 0; i < 4; i++){
        pinMode(radky[i], OUTPUT);
    }
    for(int i = 0; i < 3; i++){
        pinMode(sloupce[i], INPUT);
    }

    cas = millis();
}

void loop(){
    for(int a = 0; a < 4; a++){
        digitalWrite(radky[a],HIGH);
        for(int b = 0; b < 3; b++){
```

```
31              if(digitalRead(sloupce[b]) == HIGH
32              && millis() - cekej > cas){
33                  tone(10,440,100);
34                  cas = millis();
35
36                  if(znaky[a][b] == '#'){
37                  //pokud najde krizek, vymaze kod
38                      vynuluj();
39                  }
40                  else if(znaky[a][b] == '*'){
41                  //kdyz najde hvezdicku, zkontroluje shodu
42                      for(int c = 0; c < 100; c++){
43                          if(kod[c] != spravnyKod[c]){
44                              chyby++;
45                          }
46                      }
47                      if(chyby > 0){
48                          vynuluj();
49                          for(int d = 0; d < 10; d++){ //sirena
50                              for(int c = 200; c < 2000; c++){
51                                  tone(10, c, 5);
52                              }
53                              for(int c = 2000; c > 200; c--){
54                                  tone(10, c, 5);
55                              }
56                          }
57                      }
58                      else{
59                          vynuluj();
60                          tone(10, 1000, 100); //trikrat pipne
61                          delay(200);
62                          tone(10, 1000, 100);
63                          delay(200);
64                          tone(10, 1000, 100);
65                      }
66                  }
67                  else{
68                      kod[pozice] = znaky[a][b];
69                      pozice++;
70                  }
71              }
72          }
73          digitalWrite(radky[a],LOW);
74      }
75  }
76
77  void vynuluj(){ //funkce pro vymazani retezce kod
78      for(int c = 0; c < 100; c++){
79          kod[c] = 0;
80      }
81      pozice = 0;
82      chyby = 0;
83  }
```

Kapitola 63
Programátorská klávesnice

Posledním příkladem využívajícím keypad je slibovaná programátorská klávesnice. Ta naše bude používat jenom osm kláves keypadu – pro každý typ závorek dvojici kláves (levá a pravá závorka). ASCII kódy jednotlivých znaků v desítkové soustavě vidíte v tabulce.

Znak	ASCII kód
(	40
)	41
{	123
}	125
[	91
]	93
<	60
>	62

Tyto kódy přepíšeme do pole `znaky`. Stisknutí klávesy do PC odešleme pomocí funkce `Keyboard.print()` (viz Arduino jako klávesnice a myš na straně 73). Dále je kód stejný jako v předchozích příkladech. Když odešleme tyto znaky do PC, budou správně fungovat, jen pokud máme nastavenou anglickou klávesnici. Pokud tomu tak je, použijeme pro odeslání znaku funkci `zmackniAnglicky` – ta odešle pouze hodnotu znaku. Pokud ale máme nastavenou českou klávesnici, musíme před odesláním znaku přepnout na anglickou a po odeslání zase zpět. K přepínání jazyků slouží klávesová zkratka `L_ALT + L_SHIFT`. Pro speciální klávesy má Arduino vyhrazené konstanty, o nichž nalezneme podrobnější informace zde[46]. Kód klávesnice vypadá následovně.

```
byte radky[4] = {3,8,7,5};
byte sloupce[3] = {4,2,6};

char znaky[4][3] = {{40,41,' '},
                    {123,125,' '},
                    {91,93,' '},
                    {60,62,' '}};

int cekej = 200;
long cas;

void setup(){
    for(int i = 0; i < 4; i++){
        pinMode(radky[i], OUTPUT);
    }
    for(int i = 0; i < 3; i++){
        pinMode(sloupce[i], INPUT);
    }

    cas = millis();
}
void loop(){
    for(int a = 0; a < 4; a++){
        digitalWrite(radky[a],HIGH);
        for(int b = 0; b < 3; b++){
            if(digitalRead(sloupce[b]) == HIGH
            && (millis() - cekej) >= cas){
                zmackniCesky(znaky[a][b]);
```

[46] `http://arduino.cc/en/Reference/KeyboardModifiers`

```
29                cas = millis();
30            }
31        }
32        digitalWrite(radky[a],LOW);
33    }
34 }
35
36 void zmackniCesky(char znak){
37    Keyboard.press(KEY_LEFT_ALT);
38    delay(10);
39    Keyboard.press(KEY_LEFT_SHIFT);
40    delay(10);
41    Keyboard.releaseAll();
42    delay(10);
43    Keyboard.print(znak);
44    delay(10);
45    Keyboard.press(KEY_LEFT_ALT);
46    delay(10);
47    Keyboard.press(KEY_LEFT_SHIFT);
48    delay(10);
49    Keyboard.releaseAll();
50    delay(10);
51 }
52
53 void zmackniAnglicky(char znak){
54    Keyboard.print(znak);
55 }
```

Uvedené kódy mohou sloužit jako základ pro další tvorbu – jednoduchou úpravou se dá vytvořit například klávesnice posílající smajlíky.

Část XVIII

Projekt: Robotická ruka

Kapitola 64
Servomotory

Pro účely práce se servomotory je Arduino vybaveno knihovnou `Servo.h`. Tu do kódu vložíme příkazem `#include <Servo.h>`. Knihovna obsahuje několik funkcí a příkazů, z nichž nejpoužívanější jsou:

Název	Funkce
`Servo motor`	Vytvoří objekt `Servo` se jménem `motor` (jméno je volitelné).
`motor.attach(pin)`	Řekne programu, na kterém pinu je servo připojeno.
`motor.write(hodnota)`	Nastaví pozici serva, hodnota je většinou v rozmezí 0 až 180 (občas i méně).

Ze serva vychází tři vodiče – žlutý/oranžový, hnědý a červený, odpovídající postupně signálu (připojuje se k Arduinu), GND a +5V. Jednoduchý program, který vezme hodnotu naměřenou na pinu A0, upraví její rozsah a odešle ji jako pozici serva, vypadá takto.

```
1  #include <Servo.h>
2
3  Servo motor;
4  int h; //hodnota mereni
5
6  void setup(){
7      motor.attach(10); //pripojime servo na pin 10
8  }
9
10 void loop(){
11     h = map(analogRead(A5),0,1024,0,180);
12     motor.write(h);
13 }
```

To je k teorii řízení serv vše. Jednoduché, že? Nyní se tedy můžeme pustit do projektu robotické ruky.

Kapitola 65
Robotická ruka

Cílem tohoto projektu je vytvořit jednoduchou robotickou ruku, která bude mít na konci fixu. Tou budeme moct vytvářet kresby na papíru. Budeme potřebovat:

1. Dřevo na mechanické konstrukce – v našem případě dřevěná lať ka s průřezem 20×5 mm.
2. Tavná pistole (nebo šrouby a herkules či jiné lepidlo – v závislosti na provedení).
3. Vodiče.
4. Piny.
5. Dutinkovou lištu – třípinovou.
6. Kousek prototypovacího plošného spoje.
7. 2× 10 kΩ potenciometr s lineárním průběhem.
8. Velké servo s konstrukcí[47].
9. Střední servo[48].

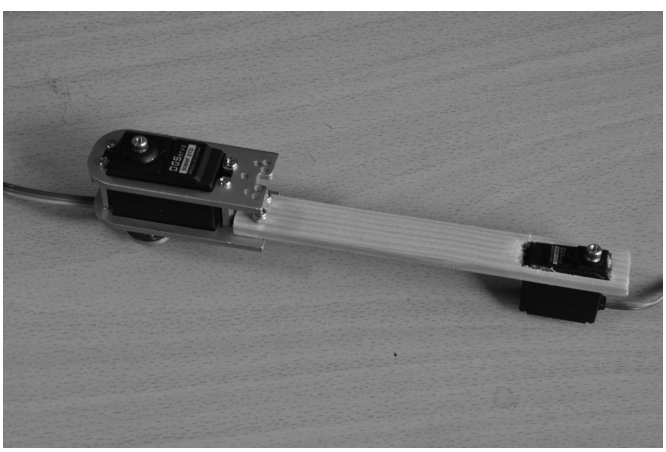
[47] Velké servo – `http://www.hwkitchen.com/products/robotic-claw-pan-tilt-bracket-mkii/`
[48] Střední servo – `http://www.hwkitchen.com/products/servo-medium/`

10. Klip na papíry o velikosti pera, se kterým budeme kreslit.
11. Dřevěnou desku o rozměrech alespoň 30×40 cm.

Začneme tím, že si nařežeme dřevěné části. Uřízneme jednu 17 cm dlouhou, jednu 15 cm dlouhou a dvě třícentimetrové části. Na jeden konec patnácticentimetrové laťky navrtáme dvě dírky se stejnou roztečí, jako ty na konstrukci u velkého serva. Na druhý konec vyřežeme obdélníkovou díru velikosti menšího serva. Sem servo vložíme a zalepíme nebo přišroubujeme.

Obrázek 65.1: Menší rameno

V dalším kroku přišroubujeme vytvořené rameno ke konstrukci většího serva.



Obrázek 65.2: Kratší rameno s připevněnými motory

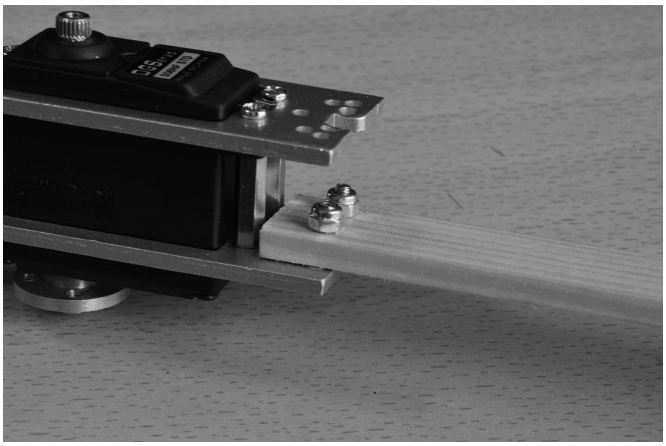
Budeme pokračovat úpravou delší laťky. Na jeden konec nalepíme plastový kulatý převod menšího serva plochou stranou ke dřevu. Slepíme dva nejmenší kousky dřeva největší stranou k sobě a přilepíme je v pravém úhlu na druhý konec laťky. Na takto vzniklou podpěru přilepíme klip na papíry.

Na dřevěnou desku přilepíme nebo přišroubujeme část konstrukce velkého serva (plastovou částí nahoru). Rozměry můžete vidět na obrázku.

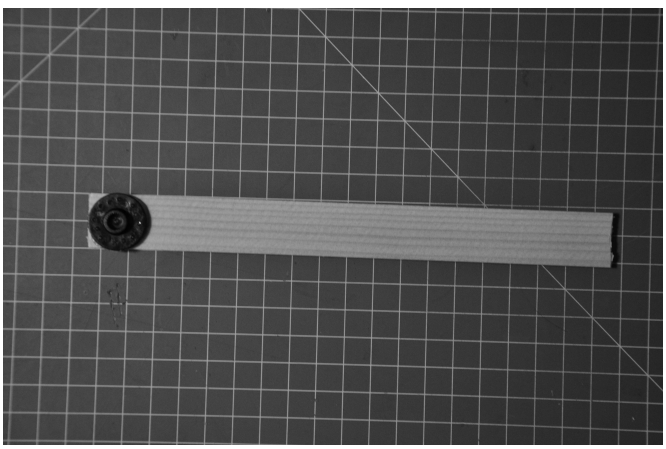
Velké servo připojíme k Arduinu a funkcí `.write()` ho nastavíme na polohu 105. Poté nasadíme první rameno rovnoběžně s delším okrajem.

Připojíme malé servo a nastavíme ho na hodnotu 60. Poté připojíme druhé rameno kolmo k prvnímu.

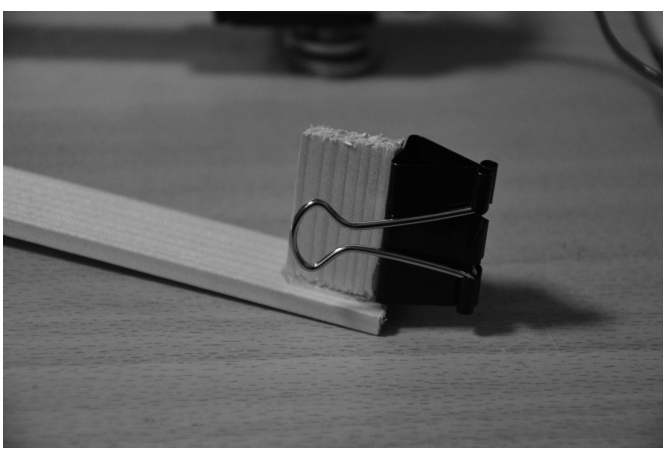
Jelikož má menší servo krátký kabel, musíme si vyrobit „prodlužku“. Budeme potřebovat třížilový kabel, na jehož jednom konci budou piny a na druhém zdířky.



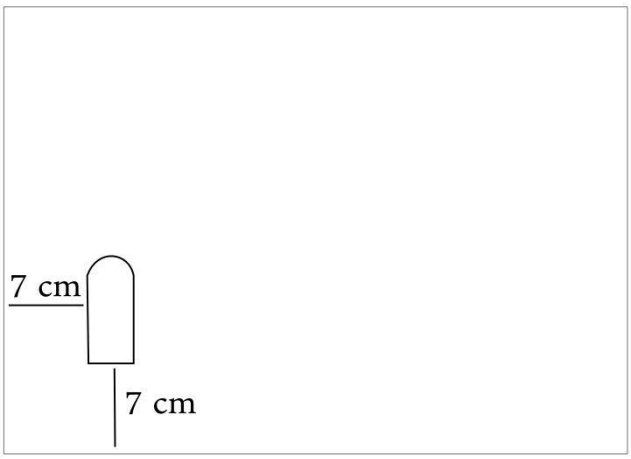
Obrázek 65.3: Detail



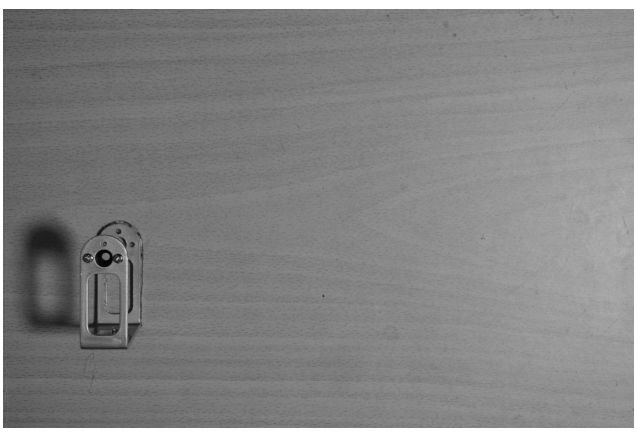
Obrázek 65.4: Delší rameno



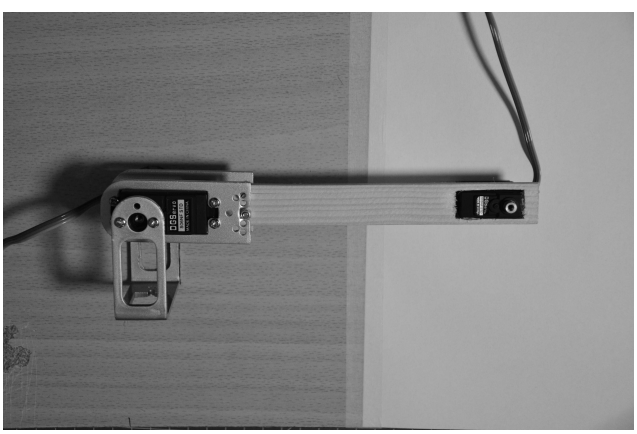
Obrázek 65.5: Detail



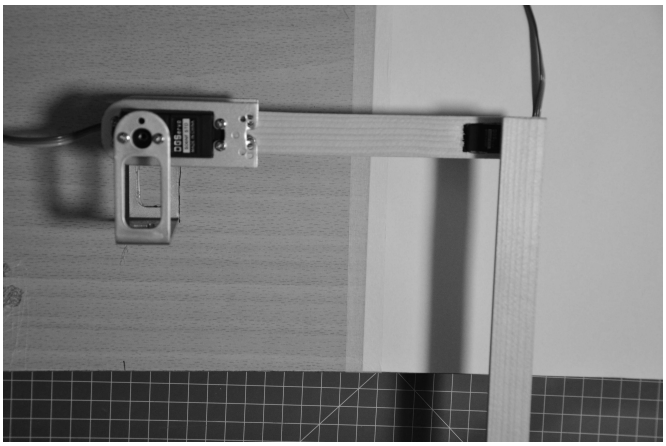
Obrázek 65.6: Nákres kreslicí plochy



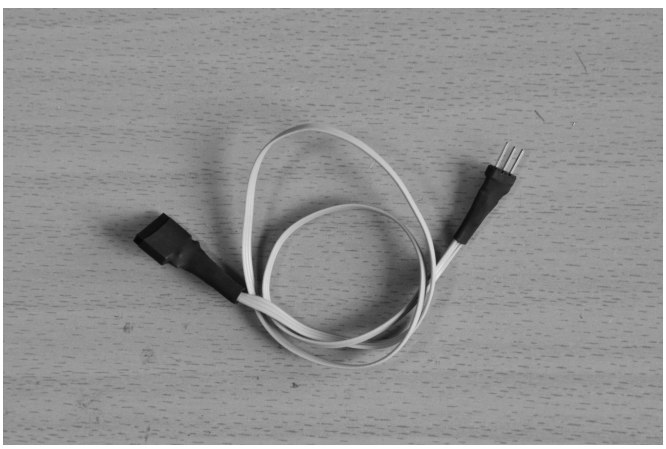
Obrázek 65.7: Foto kreslicí plochy



Obrázek 65.8: Připevnění kratšího ramene



Obrázek 65.9: Připevnění delšího ramene



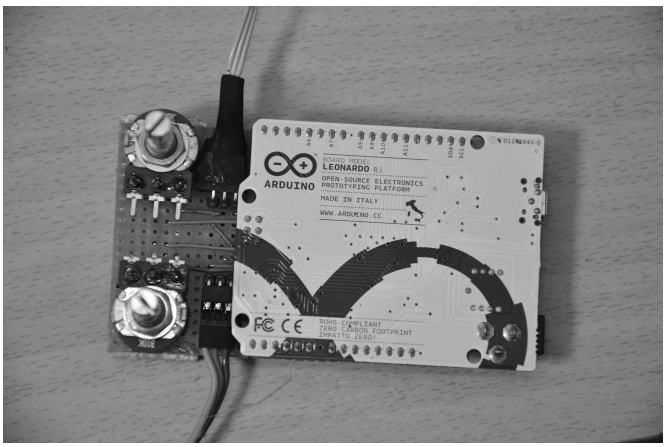
Obrázek 65.10: Propojovací kabel

Nyní přichází na řadu zapojení. V tomto příkladu je použito následující: Velké servo je připojeno na pin A3, malé na A2. Také využijeme dva potenciometry, kterými budeme nastavovat polohu serv. Ty jsou připojeny k pinům A5 a A4. Výběr pinů však záleží čistě na nás. Na obrázku na další straně vidíte příklad konstrukce ovládání.

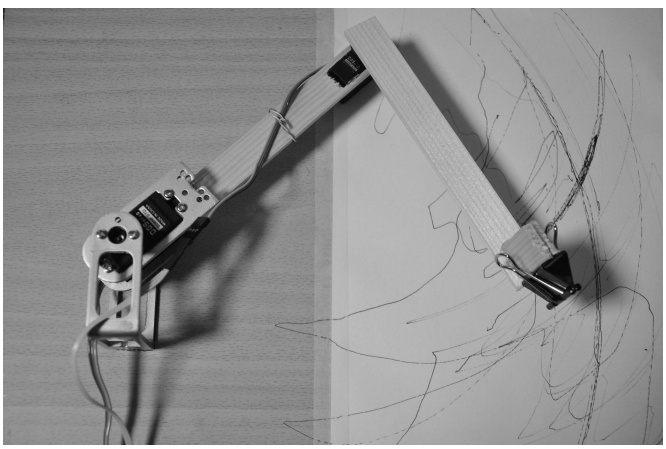
```
1  #include <Servo.h>
2
3  Servo sv, sm;
4  int h1, h2; //hodnoty mereni
5
6  void setup(){
7      sv.attach(A3);
8      sm.attach(A2);
9  }
10
11 void loop(){
12     h1 = map(analogRead(A5),0,1024,0,170);
13     h2 = map(analogRead(A4),0,1024,0,170);
14
15     sv.write(h1);
```

```
16      sm.write(h2);
17
18      Serial.println(h2);
19  }
```

Už máme vše sestaveno a zapojeno i naprogramováno. Na desku si izolepou upevníme papír A4 (má přibližně stejnou velikost jako naše deska), připevníme tužku, nebo fixu do klipu a můžeme začít malovat.



Obrázek 65.11: Ovládací deska



Obrázek 65.12: Hotové rameno

Pokud se vám zdá ovládání, kdy se potenciometry nastavují úhly serv neohrabané, máte pravdu. Poměrně jednoduše se dá ale vytvořit algoritmus, ve kterém budete mít možnost nastavit přímo souřadnice požadovaného bodu. Stačí k tomu jenom pár goniometrických funkcí – to už ale nechám na vás.

Část XIX

WiFi shield

Kapitola 66
Seznámení

Občas se hodí mít možnost s Arduinem komunikovat bezdrátově. Pro různé účely se hodí různé způsoby komunikace – někdy je vhodnější použít bluetooth, někdy potřebujeme větší dosah a šáhneme například po modulech nRF24L01+, a když jsme v dosahu nějaké WiFi sítě, můžeme použít WiFi shield. Jedná se o výrobek z oficiální produkce Arduina, tudíž k němu (jako k většině oficiálních desek) existuje výborná dokumentace. Na začátek si představme základní vlastnosti shieldu.

- WiFi shield se umí připojit k WiFi sítím se standardem 802.11b a 802.11g.
- Může se připojit k sítím bez hesla a také k těm se zabezpečením WEP a WPA2 Personal.
- Shield obsahuje slot na microSD kartu.
- S Arduinem shield komunikuje přes SPI rozhraní (stejně jako Ethernet shield).
- SS pin WiFi shieldu je vyveden na pinu 10. SS pin SD slotu je připojen na pin 4.
- WiFi a SD kartu není možné používat současně.
- Na desce také nalezneme mini USB port pro update firmware shieldu.
- Aby se mohl shield k síti připojit, musí síť vysílat svoje SSID (nesmí být skryto).
- Po zmáčknutí tlačítka RESET dojde k restartu Arduina i shieldu.
- Při práci se shieldem je doporučeno nepoužívat pin 7. Je totiž využíván knihovnou pro ovládání shieldu.

Na desce nalezneme několik LED diod:

Název	Barva	Funkce
L9	žlutá	Je připojena na pin 9.
LINK	zelená	Signalizuje připojení k WiFi.
ERROR	červená	Signalizuje problém s komunikací.
DATA	modrá	Signalizuje přenos dat.

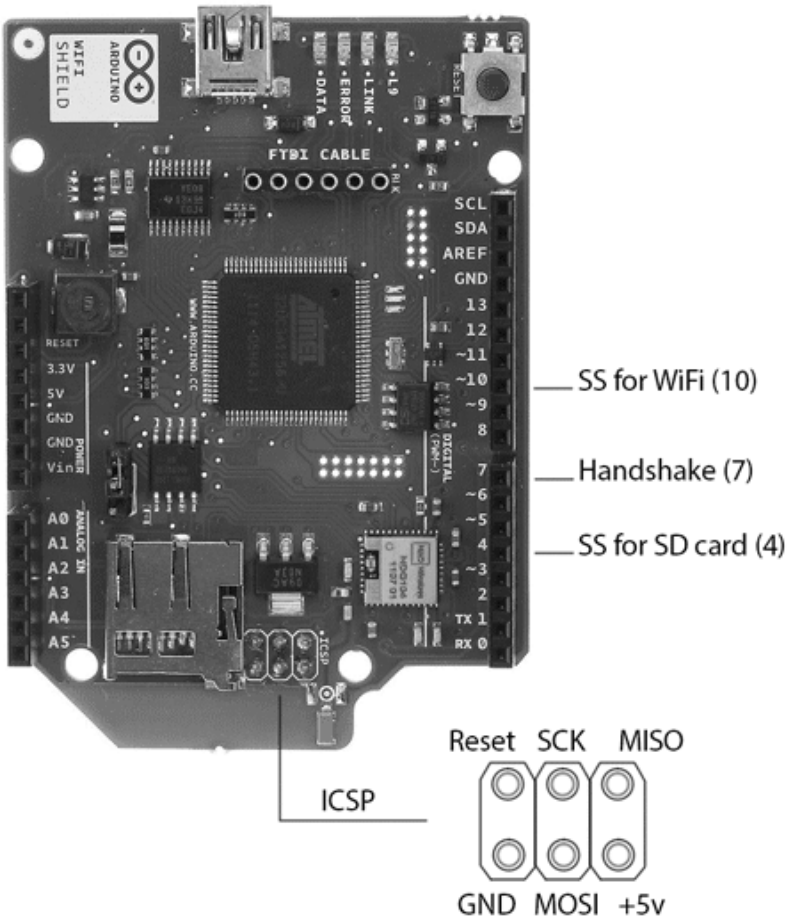
Pokud používáme shield s Arduinem starším než Arduino Uno rev3, musíme propojit 3,3 V s IOREF. Starší desky poznáme podle toho, že mají méně konektorů a pin IOREF shieldu zůstane nepřipojen. Pozor ale na to, abychom toto propojení nepoužili s novější deskou.

Pro ovládání slouží knihovna `WiFi.h`. Tu obsahuje Arduino IDE, tudíž ji nemusíme stahovat. Do kódu ji vložíme známým příkazem `#include <WiFi.h>`. Předtím, než se pustíme do programování shieldu, musíme ověřit, jestli je jeho firmware aktuální.

Kapitola 67
Firmware shieldu

67.1 Zjištění verze firmware

Přestože je poslední verze firmware používaná už dlouhou dobu, může se stát, že narazíme na shield využívající starší verzi. Abychom zajistili správnou funkčnost, je vhodné firmware aktualizovat. Nejdříve ale musíme zjistit, jakou verzi vlastně máme. To provedeme pomocí jednoduchého kódu, který nám po sériové lince vypíše verzi firmware. S funkcemi se zatím zaobírat nemusíme, vše bude vysvětleno později.

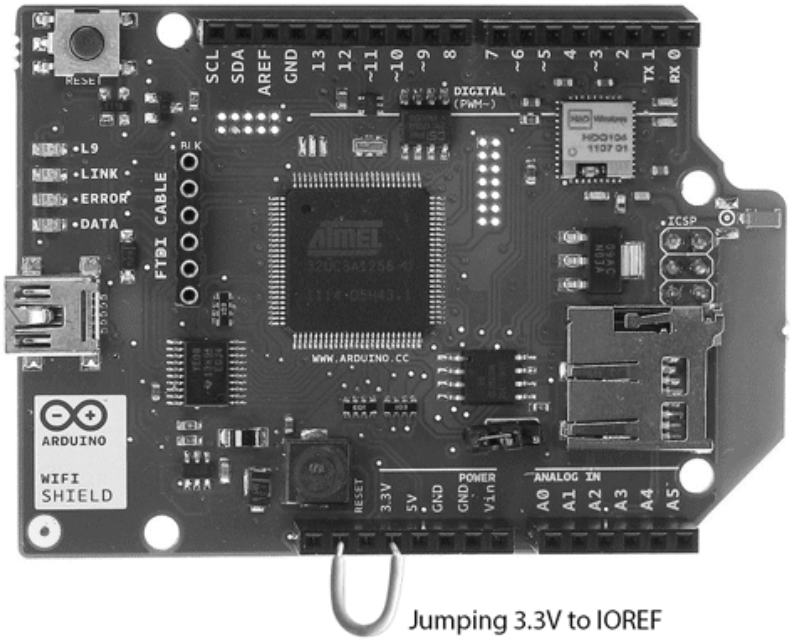


Obrázek 66.1: Piny WiFi shieldu [30]

```
1  #include <SPI.h>
2  #include <WiFi.h>
3
4  void setup() {
5      Serial.begin(9600);
6      if (WiFi.status() == WL_NO_SHIELD) {
7          Serial.println("WiFi shield je nepripojen");
8          while(true); //zacykli program – ten dale nepokracuje
9      }
10     Serial.print("Verze firmware: ");
11     Serial.println(WiFi.firmwareVersion());
12 }
13
14 void loop() {}
```

Uvedený postup je použitelný pro Windows. Návod upgrade pro Mac a Linux naleznete zde[49].
Pokud verze firmware není 1.1.0 (nebo vyšší), je vhodné provést aktualizaci.

[49] Návod na update firmware – `http://arduino.cc/en/Hacking/WiFiShieldFirmwareUpgrading`



Obrázek 66.2: Propojení pro starší desky [30]

67.2 Aktualizace firmware

Před začátkem update musíme propojit dva piny, které umožní komunikaci s čipem AT32UC3 – mozkem shieldu. Jsou umístěny vedle microSD slotu. Procesor řídí chod čipu HDG104, který se stará o WiFi. Také bychom měli shield odpojit od Arduina.

Firmware obou čipů můžeme nahrát přes USB. K tomu ale budeme potřebovat speciální software, který se jmenuje FLIP. Je to oficiální software od Atmel a dá se stáhnout přímo z jeho stránek[50]. Po stažení vhodné verze spustíme instalaci a bez jakékoliv nutnosti konfigurace jej nainstalujeme. Z GitHubu[51] stáhneme nejnovější verzi firmware. Zde nás budou zajímat soubory `wifi_dnld.elf`[52] a `wifiHD.elf`[53], které stáhneme.

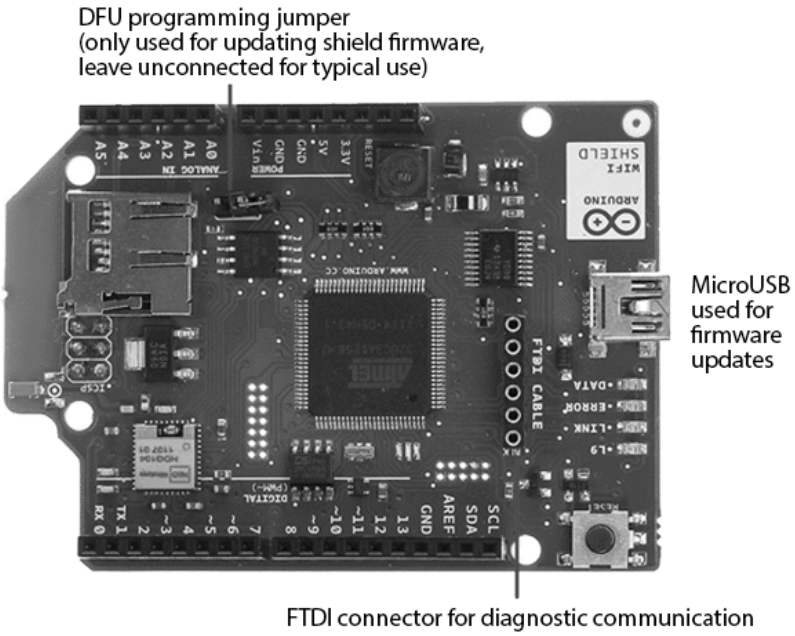
Poté spustíme příkazový řádek a přepneme se do složky, kde je nainstalovaný soubor FLIP (většinou `C:\Program Files (x86)\AtmelFlip X.X.X\bin`). Pozor! Může být nutné spouštět příkazový řádek s právy správce. V příkazovém řádku se do požadované složky přesuneme pomocí příkazu `cd` následovaným absolutní adresou, například tedy:

```
1 cd C:\Program Files (x86)\AtmelFlip 3.4.7\bin
```

Pomocí mini USB připojíme shield k počítači. Ve většině případů nedojde k automatické instalaci ovladačů a my je tedy budeme muset nainstalovat ručně. To provedeme tak, že otevřeme *Ovládací panely* a v nich s právy administrátora spustíme *Správce zařízení*. Pokud nedošlo ke správné instalaci driverů, uvidíme zde podobnou položku, jaká je na obrázku č. 67.3.

Pravým tlačítkem na něj klikneme a vybereme možnost *Aktualizovat software ovladače*, poté zvolíme *Vyhledat ovladač v počítači* a v následující nabídce otevřeme složku s nainstalovaným programem FLIP (např. `C:\Program Files (x86)\AtmelFlip X.X.X`). Kliknutím na tlačítko *Další* program nalezne potřebné ovladače a nainstaluje je. Nyní už je vše připraveno pro instalaci nového firmware. Postupně nahrajeme oba stažené soubory, oba pomocí příkazu níže. V příkladu jsou oba soubory uložené ve složce `C:\Users\uzivatel\Downloads`.

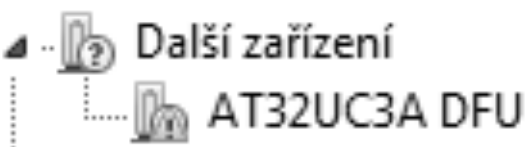
[50] FLIP – `http://www.atmel.com/tools/FLIP.aspx`
[51] GitHub s firmware – `https://github.com/arduino/Arduino/tree/master/hardware/arduino/firmwares/wifishield`
[52] `https://github.com/arduino/Arduino/tree/master/hardware/arduino/firmwares/wifishield/wifi_dnld/Release`
[53] `https://github.com/arduino/Arduino/tree/master/hardware/arduino/firmwares/wifishield/wifiHD/Release`



Obrázek 67.1: Konfigurační piny [30]

```
C:\>cd C:\Program Files (x86)\Atmel\Flip 3.4.7\bin
C:\Program Files (x86)\Atmel\Flip 3.4.7\bin>
```

Obrázek 67.2: Příkazový řádek



Obrázek 67.3: Neznámé zařízení

```
1 batchisp.exe -device AT32UC3A1256 -hardware usb -operation erase f memory
      flash blankcheck loadbuffer C:\cestaksouboru program verify start reset 0
```

Začneme souborem `wifi_dnld.elf`.

```
1 batchisp.exe -device AT32UC3A1256 -hardware usb -operation erase f memory
      flash blankcheck loadbuffer C:\Users\uzivatel\Downloads\wifi_dnld.elf
      program verify start reset 0
```

Po úspěšném uploadu shield restartujeme, a poté nahrajeme i soubor `wifiHD.elf`.

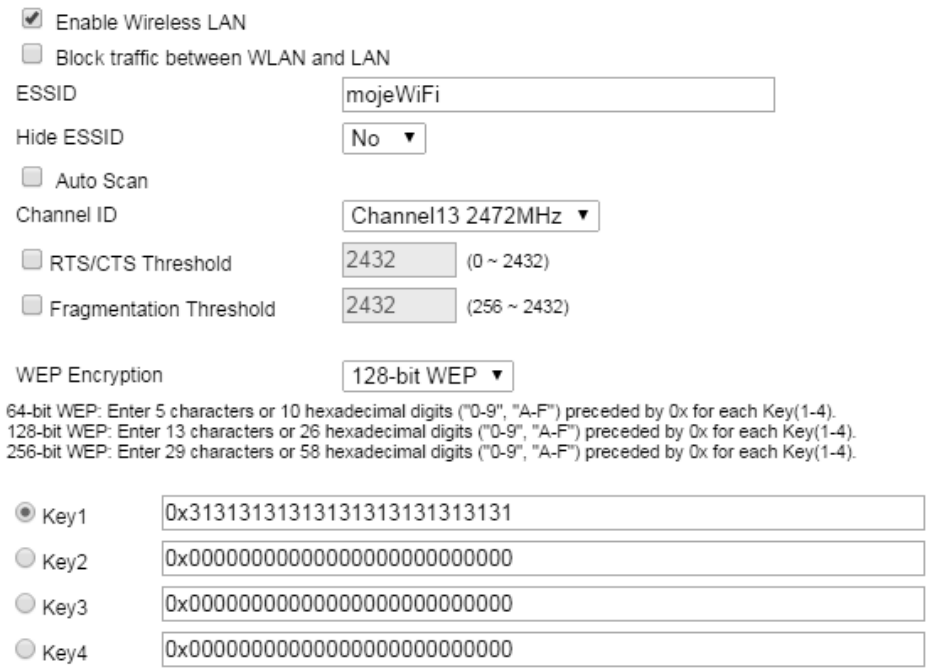
```
1 batchisp.exe -device AT32UC3A1256 -hardware usb -operation erase f memory
      flash blankcheck loadbuffer C:\Users\uzivatel\Downloads\wifiHD.elf
      program verify start reset 0
```

Tím je proces instalace aktuálního software u konce. Dva konektory, které jsme předtím spojili, teď rozpojíme a odpojíme shield od USB. Aktuální firmware by měl mít verzi 1.1.0 (nebo vyšší).

Kapitola 68
Údaje potřebné pro připojení k WiFi

Pro připojení k otevřené síti nechráněné heslem stačí vědět její SSID (název).

Pro sítě se zabezpečením WEP je potřeba znát SSID, klíč (anglicky *key*) a index klíče (anglicky *key index*). Klíč je heslo v hexadecimální podobě. Většinou se získává převedením řetězce do hexadecimální reprezentace nebo přímo zadáním tohoto řetězce (záleží na nastavení WiFi přístupového bodu).



Obrázek 68.1: Ukázka nastavení WEP klíče

U sítí WPA2 Personal nám stačí SSID a heslo.

Kapitola 69
Přehled funkcí pro práci s WiFi

V následujícím přehledu si ukážeme funkce a objekty, které jsou potřeba při práci s WiFi shieldem. Některé funkce knihovny WiFi jsou velmi podobné těm, se kterými jsme se setkali při práci s Ethernet shieldem a některé jsou dokonce stejné.

69.1 Třída `WiFi`

V následující tabulce zmíníme příkazy z této třídy.

Název	Zápis	Funkce

`WiFi.begin()`	`WiFi.begin(ssid);`
	`WiFi.begin(ssid, pass);`
	`WiFi.begin(ssid, keyIndex, key);`
		Tato funkce spustí komunikaci s WiFi přístupovým bodem. Může mít tyto parametry:
		`ssid` – název sítě,
		`pass` – heslo WPA2 sítě,
		`key` – klíč WEP sítě,
		`keyIndex` – WEP síť může mít až čtyři klíče pro připojení, tímto jí sdělujete, jaký z těchto čtyř hodláte použít.
		Funkce navíc vrací dvě různé hodnoty odpovídající konstantám:
		`WL_CONNECTED` – když se připojení k síti podařilo,
		`WL_IDLE_STATUS` – když se připojení nepodařilo, ale shield funguje.

`WiFi.disconnect()`	`WiFi.disconnect()`	Odpojí shield od sítě, ke které je právě připojen.
`WiFi.status()`	`WiFi.status();`	Zjistí stav shieldu a připojení k síti. Vrací hodnoty odpovídající konstantám: `WL_NO_SHIELD` – WiFi shield není připojen k Arduinu. `WL_IDLE_STATUS` – Shield je připojen, ale nepodařilo se připojit. `WL_NO_SSID_AVAIL` – SSID není dostupná. `WL_SCAN_COMPLETED` – Hledání sítí dokončeno. `WL_CONNECTED` – Připojeno k síti. `WL_CONNECT_FAILED` – Připojování selhalo. `WL_CONNECTION_LOST` – Ztráta spojení. `WL_DISCONNECTED` – Odpojeno.

`WiFi.config()`	`WiFi.config(ip);`
	`WiFi.config(ip, dns);`
	`WiFi.config(ip, dns, gateway);`
	`WiFi.config(ip, dns, gateway, subnet);`
		Umožňuje změnu IP adresy, adresu DNS, gateway a subnet. Doporučuji používat jen v případě, že víte, co děláte. Parametry funkce: `ip`, `dns`, `gateway` a `subnet`.

`WiFi.setDNS()`	`WiFi.setDNS(dns_server1);` `WiFi.setDNS(dns_server1, dns_server2);`	Umožňuje nastavení DNS serveru. Funkce má parametry: `dns_server1` – primární DNS server, `dns_server2` – sekundární DNS server. Opět doporučuji používat jen tehdy, když víte, co nastavujete.
`WiFi.scanNetworks()`	`WiFi.scanNetworks();`	Vrátí počet nalezených WiFi sítí. Tato funkce musí být volána, mají-li poté být bez problému použity funkce `.SSID()`.
`WiFi.SSID()`	`WiFi.SSID();` `WiFi.SSID (wifiAccessPoint);`	Při volání funkce bez parametru vrátí SSID sítě, ke které je shield aktuálně připojen. Parametr `wifiAccessPoint` je číslo sítě, které jí bylo přiděleno funkcí `.scanNetworks()`. V případě volání s parametrem vrátí SSID vybrané sítě.
`WiFi.BSSID()`	`WiFi.BSSID(bssid);`	Funkce vrátí MAC adresu přístupového bodu, ke kterému je shield připojen. Má jediný parametr: `bssid` – je pole, do kterého se uloží adresa, musí tedy být předem nadefinované a musí mít 6 prvků `(byte bssid[6];)`.

Název	Zápis	Funkce
`WiFi.RSSI()`	`WiFi.RSSI();` `WiFi.RSSI (wifiAccessPoint);`	Funkce funguje stejně jako `.SSID()`, pouze vrací sílu signálů sítě v dBmW (decibel-miliwatt).
`WiFi.encryptionType()`	`WiFi.encryptionType();` `WiFi.encryptionType (wifiAccessPoint);`	Stejné jako `.SSID()`, ale vrací typ zabezpečení.
`WiFi.macAddress()`	`WiFi.macAddress(mac);`	Vrátí MAC adresu shieldu. Parametr `mac` funguje stejně jako `bssid` u funkce `.BSSID()`.
`WiFi.getSocket()`	`WiFi.getSocket();`	Vrátí první přijatý socket.
`WiFi.localIP()`	`WiFi.localIP();`	Vrátí aktuální IP adresu shieldu. Vrácená IP je datového typu IPAdress.
`WiFi.subnetMask()`	`WiFi.subnetMask();`	Vrátí subnet WiFi sítě. Je datového typu IPAdress.
`WiFi.gatewayIP()`	`WiFi.gatewayIP();`	Vrátí gateway sítě. Vrácená data jsou také datového typu IPAdress.

69.2 Třída `WiFiServer`

Zde je souhrn příkazů.

Název	Zápis	Funkce
`Server()`	`WiFiServer server(port);`	Vytvoří WiFi server, který bude naslouchat událostem na zadaném portu.
`server.begin()`	`server.begin()`	Zahájí naslouchání serveru na nastaveném portu.
`server.available()`	`server.available();`	Vrátí informace o klientovi připojeném k vytvořenému serveru. Vrácená data jsou datového typu `WiFiClient`.
`server.write()`	`server.write(data);`	Pošle data všem klientům připojeným k serveru. Parametr `data` může být datového typu `byte` nebo `char`.
`server.print()`	`server.print(data);` `server.print(data, BASE);`	Pošle data všem připojeným klientům v podobě ASCII. `data` – informace k vypsání `BASE` – v jaké soustavě se mají vypsat čísla (`BIN`, `DEC`, `OCT`, `HEX`).
`server.println()`	`server.println(data);` `server.println(data, BASE);`	Stejné jako `.print()`, pouze na konec přidá zalomení řádku.

69.3 Třída `WiFiClient`

Obsahuje informace o klientovi a funkce pro jeho obsluhu.

Název	Zápis	Funkce
`WiFiClient()`	`WiFiClient client;`	Vytvoří proměnnou typu WiFiClient sloužící pro obsluhu a práci s klientem.
`client.connected()`	`client.connected();`	Funkce vrací `true`, pokud jsou k dispozici nějaká data odeslaná klientem. V opačném případě vrací `false`.

Název	Zápis	Funkce
`client.connect()`	`client.connect(ip,port);` `client.connect(URL,port);`	Připojí se k zadanému serveru. Ten může být zvolen pomocí `ip` adresy, nebo URL. Parametr `port` specifikuje port, přes který se k serveru připojujeme. Funkce vrací `true`/`false` podle úspěšnosti operace.
`client.write()`	`client.write(data);`	Pošle data serveru, ke kterému je připojen.
`client.print()`	`client.print(data);` `client.print(data,BASE);`	Pošle data serveru jako ASCII.
`client.println()`	`client.println(data);` `client.print(data,BASE);`	Pošle serveru data jako ASCII se zalomením řádku na konci.
`client.available()`	`client.available();`	Vrátí počet bytů, které jsou dostupné ke čtení ze serveru.
`client.read()`	`client.read();`	Vrátí následující přijatý byte. Pokud žádný není, vrátí `-1`.
`client.flush()`	`client.flush();`	Smaže všechny přijaté byty čekající na přečtení.
`client.stop()`	`client.stop();`	Odpojí klienta od serveru.

Kapitola 70
Příklady: WiFi shield

Tímto jsme si představili funkce používané pro obsluhu WiFi shieldu. Nyní si můžeme předvést několik ukázek. Ty vycházejí z oficiálních příkladů z dokumentace.

70.1 Připojení k síti

Následující příklad ukazuje, jak se připojit k síti se zabezpečením WEP. Jednoduchou úpravou parametrů u `WiFi.begin()` se ale dá příklad modifikovat i pro WEP2 síť a také síť bez zabezpečení.

```
1  #include <WiFi.h>
2
3  char ssid[] = "SSID_site";  //SSID site
4  char key[] = "klic_site";  //klic site
5  int keyIndex = 0;     //cislo klice
6  int status = WL_IDLE_STATUS;  //pomocna promenna uchovavajici stav pripojeni
7
8  void setup(){
9      Serial.begin(9600);
10
11     //zkontroluje, jestli je shield pripojen
12     if (WiFi.status() == WL_NO_SHIELD) {
13         Serial.println("WiFi shield neni pripojen");
14         while(true); //zacykli program, nic se nebude dit dal
15     }
16
17     //pripoji se k siti
18     while ( status != WL_CONNECTED) {
19         Serial.print("Pripojuji se k SSID: ");
20         Serial.println(ssid);
21         status = WiFi.begin(ssid, keyIndex, key);
22         delay(10000);
23     }
24
```

```
25      Serial.print("Jste pripojen");
26      printCurrentNet();
27      printWifiData();
28  }
29  void loop(){
30      //kazdych 10 sekund zkontrolujeme WiFi sit
31      delay(10000);
32      printCurrentNet();
33  }
34
35  void printWifiData() {
36      //funkce pro vypis IP a MAC shieldu
37      IPAddress ip = WiFi.localIP();
38      Serial.print("IP adresa: ");
39      Serial.println(ip);
40      Serial.println(ip);
41
42      byte mac[6];
43      WiFi.macAddress(mac);
44      Serial.print("MAC adresa: ");
45      Serial.print(mac[5],HEX);
46      Serial.print(":");
47      Serial.print(mac[4],HEX);
48      Serial.print(":");
49      Serial.print(mac[3],HEX);
50      Serial.print(":");
51      Serial.print(mac[2],HEX);
52      Serial.print(":");
53      Serial.print(mac[1],HEX);
54      Serial.print(":");
55     Serial.println(mac[0],HEX);
56  }
57
58  void printCurrentNet() {
59      //odesle SSID site
60      Serial.print("SSID: ");
61      Serial.println(WiFi.SSID());
62
63      //odesle MAC adresu pristupoveho bodu
64      byte bssid[6];
65      WiFi.BSSID(bssid);
66      Serial.print("BSSID: ");
67      Serial.print(bssid[5],HEX);
68      Serial.print(":");
69      Serial.print(bssid[4],HEX);
70      Serial.print(":");
71      Serial.print(bssid[3],HEX);
72      Serial.print(":");
73      Serial.print(bssid[2],HEX);
74      Serial.print(":");
75      Serial.print(bssid[1],HEX);
76      Serial.print(":");
77      Serial.println(bssid[0],HEX);
78
79      //odesle silu signalu
80      long rssi = WiFi.RSSI();
81      Serial.print("Sila signalu:");
82      Serial.println(rssi);
83
84      //odesle typ zabezpeceni
```

```
85      byte encryption = WiFi.encryptionType();
86      Serial.print("Typ zabezpeceni:");
87      Serial.println(encryption,HEX);
88      Serial.println();
89  }
```

70.2 Interakce se serverem

Pomocí dvou odkazů (zapni a vypni) budeme ovládat LED diodu připojenou na pin 9. Pomocí sériové linky nám Arduino vypíše, na jakou IP adresu se máme připojit. V příkladu se používá HTTP request. Práci s ním jsme si popsali na straně 161 u Ethernet shieldu.

```
1   #include <SPI.h>
2   #include <WiFi.h>
3
4   char ssid[] = "SSID_site";
5   char pass[] = "heslo_site";
6
7   int status = WL_IDLE_STATUS;
8   WiFiServer server(80);
9
10  void setup() {
11      Serial.begin(9600);
12      pinMode(9, OUTPUT);
13
14      if (WiFi.status() == WL_NO_SHIELD) {
15          Serial.println("Shield nepripojen");
16          while(true);
17      }
18
19      while ( status != WL_CONNECTED) {
20          Serial.print("Pripojuji k SSID: ");
21          Serial.println(ssid);
22
23          status = WiFi.begin(ssid, pass);
24          delay(10000);
25      }
26      server.begin();
27      printWifiStatus();
28  }
29  void loop() {
30      //ceka na pripojeni klienta
31      WiFiClient client = server.available();
32
33      if(client){ //kdyz naleznem klienta
34          String currentLine = "";
35          while (client.connected()){
36              if (client.available()){
37                  char c = client.read();
38                  Serial.write(c);
39                  //pokud najde znak zalomeni radku (ukoncuje request od klienta)
40                  if (c == 'n'){
41                      if (currentLine.length() == 0) {
42                          client.println("HTTP/1.1 200 OK");
43                          client.println("Content-type:text/html");
44                          client.println();
45                          //hlavicka je oddelena znakem noveho radku
46
47                          //zde zacina HTML kod stranky
48                          client.print("<a href=\"/H\">ZAPNI</a><br>");
```

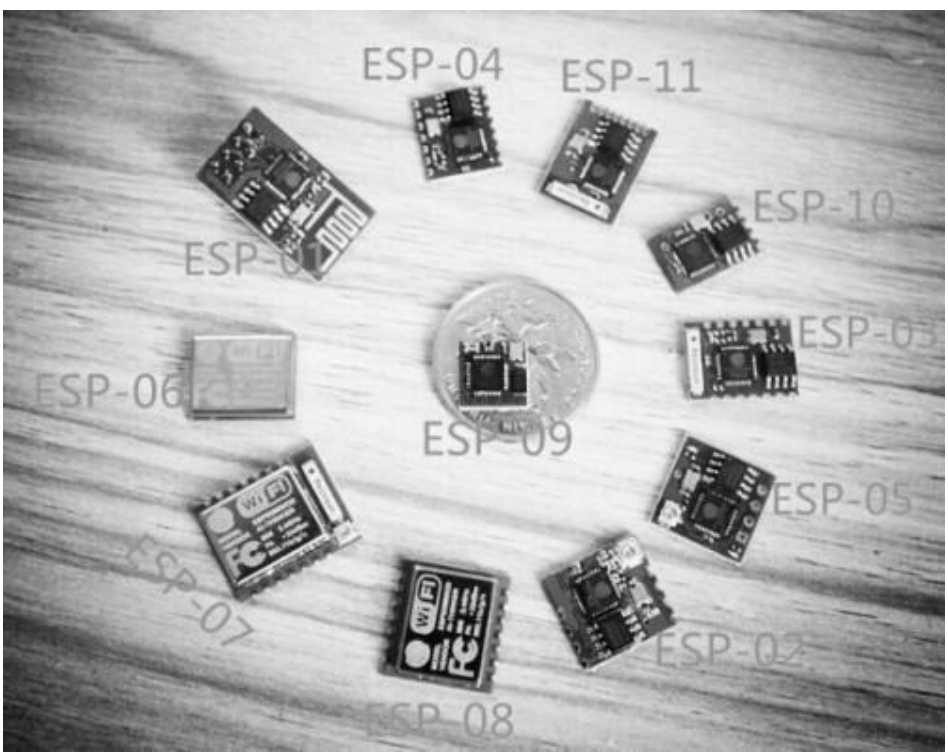
```
49                      client.print("<a href=\"/L\">VYPNI</a><br>");
50                      client.println(); //dalsim prazdnym radkem konci HTML
                            cast
51                      break;
52                  }
53                  else{
54                      currentLine = "";
55                  }
56              }
57          else if (c != 'r') {
58              currentLine += c;
59          }
60
61          //kontroluje, jestli request konci na H, nebo L
62          if (currentLine.endsWith("GET /H")) {
63              digitalWrite(9, HIGH); //zapne LED
64          }
65          if (currentLine.endsWith("GET /L")) {
66              digitalWrite(9, LOW); //vypne LED
67          }
68          }
69      }
70          client.stop();
71          Serial.println("client disonnected");
72      }
73  }
74
75  void printWifiStatus() {
76    Serial.print("SSID: ");
77    Serial.println(WiFi.SSID());
78
79    IPAddress ip = WiFi.localIP();
80    Serial.print("IP Address: ");
81    Serial.println(ip);
82
83    long rssi = WiFi.RSSI();
84    Serial.print("signal strength (RSSI):");
85    Serial.print(rssi);
86    Serial.println(" dBm");
87
88    Serial.print("http://");
89    Serial.println(ip);
90  }
```

Tyto základní příklady, společně se znalostmi zmíněnými v části o Ethernet shieldu, slouží jako odrazový můstek pro další tvorbu.

Část XX

ESP8266

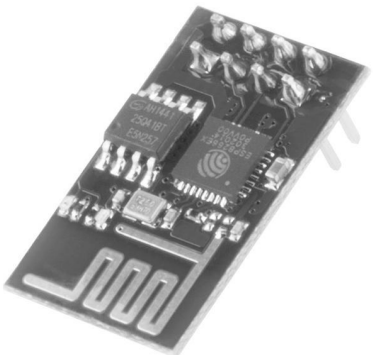
V roce 2014 se na trhu objevil malý WiFi modul, který vzbudil mezi bastlíři a hackery značný rozruch. Umožnil levně a poměrně jednoduše připojit prakticky jakékoliv zařízení s mikroprocesorem a pár volnými piny k WiFi síti. Doteď se jedná o jedno z nejlepších řešení, co se poměru cena/výkon týče.



Obrázek 71.1: Rodina čipů ESP8266 [36]

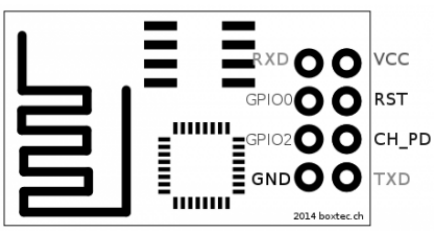
Kapitola 71
Co je to ESP8266

Můžeme se setkat s více verzemi modulů ESP8266 různých tvarů a velikostí, na všech však nalezneme čip ESP8266EX. Jednotlivé verze se od sebe liší například tím, jestli je elektronika na desce stíněná či nikoliv, nebo také počtem pinů. My použijeme asi nejrozšířenější verzi ESP8266-1. Ta nám, stejně jako ostatní modely, nabídne připojení WiFi 802.11 b/g/n, integrovaný TCP/IP stack a mnohé další.



Obrázek 71.2: ESP8266 verze 1 [21]

Pokud se na modul podíváme, nalezneme na něm celkem osm pinů. Čip budeme ovládat přes UART (sériovou linku) – té jsou vyhrazeny dva piny (RXD a TXD). Čip budeme ovládat pomocí AT příkazů, které jsou posílány právě po sériové lince. O tom ale později. Dále zde nalezneme dva napájecí piny, jeden pin pro restart čipu a jeden pro zapínání a vypínání čipu. Zbývající dva piny GPIO0 a GPIO2 v případě, kdy budeme ovládat modul Arduinem, asi moc nevyužijeme. Přicházejí na řadu, budeme-li chtít používat modul samostatně bez Arduina.



Obrázek 71.3: Piny ESP8266-1 [29]

Možná si říkáte, k čemu mi bude modul bez Arduina. Tím se dostáváme k dalšímu důvodu, proč je modul ESP8266 tak oblíbený. Čip ESP8266EX totiž nemusí sloužit jenom k tomu, aby hloupě vykonával příkazy, které mu Arduino (nebo jiné zařízení) pošle. Je totiž možné ho naprogramovat tak, aby sám plnil úlohu hlavní jednotky. V tomto případě využijeme právě piny GPIO0 a GPIO2, a to například k blikání LED diody nebo jako piny pro I2C sběrnici.

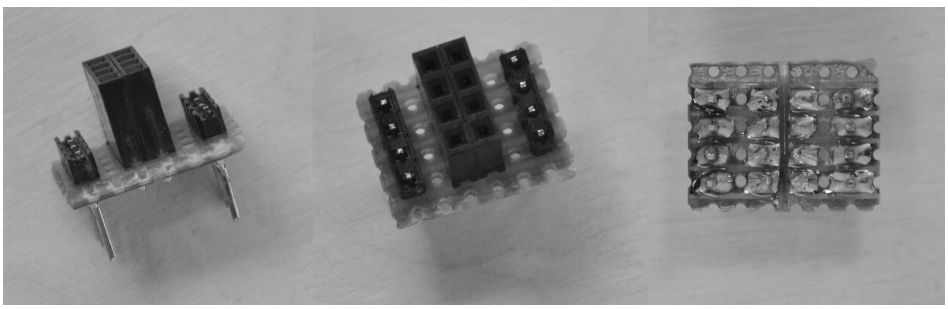
Kapitola 72
ESP8266 a Arduino

72.1 Zapojení

Při zapojování je potřeba si dávat největší pozor na to, že celý modul pracuje na 3,3 V, a ne na 5 V jako většina Arduin. Nestačí ale připojit VCC pin na 3.3V pin Arduina. I u sériové linky je zde totiž limit

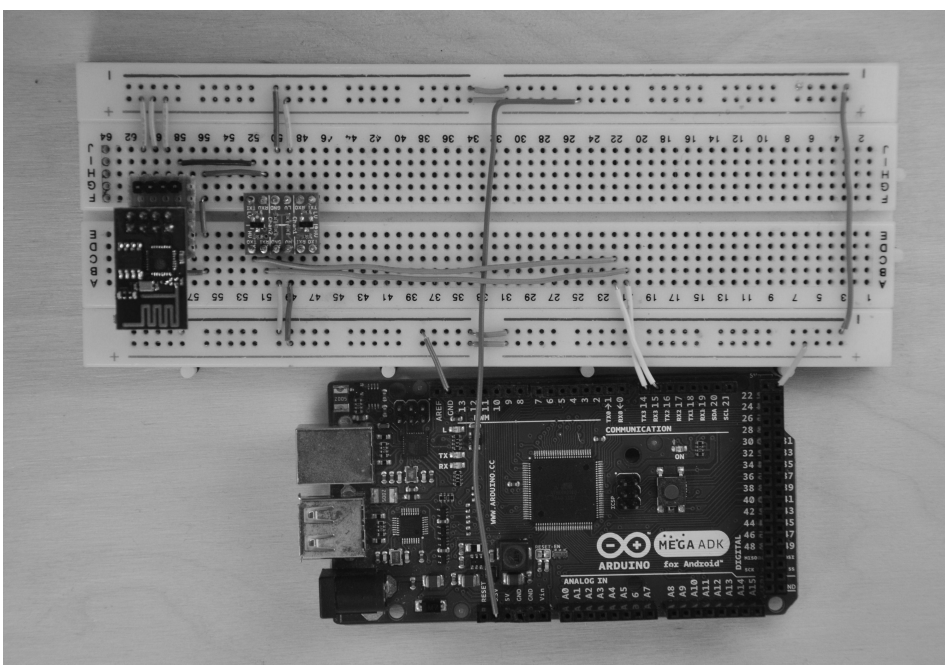
3,6 V, kdežto sériová linka Arduina pracuje na 5 V. Při použití většího napětí by mohlo dojít k poškození čipu. Proto je nutné zařídit, aby bylo napětí sériové linky Arduina sníženo (pokud máte Arduino pracující na 3,3 V, nemusíte převod úrovní napětí řešit). Asi nejbezpečnější je použít převodník úrovní – například tento[54]. Pokud ho ale nemáte po ruce, jde to i pomocí děliče napětí. My si ale ukážeme, jak připojit modul k Arduino přes zmíněný převodník.

Oříškem také může být, že modul nemá piny v jedné řadě, ale ve dvou blízko sebe. Proto doporučuji spájet si podobnou redukci, kterou vidíte na další straně na obrázku 72.1. Ta umožní jednoduché zasunutí modulu do nepájivého kontaktního pole.



Obrázek 72.1: Redukce pro ESP8266

Nyní vše musíme správně zapojit. Výsledek by mohl vypadat jako na obrázku č. 72.2.

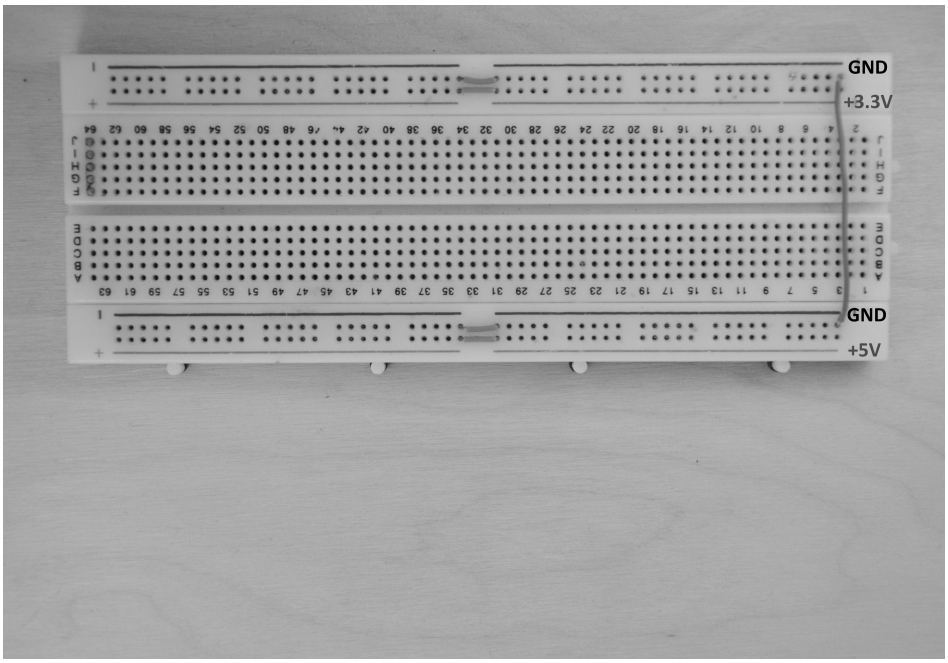


Obrázek 72.2: Zapojení ESP8266 a Arduina

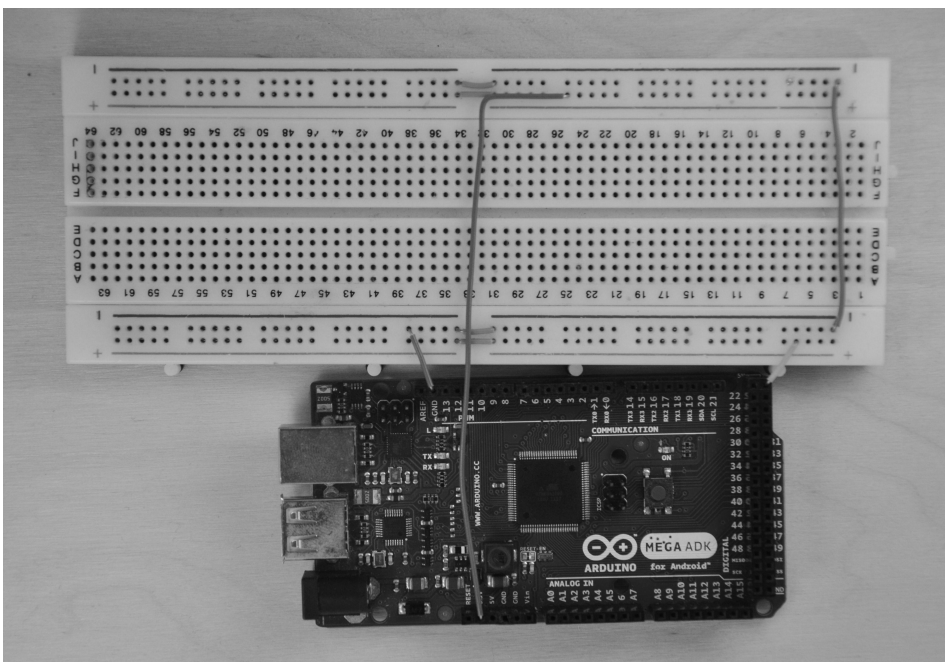
Vše ale postupně. Na kontaktním poli nalezneme po obou stranách dlouhé propojené linky. Na obrázku horní linky jsou v našem zapojení určené pro +3,3 V, spodní linky pro +5 V. Zem obou linek spojíme.

V dalším kroku připojíme ke kontaktnímu poli Arduino. +5V Arduina připojíme na spodní červenou linku, +3.3V na horní červenou linku. GND připojíme na libovolnou modrou linku (ty už máme propojené, takže připojením jedné připojíme i druhou).

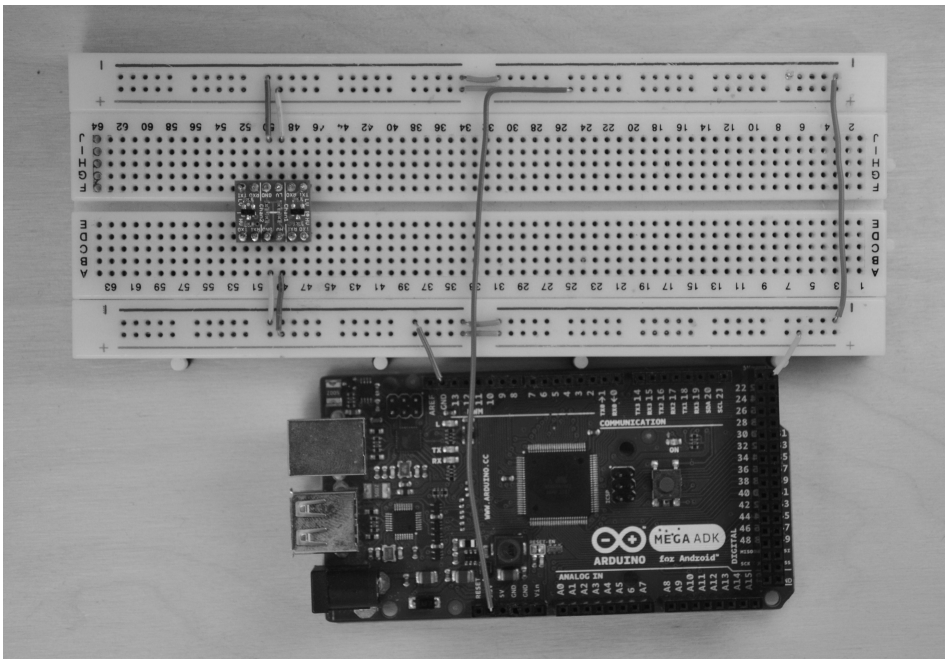
[54] Převodník úrovní – `http://www.hwkitchen.com/products/logic-level-converter/`



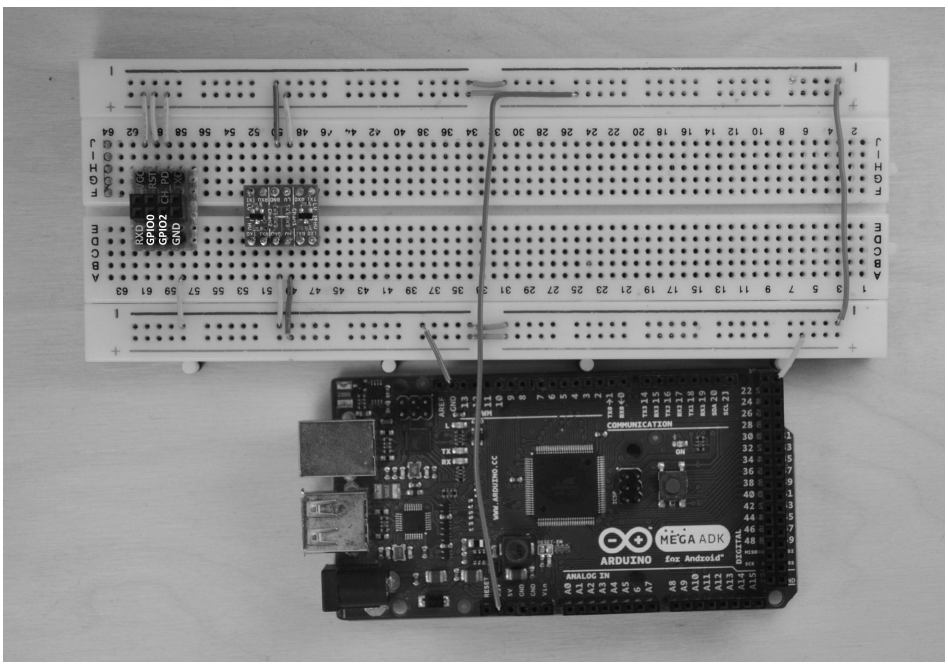
Obrázek 72.3: Zapojení ESP8266 – krok první



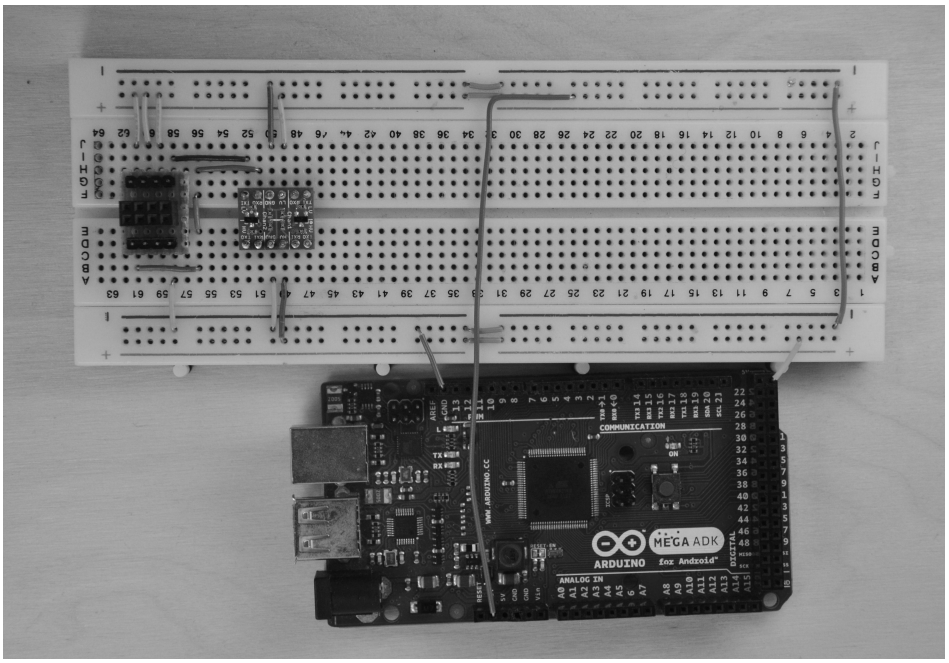
Obrázek 72.4: Zapojení ESP8266 – krok druhý



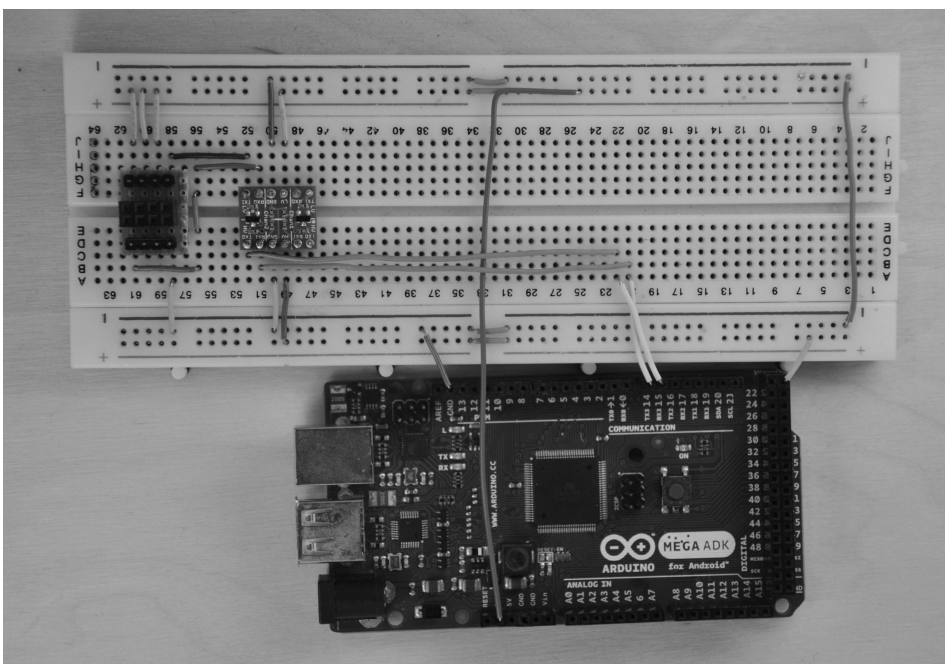
Obrázek 72.5: Zapojení ESP8266 – krok třetí



Obrázek 72.6: Zapojení ESP8266 – krok čtvrtý



Obrázek 72.7: Zapojení ESP8266 – krok pátý



Obrázek 72.8: Zapojení ESP8266 – krok šestý

Dále zapojíme převodník úrovní. Prostřední pin s popisem LV by měl směřovat nahoru (z pohledu obrázku). LV (low voltage) připojíme na +3.3V, HV (high voltage) na +5V. Obě dvě GND připojíme ke společné GND.

Nyní přichází na řadu námi vytvořená redukce pro připojení ESP modulu. Zasuneme jej do nepájivého pole a připojíme patřičné konektory k napájení. VCC, RST a CH_PD na +5V, GND na společnou GND.

Napájení modulu i převodníku máme zapojeno, nyní propojíme sériovou linku Arduina přes převodník se sériovou linkou modulu. Zde situace může být matoucí, proto je zapojení vysvětleno pomocí následující tabulky. Horní a dolní piny jsou brány podle obrázku.

Arduino	Převodník (horní piny)	Převodník (dolní piny)	ESP8266
RX3	TXO (output)	TXI (input)	TXD
TX3	RXI (input)	RXO (output)	RXD
Společná GND	GND	GND	Společná GND
Společná +5V	HV	LV	Společná +5V

Pozor na překřížené linky mezi Arduinem a převodníkem!

Nakonec zasuneme modul ESP8266 do připravené redukce – viz první obrázek zapojení.

72.2 Ovládání modulu

Jak už jsme zmínili dříve, ovládání modulu probíhá pomocí `AT` příkazů. Přehled dostupných příkazů naleznete zde[55], popřípadě i s vysvětlením zde[56]. My si je všechny představíme dále. Na začátek musíme ověřit funkčnost modulu. Jak je vidět ze zapojení, je modul připojen na TX3 a RX3, kterým odpovídá objekt `Serial3`. Ten ale najdeme jen u větších desek (Mega…). Například u Arduino Uno budeme další sériové porty hledat marně. V tomto případě připojíme modul na jedinou sériovou linku, kterou máme k dispozici (při programování je potom nutné modul odpojit).

Nejjednodušším příkladem `AT` příkazu je příkaz `AT` doprovázený znaky CR (anglicky *cariage return*, pozůstatek z doby psacích strojů a jehličkových tiskáren, symbolizuje návrat tiskací hlavy na začátek řádku) a LF (anglicky *line feed*, nový řádek). V programu ale nemůžeme napsat `CR`, popřípadě `LF`, protože by je program považoval za řetězec se dvěma znaky. Existuje vícero způsobů, jak tyto znaky napsat – jedním z nich jsou takzvané escape sekvence (CR = `\r`, LF = `\n`), druhým například přímé použití jejich číselné hodnoty (CR = 13, LF = 10). Další možností je použít funkci `Serial.println("...")`, která znaky CR a LF odešle za nás. Ne vždy se nám ale toto chování hodí.

```
1  //ekvivalentni jsou tedy zapisy:
2  Serial.print("AT\r\n");
3
4  Serial.print("AT");
5  Serial.print("\r");
6  Serial.print("\n");
7
8  Serial.println("AT");
9
10 Serial.print("AT");
11 Serial.write(13);
12 Serial.write(10);
```

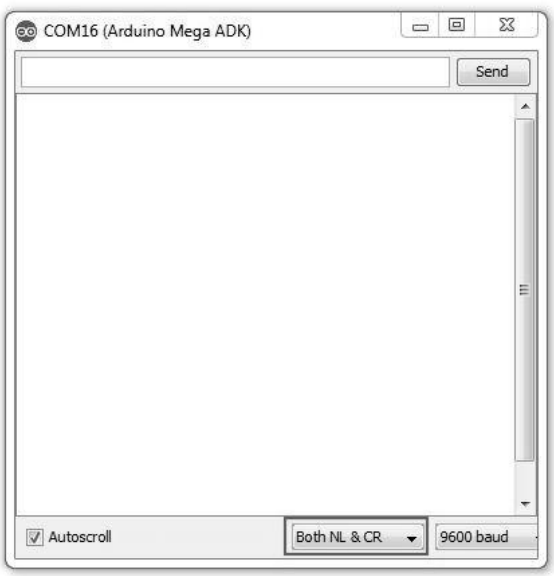
Pro účely zkoušení si napíšeme program, který bude pouze přeposílat znaky mezi `Serial` a `Serial3` (tedy mezi PC a modulem). Poté budeme pomocí monitoru sériové linky posílat modulu `AT` příkazy, abychom si je vyzkoušeli.

```
1  void setup(){
2      Serial.begin(9600);
3      Serial3.begin(9600); //9600, 57600, 115200
```

[55] `AT` příkazy – `https://cdn.sparkfun.com/datasheets/Wireless/WiFi/Command Doc.pdf`
[56] Vysvětlené `AT` příkazy – `https://room-15.github.io/blog/2015/03/26/esp8266-at-command-reference/`

```
4  }
5
6  void loop(){
7      while(Serial.available()){
8          Serial3.write(Serial.read());
9      }
10
11     while(Serial3.available()){
12         Serial.write(Serial3.read());
13     }
14 }
```

Všimněte si, že u `Serial3.begin()` je v komentáři uvedeno `//9600, 57600, 115200`. To je z toho důvodu, že v závislosti na verzi používaného firmware komunikují moduly na různých rychlostech. Pokud nevíte, jakou verzi firmware váš modul zrovna má, vyzkoušejte všechny tři a uvidíte, kdy se modul ozve. Než zkusíme první příkaz, musíme sériovému monitoru říci, že chceme za každým odeslaným řetězcem odeslat i znaky CR a LF. Toto je možné nastavit hned vedle výběru rychlosti komunikace.



Obrázek 72.9: Nastavení monitoru sériové linky

Nyní už máme vše nastavené. Do textového pole monitoru zadáme příkaz `AT`. Modul by měl odpovědět:

```
1 AT
2
3 OK
```

Tím máme jistotu, že modul funguje. To nám ale určitě stačit nebude. Budeme po modulu například chtít, aby se připojil k serveru a něco z něj stáhl. Předtím si ale musíme vysvětlit, na co jsou které `AT` příkazy.

72.3 `AT` příkazy pro ESP8266

Příkazy můžeme rozdělit do tří kategorií. První kategorii bychom mohli nazvat základní. Nalezneme v ní ty příkazy, které se starají o samotný běh modulu. Další dvě kategorie můžeme nazvat WiFi příkazy a TCPIP příkazy.

Druhým způsobem, jak příkazy rozdělit, je podle jejich funkčnosti a také způsobu zápisu.

Typ	Příklad	Popis
Dokumentační	`AT+PRIKAZ=?`	Vrátí rozsah hodnot, které je možné předat v parametru příkazu.
Informační	`AT+PRIKAZ?`	Vrátí aktuálně nastavenou hodnotu.
Nastavovací	`AT+PRIKAZ=parametr`	Nastaví hodnotu podle zadaného parametru.
Vykonávací	`AT+PRIKAZ`	Provede zadaný příkaz.

72.3.1 Základní příkazy

Zde je seznam základních příkazů.

Příkaz	Odpověď	Popis
`AT`	`OK`	Jednoduchý příkaz, který slouží pouze k tomu, abychom zjistili, jestli modul funguje.
`AT+RST`	`OK`	Restartuje modul a vrátí informace (liší se podle verze firmware).
`AT+GMR`	verze	Vrátí aktuální verzi firmware.
`AT+GSLP=cas`	čas	Uvede modul do stavu spánku na zadaný čas (v milisekundách).
`ATE0`	`OK`	Zakáže opakování zadaného příkazu modulem.
`ATE1`	`OK`	Povolí opakování zadaného příkazu modulem.

72.3.2 WiFi příkazy

Zde je zmíněn seznam WiFi příkazů.

Příkaz	Odpověď	Popis
`AT+CWMODE=?`	`+CWMODE:(1-3)`	Vrátí rozsah možných hodnot.
`AT+CWMODE?`	`+CWMODE: mod`	Vrátí číslo aktuálního módu.
`AT+CWMODE=mod`	`OK`	Nastaví mód. 1 – klient 2 – přístupový bod 3 – klient + přístupový bod
`AT+CWJAP?`	`+CWJAP: ssid`	Vypíše SSID bodu, ke kterému je modul aktuálně připojen.
`AT+CWJAP=ssid,heslo`	`OK`	Připojí se k WiFi se zadaným jménem a heslem.
`AT+CWLAP`	`+CWLAP:(zab,ssid,sila,mac)`	Vrátí seznam viditelných WiFi sítí. `Zab` udává typ zabezpečení: 0 = otevřená síť 1 = WEP 2 = WPA_PSK 3 = WPA2_PSK 4 = WPA_WPA2_PSK `ssid` značí jméno WiFi, `sila` sílu signálu a `mac` adresu přístupového bodu.
`AT+CWLAP=ssid, mac,kan`	`+CWLAP:(zab,ssid,sila,mac)`	Zjistí, jestli existuje přístupový bod se zadanou `ssid`, `mac` a kanálem.
`AT+CWQAP`	`OK`	Odpojí se od aktuálně připojeného přístupového bodu.

Příkaz	Odpověď	Popis
`AT+CWSAP=ssid, heslo,kan,zab`	`OK`	Vytvoří přístupový bod SoftAP[57]), ke kterému je možné se připojit pomocí zadaných údajů. Zabezpečení se řídí stejnými pravidly jako u `AT+CWLAP`.
`AT+CWSAP?`	`+CWSAP:ssid,heslo,kan,zab`	Vrátí aktuální nastavení přístupového bodu.
`AT+CWLIF`	`ip,mac`	Vrátí informace o připojených zařízeních.

72.3.3 TCPIP příkazy

ESP8266 podporuje i vícenásobná připojení. Aktuální mód připojení se nastavuje pomocí `AT+CIPMUX`. Od jeho nastavení se odvíjí i použití některých příkazů z této skupiny.

Příkaz	Odpověď	Popis
`AT+CIPSTATUS`	`Status:cislo`	Vrátí aktuální status modulu: 2 = má přidělenou IP, 3 = připojen, 4 = nepřipojen.
`AT+CIPSTART= typ,adr,port`	`OK`	Pro jednoduché připojení. Připojí se jako klient na zadanou adresu a port. Typ připojení může být TCP, nebo UDP.
`AT+CIPSTART= id,typ,adr,port`	`OK`	Pro vícenásobné připojení. Připojí se jako klient. Stejné jako předchozí, jen musíme specifikovat id připojení (může být mezi 0 a 4).
`AT+CIPSTART=?`	`+CIPSTART:(id)("type"), ("ip address"),(port)` `+CIPSTART:((id)"type"), ("domain name"),(port)`	Vrátí možné parametry příkazu a jejich zápis.
`AT+CIPSEND= delka`	`SEND OK`	Jednoduché připojení. Oznámí, že budou odeslána data. Délka značí, kolik znaků bude odesláno (maximálně 2048 znaků najednou).
`AT+CIPSEND= id,delka`	`SEND OK`	Vícenásobné připojení. `id` značí id požadovaného připojení.
`AT+CIPCLOSE`	`OK`	Jednoduché připojení. Ukončí aktuální připojení.
`AT+ CIPCLOSE=id`	`OK`	Vícenásobné připojení. Ukončí aktuální připojení s daným id.
`AT+CIFSR`	`+CIFSR:ip`	Vrátí lokální IP adresu ESP8266 modulu.
`AT+CIPMUX?`	`+CIPMUX:mod`	Vrátí aktuální mód připojení: 0 = jednoduché připojení, 1 = vícenásobné připojení.
`AT+ CIPMUX=mode`	`OK`	Nastaví mód připojení.
`AT+CIPSERVER= mod[,port]`	`OK`	Upraví server. `mod` může nabývat hodnot: 0 = smaže aktuální server (musí následovat restart), 1 = vytvoří server. Pokud parametr `port` nezadáme, je nastavena výchozí hodnota 333. Server může být vytvořen pouze pokud `AT+CIPMUX=1`.
`AT+CIPMODE?`	`+CIPMODE:mod`	Vrátí aktuální mód přenosu dat: 0 = normální mód, 1 = rozšířený (anglicky *unvarnished*) mód.

[57] Wikipedia, SoftAP – `https://en.wikipedia.org/wiki/SoftAP`

Příkaz	Odpověď	Popis
`AT+ CIPMODE=mod`	`OK`	Nastaví mód přenosu dat.
`AT+CIPSTO?`	`+CIPSTO:time`	Vrátí timeout serveru v sekundách.
`AT+CIPSTO=cas`	`OK`	Nastaví timeout serveru (0 – 7200 sekund).
`AT+CIUPDATE`	`+CIPUPDATE:n`	Spustí update firmware přes internet. Nedělejte, pokud přesně nevíte, co děláte. Může dojít až k nefunkčnosti modulu. Parametr `n` nabývá hodnot: 1 = server nalezen, 2 = připojeno k serveru, 3 = nový firmware nalezen, 4 = aktualizace firmware.
	`+IPD,delka:data`	Jednoduché připojení. Není tak úplně příkaz. Označuje data, který do modulu přišla zvnější.
	`+IPD,id,delka:data`	Vícenásobné připojení. Zde máme navíc ID připojení.

Pokud jsou parametry řetězce, musíme je dát do uvozovek. `AT` příkazy se mohou mírně lišit mezi jednotlivými verzemi firmware.

Snad jsme vás neodradili dlouhým výčtem příkazů. Rozhodně není potřeba je umět nazpaměť, jen je dobré vědět, že něco takového existuje a kde najít další informace. Teď už ale k prvnímu příkladu.

72.4 Jednoduchá interakce se serverem

Připojíme se k serveru `www.google.com` a stáhneme pomocí HTTP requestu jeho obsah jako HTML. Jelikož by se nám některé příkazy často opakovaly, jsou umístěné ve funkcích. Jsou to funkce: `resendData()`, která jen přeposílá data mezi `Serial` a `Serial3`. Pokud si nepamatujete, jak funguje HTTP, podívejte se do čísti o Ethernet shieldu.

```
1 boolean resendData(){
2     while(Serial.available()){
3         Serial3.write(Serial.read());
4     }
5
6     while(Serial3.available()){
7         Serial.write(Serial3.read());
8     }
9 }
```

Dále funkce `waitFor()`, která čeká na zadaný řetězec. Pokud je nalezen, vrátí `true`, pokud je po určitou dobu nenalezen, vrátí `false`. Všimněte si funkce `Serial3.setFind()`, která čte buffer sériové linky, dokud nenarazí na zadaný řetězec, nebo nevyprší čas zadaný funkcí `Serial3.setTimeout()`.

```
1 boolean waitFor(char *s, long waitingTime){
2     Serial3.setTimeout(waitingTime);
3     return Serial3.find(s);
4 }
```

A poslední vytvořenou funkcí je `command()`. Ta odešle zadaný `AT` příkaz, vypíše přes sériovou linku, co se děje, a čeká na výsledek. Pokud se příkaz povedl, vrátí `true`. Jinak vrátí `false`.

```
1 boolean command(String cmd, String comment, String success, String
      unsuccess, char *waitForString, long waitingTime){
2     Serial.println();
3     Serial.println(comment);
4     Serial.println("] " + cmd);
5     Serial3.println(cmd);
6     boolean result = waitFor(waitForString, waitingTime);
7     if(result){
```

```
8         Serial.println("[ " + success);
9     }
10    else{
11        Serial.println("[ " + unsuccess);
12    }
13    resendData();
14
15    return result;
16 }
```

Celý kód, který stáhne zdrojový kód úvodní stránky `www.google.com`, tedy vypadá takto. Jelikož musíme občas poslat po sériové lince znak uvozovek, použijeme escape sekvenci pro uvozovky: `\"`.

```
1  char ssid[] = "ssid-site"; //SSID wifi site
2  char pwd[] = "heslo-site"; //heslo site
3  char ip[] = "74.125.141.103"; //IP adresa www.google.com
4
5  String cmd; //retezec pro ukladani a skladani prikazu
6  String http;
7
8  void setup(){
9      Serial.begin(9600);
10     Serial3.begin(9600); //9600, 57600, 115200
11
12     cmd = "AT";
13     command(cmd, "Zkousim pritomnost modulu", "Modul komunikuje", "Modul
           nekomunikuje", "OK", 3000);
14
15     cmd = "AT+RST";
16     command(cmd, "Restartuji modul", "Modul restartovan", "Modul se
           nepovedlo restartovat", "ready", 5000);
17
18     cmd = "AT+CWMODE=1";
19     command(cmd, "Nastavuji klient mod", "Nastaveno", "Nepovedlo se
           nastavit", "OK", 2000);
20
21     cmd = "AT+CWJAP=\"";
22     cmd += ssid;
23     cmd += "\",\"";
24     cmd += pwd;
25     cmd += "\"";
26     command(cmd, "Pripojuji k WiFi", "Pripojeno", "Nelze se pripojit", "OK",
           10000);
27
28     cmd = "AT+CIPMUX=0";
29     command(cmd, "Nastavuji jednoduche pripojeni", "Nastaveno", "Nepovedlo
           se nastavit", "OK", 1000);
30
31     cmd = "AT+CIPSTART=\"TCP\",\"";
32     cmd += ip;
33     cmd += "\",80";
34     command(cmd, "Pripojuji k zadane IP", "Pripojeno", "Nepovedlo se
           pripojit", "OK", 5000);
35     waitFor("Linked", 1000);
36
37     http = "GET / HTTP/1.1\r\n\r\n";
38     cmd = "AT+CIPSEND=";
39     cmd += http.length();
40     if(command(cmd, "Posilam HTTP request", "Odeslano", "Cas spojeni
           vyprsel", ">", 5000)){
41         Serial3.print(http);
42         Serial.println(">");
```

```
43      }
44      else{
45          cmd = "AT+CIPCLOSE";
46          command(cmd, "Ukoncuji", "Ukonceno", "", "OK", 2000);
47      }
48  }
49
50  void loop(){
51      resendData();
52  }
53
54  boolean command(String cmd, String comment, String success, String
        unsuccess, char *waitForString, long waitingTime){
55      Serial.println();
56      Serial.println(comment);
57      Serial.println("] " + cmd);
58      Serial3.println(cmd);
59      boolean result = waitFor(waitForString, waitingTime);
60      if(result){
61          Serial.println("[ " + success);
62      }
63      else{
64          Serial.println("[ " + unsuccess);
65      }
66      resendData();
67
68      return result;
69  }
70
71  boolean waitFor(char *s, long waitingTime){
72      Serial3.setTimeout(waitingTime);
73      return Serial3.find(s);
74  }
75
76  boolean resendData(){
77      while(Serial.available()){
78          Serial3.write(Serial.read());
79      }
80
81      while(Serial3.available()){
82          Serial.write(Serial3.read());
83      }
84  }
```

Vrácená data můžeme načítat a dále s nimi pracovat. Pokud nějakému příkazu nerozumíte, nejjednodušší způsob, jak vyzkoušet jeho funkčnost, je ho odeslat přes monitor sériové linky. Co když chceme, aby na modulu běžel jednoduchý server?

72.5 Vytváříme server

K vytvoření serveru použijeme funkce, které jsme si ukázali v předchozím příkladu. Vytvoříme si server, který bude čekat na HTTP request `GET /promenna HTTP/1.1`. Jeho úkolem bude načíst číslo zapsané za lomítkem na místě promenna. Na server odešleme tento request tak, že se připojíme na vytvořenou WiFi a do prohlížeče zadáme `ip:port/promenna` – tedy například `192.164.4.1:333/5`.

Už jsme si řekli, že když modulu přijdou nějaká data, vrátí nám je ve tvaru: `+IPD,id,delka:data`. Data tvoří právě náš HTTP – například `GET /5 HTTP/1.1`. Celá přijatá informace tedy bude vypadat takto: `+IPD,id,delka:GET /promenna HTTP/1.1`. Nám stačí přečíst třetí číslo po řetězci `IPD`, čímž získáme naši proměnnou.

```
1  char ssid[] = "ssid-site"; //SSID WiFi site
2  char pwd[] = "heslo-site"; //heslo site
```

```
3
4  String cmd; //retezec pro ukladani a skladani prikazu
5  String http;
6
7  void setup(){
8      Serial.begin(9600);
9      Serial3.begin(9600); //9600, 57600, 115200
10
11     cmd = "AT";
12     command(cmd, "Zkousim pritomnost modulu", "Modul komunikuje", "Modul
           nekomunikuje", "OK", 3000);
13
14     cmd = "AT+RST";
15     command(cmd, "Restartuji modul", "Modul restartovan", "Modul se
           nepovedlo restartovat", "ready", 5000);
16
17     cmd = "AT+CIPMUX=1";
18     command(cmd, "Nastavuji vicenasobne pripojeni", "Nastaveno", "Nepovedlo
           se nastavit", "OK", 1000);
19
20     cmd = "AT+CWMODE=2";
21     command(cmd, "Nastavuji AP mod", "Nastaveno", "Nepovedlo se nastavit",
           "OK", 2000);
22
23     cmd = "AT+CWSAP=\"ESP-WIFI\",\"12345678\",11,2";
24     command(cmd, "Vytvarim pristupovy bod", "Vytvoreno", "Nepovedlo se
           vytvorit pristupovy bod", "OK", 3000);
25
26     cmd = "AT+CIPSERVER=1";
27     command(cmd, "Vytvarim server", "Vytvoreno", "Nepovedlo se vytvorit
           server", "OK", 2000);
28
29     cmd = "AT+CIFSR";
30     command(cmd, "Zjistuji aktualni adresu", "", "", "", 2000);
31 }
32
33 void loop(){
34     if(waitFor("+IPD", 50)){ //hledej vyskyt +IPD
35         Serial3.parseInt(); //jedno cislo zahod
36         Serial3.parseInt(); //druhe cislo zahod
37         int p = Serial3.parseInt(); //precti treti cislo
38
39         Serial.print("Nacetl jsem: ");
40         Serial.println(p);
41
42         Serial3.flush(); //zbytek zpravy me ted nezajima, zahod ho
43     }
44
45     resendData();
46 }
47
48 boolean command(String cmd, String comment, String success, String
       unsuccess, char *waitForString, long waitingTime){
49     Serial.println();
50     Serial.println(comment);
51     Serial.println("] " + cmd);
52     Serial3.println(cmd);
53     boolean result = waitFor(waitForString, waitingTime);
54     if(result){
55         Serial.println("[ " + success);
```

```
56      }
57      else{
58          Serial.println("[ " + unsuccess);
59      }
60      resendData();
61
62      return result;
63  }
64
65  boolean waitFor(char *s, long waitingTime){
66      Serial3.setTimeout(waitingTime);
67      return Serial3.find(s);
68  }
69
70  boolean resendData(){
71      while(Serial.available()){
72          Serial3.write(Serial.read());
73      }
74
75      if(Serial3.available()){
76          while(Serial3.available()){
77              Serial.write(Serial3.read());
78          }
79      }
80  }
```

Po připojení k WiFi a načtení adresy modulu by se měla zobrazit zpráva: `Nacetl jsem: x`.

Odeslaná data nemusejí být nutně HTTP requesty. Může se jednat například o zprávy protokolu MQTT, který se v zařízeních připojených do internetu věcí používají častěji. MQTT je ale nad rámec této kapitoly. Postupně se však dostáváme k provozu ESP8266 bez Arduina. Začneme pozvolna.

Kapitola 73
ESP8266 Thing

V překladu ESP8266 „Věc“[58] od společnosti Sparkfun je zajímavý kousek hardware. Možná si říkáte, co značí slovíčko Thing, neboli „Věc“ v názvu. Filozofie tohoto pojmenování je jednoduchá – tato „Věc“ má být jednou jednotkou velkého Internetu věcí[59]. Na samotné desce nalezneme prakticky celý modul ESP, který jsme si již představili, ale navíc také konektor pro napájení přes USB se stabilizátorem na 3,3 V, konektory pro připojení baterie a antény a mnohé další maličkosti, které nám usnadní práci. Začneme zapojením.

Obrázek 73.1: ESP8266 Thing [22]

[58] ESP8266 Thing – `http://www.hwkitchen.com/products/sparkfun-esp8266-thing/`
[59] Internet věcí – `https://cs.wikipedia.org/wiki/Internet_věcí`

73.1 Zapojení ESP8266 Thing

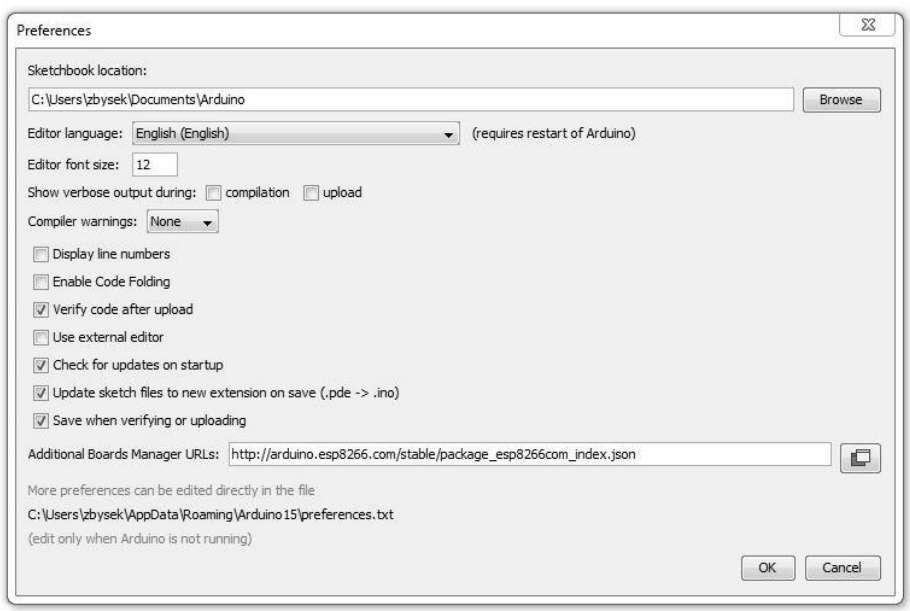
Ačkoliv by se mohlo zdát, že je možné tento modul programovat přes USB, bohužel tomu tak není. USB slouží pouze k napájení. Pro programování tedy budeme muset použít USB-Serial převodník (například tento[60]). Použít můžeme praktický jakýkoliv, jen musíme zajistit, aby tento převodník pracoval na 3,3 V. Propojení je pak následující.

USB-Serial převodník	Sparkfun IoT Thing
RX	TXO
TX	RXI
DTR	DTR
GND	GND
3.3V	VIN

73.2 Nastavení IDE

Dále připojíme pin 0 na GND, čímž řekneme, že budeme programovat. Na ukázku použijeme příklady z dokumentace[61] modulu. Nejdříve si však nastavíme Arduino IDE. To musí být ve verzi 1.6.5! Otevřeme si *File* → *Preferences* a do volného pole *Aditional Boards Manager URLs* přidáme:

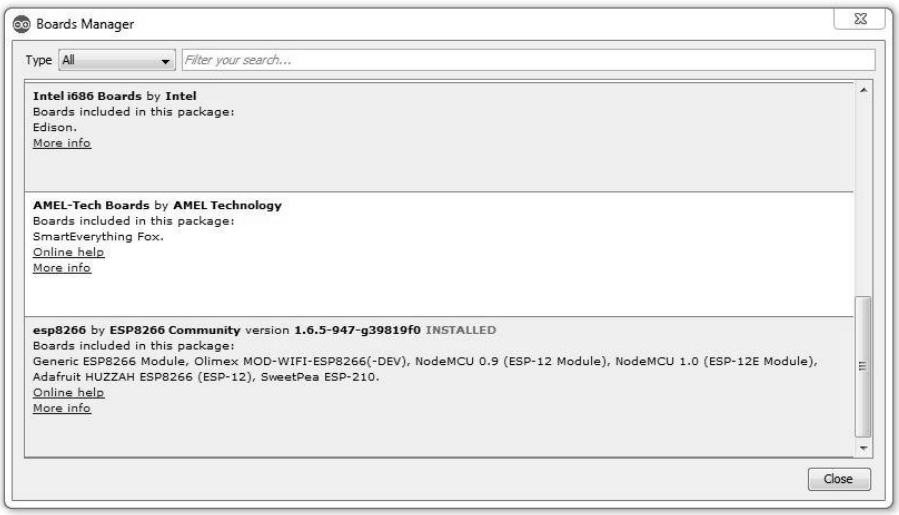
```
1 http://arduino.esp8266.com/stable/package_esp8266com_index.json
```



Obrázek 73.2: Nastavení desek v Arduino IDE

[60] USB-Serial modul – `http://www.hwkitchen.com/products/breakout-board-for-ft232rl-usb-to-seria/`
[61] Dokumentace ESP8266 Thing – `https://learn.sparkfun.com/tutorials/esp8266-thing-hookup-guide/all`

Potvrdíme tlačítkem OK a otevřeme nabídku *Tools* → *Board*. V horní části nalezneme položku *Boards Manager*. Když na ni klikneme, otevře se nám nabídka desek, které je možné přidat. Nalezneme položku *esp8266*, klikneme na ni a zvolíme možnost *Install*. Poté se stáhnou požadované nástroje.



Obrázek 73.3: Přidání ESP desek v Arduino IDE

73.3 Hello World!

V nabídce *Board* se nám nyní objevily nové desky. My vybereme *Sparkfun ESP8266 Thing*. Tím je nastavování hotové a my můžeme nahrát první program. Jedná se o ekvivalent k příkladu `Blink`, jenom náš modul nemá pevně napájenou LED na pinu 13, ale 5.

```
1  #define ESP8266_LED 5
2
3  void setup(){
4    pinMode(ESP8266_LED, OUTPUT);
5  }
6
7  void loop(){
8    digitalWrite(ESP8266_LED, HIGH);
9    delay(10);
10   digitalWrite(ESP8266_LED, LOW);
11   delay(50);
12 }
```

Zvolíme port, na kterém máme připojený USB-Serial převodník a zmáčkneme tlačítko *Upload*.

LED by měla nyní začít blikat.

73.4 Komunikace se serverem

V oficiální dokumentaci dále nalezneme zajímavý příklad, který využívá online datové uložiště *Phant*. V první řadě musíme stáhnout knihovnu, která umí s uložištěm komunikovat. V *Sketch* → *Include Library* → *Manage libraries* vyhledáme knihovnu *Phant* a stáhneme ji. Poté do modulu nahrajeme následující program. Nové věci jsou okomentované v kódu. LED na pinu 5 bliká, dokud se nepovede připojení k WiFi. Poté začne odesílat data na server Phant. Odesílá se stav digitálního pinu dvanáct a jediného analogového pinu ADC. Pozor! Na tento pin můžeme připojit maximálně 1 V!

Obrázek 73.4: Upload programu do ESP8266 Thing

```
1  #include <ESP8266WiFi.h> //knihovna pro praci s WiFi
2  #include <Phant.h> //knihovna Phant
3
4  const char WiFiSSID[] = "nazev-wifi";
5  const char WiFiPSK[] = "heslo-wifi";
6
7  const int LED_PIN = 5;
8  const int ANALOG_PIN = A0; //jediny analogovy pin
9  const int DIGITAL_PIN = 12; //digitalni pin s tlacitkem
10
11 //nastaveni pripojeni k Phant serveru
12 const char PhantHost[] = "data.sparkfun.com";
13 const char PublicKey[] = "wpvZ9pE1qbFJAjaGd3bn";
14 const char PrivateKey[] = "wzeB1zOxWNt1YJX27xdg";
15
16 const unsigned long postRate = 30000;
17 unsigned long lastPost = 0;
18
19 void setup(){
20     initHardware();
21     connectWiFi();
22     digitalWrite(LED_PIN, HIGH);
23 }
24
25 void loop(){
26     if (lastPost + postRate <= millis()) {
27         if (postToPhant()){
28             lastPost = millis();
29         }
```

```
30          else{
31              delay(100);
32          }
33      }
34  }
35
36  void connectWiFi(){
37      byte ledStatus = LOW;
38
39      WiFi.mode(WIFI_STA); //nastaveni WiFi
40      WiFi.begin(WiFiSSID, WiFiPSK); //pripojeni k WiFi
41
42      while (WiFi.status() != WL_CONNECTED){ //dokud se nepovede pripojit
43          digitalWrite(LED_PIN, ledStatus); // Write LED high/low
44          ledStatus = (ledStatus == HIGH) ? LOW : HIGH;
45          delay(100);
46      }
47  }
48
49  void initHardware(){
50      Serial.begin(9600);
51      pinMode(DIGITAL_PIN, INPUT_PULLUP);
52      pinMode(LED_PIN, OUTPUT);
53      digitalWrite(LED_PIN, LOW);
54  }
55
56  int postToPhant(){
57      digitalWrite(LED_PIN, HIGH);
58
59      Phant phant(PhantHost, PublicKey, PrivateKey); //vytvoreni objektu phant
60
61      uint8_t mac[WL_MAC_ADDR_LENGTH];
62      WiFi.macAddress(mac);
63      String macID = String(mac[WL_MAC_ADDR_LENGTH - 2], HEX) +
            String(mac[WL_MAC_ADDR_LENGTH - 1], HEX);
64
65      macID.toUpperCase();
66      String postedID = "Thing-" + macID;
67
68      phant.add("id", postedID);
69      phant.add("analog", analogRead(ANALOG_PIN));
70      phant.add("digital", digitalRead(DIGITAL_PIN));
71      phant.add("time", millis());
72
73      WiFiClient client;
74      const int httpPort = 80;
75      if (!client.connect(PhantHost, httpPort)){
76          return 0;
77      }
78      client.print(phant.post());
79
80      while (client.available()) {
81          String line = client.readStringUntil('\r');
82      }
83
84      digitalWrite(LED_PIN, LOW);
85
86      return 1; // Return success
87  }
```

Když se nyní podíváme na adresu `https://data.sparkfun.com/streams/wpvZ9pE1qbFJAjaGd3bn`, zobrazí se nám seznam odeslaných informací (naše i cizí).

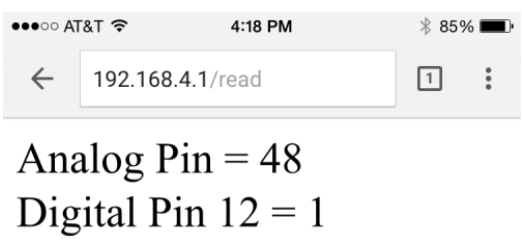
73.5 Jednoduchý server

Stejně jako u ESP připojeného k Arduinu, i zde si ukážeme, jak na modulu můžeme rozběhnout jednoduchý server. Následující program bude odesílat informace o aktuálních hodnotách pinů ADC a 12.

```
#include <ESP8266WiFi.h>

const char WiFiAPPSK[] = "sparkfun";

const int LED_PIN = 5;
const int ANALOG_PIN = A0;
const int DIGITAL_PIN = 12;

WiFiServer server(80);

void setup() {
    initHardware();
    setupWiFi();
    server.begin();
}

void loop(){
    WiFiClient client = server.available();
    if (!client){ //pokud nikdo neni pripojen
        return;
    }

    String req = client.readStringUntil('\r');
    Serial.println(req);
    client.flush();

    int val = -1;

    if (req.indexOf("/led/0") != -1){
        val = 0;
    }
    else if (req.indexOf("/led/1") != -1){
        val = 1;
    }
    else if (req.indexOf("/read") != -1){
        val = -2;
    }

    if (val >= 0){
        digitalWrite(LED_PIN, val);
    }
    client.flush();

   String s = "HTTP/1.1 200 OK\r\n";
    s += "Content-Type: text/html\r\n\r\n";
    s += "<!DOCTYPE HTML>\r\n<html>\r\n";
    if (val >= 0){
        s += "LED is now ";
        s += (val)?"on":"off";
    }
    else if (val == -2){
        s += "Analog Pin = ";
        s += String(analogRead(ANALOG_PIN));
```

```
54          s += "<br>";
55          s += "Digital Pin 12 = ";
56          s += String(digitalRead(DIGITAL_PIN));
57      }
58      else{
59          s += "Invalid Request.<br> Try /led/1, /led/0, or /read.";
60      }
61      s += "</html>\n";
62
63      client.print(s);
64      delay(1);
65      Serial.println("Client disonnected");
66  }
67
68  void setupWiFi(){
69      WiFi.mode(WIFI_AP);
70
71      uint8_t mac[WL_MAC_ADDR_LENGTH];
72      WiFi.softAPmacAddress(mac);
73      String macID = String(mac[WL_MAC_ADDR_LENGTH - 2], HEX) +
            String(mac[WL_MAC_ADDR_LENGTH - 1], HEX);
74      macID.toUpperCase();
75      String AP_NameString = "ESP8266 Thing " + macID;
76
77      char AP_NameChar[AP_NameString.length() + 1];
78      memset(AP_NameChar, 0, AP_NameString.length() + 1);
79
80      for (int i=0; i<AP_NameString.length(); i++){
81          AP_NameChar[i] = AP_NameString.charAt(i);
82      }
83
84      WiFi.softAP(AP_NameChar, WiFiAPPSK);
85  }
86
87  void initHardware(){
88      Serial.begin(115200);
89      pinMode(DIGITAL_PIN, INPUT_PULLUP);
90      pinMode(LED_PIN, OUTPUT);
91      digitalWrite(LED_PIN, LOW);
92  }
```

Když se nyní připojíme k síti, kterou modul vysílá, a zobrazíme si v prohlížeči jeho adresu, ukáže se nám jednoduchá webová stránka s naměřenými hodnotami.

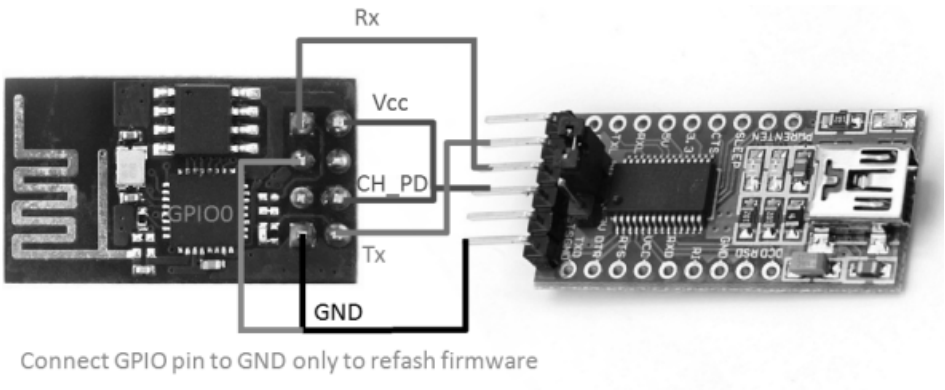


Obrázek 73.5: Web s naměřenými hodnotami

To by bylo k ESP8266 Thing vše. Poskytnuté informace by měly sloužit jako odrazový můstek pro další tvorbu.

Kapitola 74
ESP8266 s Arduino IDE

Posledním tématem, kterým se ještě budeme zabývat, je použití ESP8266 s Arduino IDE. Práce je zde velmi podobná jako u ESP8266 Thing. Vše propojíme následovně:



Obrázek 74.1: Připojení ESP8266 a převodníku [32]

Nyní v nabídce *Board* zvolíme *Generic ESP8266 Module*. Otevřeme si předchozí program `Hello World` a změníme pin LED na 2. Přes rezistor připojíme LED k pinu GPIO2 a zablikáme si.

```
1  #define ESP8266_LED 2
2
3  void setup()
4  {
5    pinMode(ESP8266_LED, OUTPUT);
6  }
7
8  void loop()
9  {
10   digitalWrite(ESP8266_LED, HIGH);
11   delay(100);
12   digitalWrite(ESP8266_LED, LOW);
13   delay(500);
14 }
```

Stejně tak můžeme využít knihovnu `ESP8266WiFi.h`, jejíž dokumentaci nalezneme na jejím GitHub[62] repository. Další příklady, jako například použití nodemcu, jsou bohužel nad rámec této kapitoly.

[62] NodeMCU GitHub – `https://github.com/ekstrand/ESP8266wifi`

Část XXI

Zdroje obrázků

Kapitola 75
Odkazy

[1] Arduino Community Logo
https://www.arduino.cc/en/Trademark/CommunityLogo

[2] Arduino Due
https://www.arduino.cc/en/Main/ArduinoBoardDue

[3] Arduino Esplora
http://arduino.cc/en/Main/ArduinoBoardEsplora

[4] Arduino Fio
http://arduino.cc/en/Main/ArduinoBoardFio

[5] Arduino Intel Galileo
http://arduino.cc/en/ArduinoCertified/IntelGalileo

[6] Arduino Leonardo
http://arduino.cc/en/Main/ArduinoBoardLeonardo

[7] Arduino Mega2560
http://arduino.cc/en/Main/ArduinoBoardMega2560

[8] Arduino Micro
http://arduino.cc/en/Main/ArduinoBoardMicro

[9] Arduino Mini
http://arduino.cc/en/Main/ArduinoBoardMini

[10] Arduino Nano
http://arduino.cc/en/Main/ArduinoBoardNano

[11] Arduino Pro s převodníkem
http://arduino.cc/en/Guide/ArduinoPro

[12] Arduino Robot
http://arduino.cc/en/Main/Robot

[13] Arduino Tre
http://arduino.cc/en/Main/ArduinoBoardTre

[14] Arduino Uno
http://arduino.cc/en/Main/ArduinoBoardUno

[15] Arduino Yún
http://arduino.cc/en/Main/ArduinoBoardYun

[16] BlueSMiRF Silver
http://www.hwkitchen.com/products/bluetooth-modem-bluesmirf-silver/

[17] Datasheet keypadu, http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Components/General/SparkfunCOM-08653_Datasheet.pdf

[18] Digital Read – INPUT
http://arduino.cc/en/Tutorial/DigitalReadSerial

[19] Digital Read – INPUT_PULLUP
http://arduino.cc/en/Tutorial/InputPullupSerial

[20] Dvousegmentový displej
http://www.gme.cz/led-display-20mm-green-hdsp-8606-p512-197

[21] ESP8266 WiFi
https://www.sparkfun.com/products/13678

[22] ESP8266 Thing
https://www.sparkfun.com/products/13231

[23] Esplora Pinout
http://pighixxx.tumblr.com/image/42953394937

[24] Ethernet shield
http://www.hwkitchen.com/products/arduino-eth-shield-rev3-without-poe-module/

[25] Konfigurační piny
http://arduino.cc/en/Guide/ArduinoWiFiShield

[26] Kosinusoida
http://leccos.com/pics/pic/kosinusoida.jpg

[27] LED matrix
http://www.adafruit.com/images/1200x900/1051-00.jpg

[28] LilyPad Arduino
http://arduino.cc/en/uploads/Main/LilyPad_5.jpg

[29] Piny ESP8266-1
http://raw.githubusercontent.com/guyz/pyesp8266/master/esp8266_pinout.png

[30] Piny WiFi shieldu
http://arduino.cc/en/Guide/ArduinoWiFiShield

[31] Propojení pro starší desky
http://arduino.cc/en/Guide/ArduinoWiFiShield

[32] Připojení ESP8266 a převodníku
http://iot-playground.com/images/articles/016/esp8266-reflash-firmware.png

[33] PWM
http://commons.wikimedia.org/wiki/File:PWM_duty_cycle_with_label.gif

[34] Rainbowduino
http://www.hwkitchen.com/products/rainbowduino-led-driver-platform/

[35] RGB LED matrix
http://pandatron.cz/elektronika2/snake8x8_fig2.jpg

[36] Rodina čipů ESP8266, http://g04.a.alicdn.com/kf/HTB1E1YKGFXXXbkXVXXq6xXFXXXG/201630778/HTB1E1YKGFXXXbkXVXXq6xXFXXXG.jpg

[37] Rozložení pinů ATmega168
http://arduino.cc/en/uploads/Hacking/Atmega168PinMap2.png

[38] Rozložení pinů ATtiny85
http://highlowtech.org/?p=1695

[39] Sedmisegmentový displej
http://www.gme.cz/led-display-8mm-orange-hdsp-u401-p512-194

[40] Sensoduino
https://play.google.com/store/apps/details?id=com.techbitar.android.sensoduino

[41] Sinusoida (akustika)
http://homen.vsb.cz/~ber30/texty/varhany/anatomie/pistaly_akustika.htm

[42] Sinusoida (matematika)
http://www.matematika.cz/content/images/sin1.png

[43] Squarewave
http://upload.wikimedia.org/wikibooks/en/e/e5/Square_Wave_T.svg

[44] Šestnáctisegmentový displej
http://cz.clipartlogo.com/image/sixteen-segment-display-field-clip-art_450488.html

[45] Tangentoida
http://www.drdelmath.com/slu_precalculus/trig_images/trig_graph_tangent.gif

[46] Zapojení znakového LCD
http://arduino.cc/en/uploads/Tutorial/LCD_schem.png

[47] Zvuková stopa, http://1.bp.blogspot.com/-GV_aYt3elV0/T84nErDNJ3I/AAAAAAAAAWI/me1tOJOqAm0/s1600/Waveform.jpg

Arduino poznámky

Arduino poznámky

Arduino poznámky

Arduino poznámky

Arduino poznámky

Arduino poznámky

Arduino poznámky

Arduino poznámky

Arduino poznámky

Arduino poznámky

Název: Průvodce světem Arduina

Autoři, autorská práva: Zbyšek Voda & tým HW Kitchen
Obálka: Autoři a Pavel Stříž
Sazba: Martin a Pavel Střížovi
Kontakt: martin@striz.cz, www.striz.cz

Počet stran: 241
Jazyk: Český
Sazba: Program TEX (formát LuaLATEX)
Písmo: CS-fonts

Papírové vydání knihy: www.hwkitchen.cz/arduino-kniha-pruvodce-svetem-arduina-2-vydani/

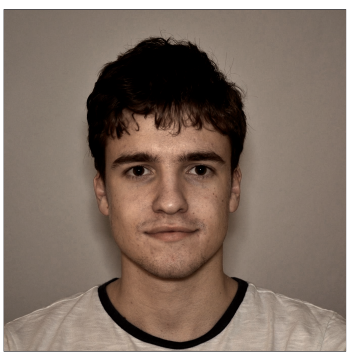
Motto: „Žádný člověk se nestane profíkem jen tak přes noc. . .“

Zbyšek Voda (fotka vlevo)

Už nějaký čas se zajímám o věci kolem Internetu věcí a otevřeného hardware a software. Tak jsem se také v roce 2010 dostal k Arduinu, pro které dodnes programuji a taky píšu články o práci s ním. Baví mě vymýšlet, jak staré věci používat novým způsobem.

Oldřich Horáček za tým HW Kitchen (vpravo)

Jsem zapálený hardwerář a tvůrce. Snažím se lidem přibližovat technologie, usnadňovat začátky a podporovat zajímavé projekty. Web Arduino.cz a tato kniha vznikla právě s tímto cílem. Moc mě baví věci udávat do pohybu a rozvíjet.



Nakladatelství Martin Stříž
Bučovice, 2018
Česká republika
www.striz.cz