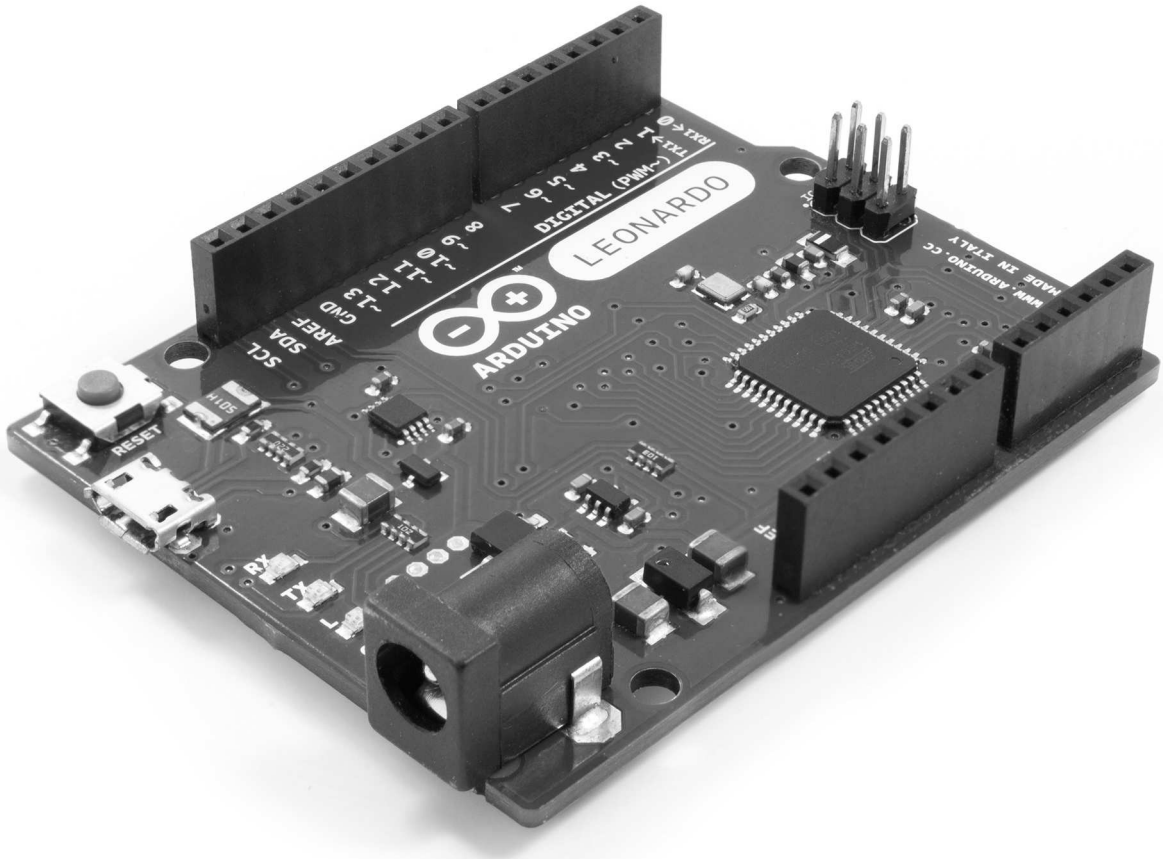
PRŮVODCE SVĚTEM ARDUINA

ZBYŠEK VODA & TÝM HW KITCHEN



ARDUINO
OPEN-SOURCE
COMMUNITY

Obsah

V průběhu roku 2014 postupně vznikal na serveru HW Kitchen 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 obsahuje osmnáct dílů tohoto seriálu. Text byl mírně upraven pro potřeby ebooku, ale jinak zůstaly články nezměněny.

Milý čtenáři. I přes všechnu moji snahu není vyloučeno, že se do textu vloudily chyby. Pokud na některou narazíš, napiš mi prosím na email zbysekvoda@seznam.cz.

Část I

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 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 tomto článku 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 pořizovat, v té době rozšířené a drahé desky 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.) A to nejenom prodejem vlastních desek, ale i sdílením všech schémat a návodů (jedná se o Open Source projekt). Programová část Arduina byla založena na Processing, což je programovací jazyk s vlastním editorem, určený k výuce programování. V dnešní době se prodalo již několik stotisíc desek Arduino. Důkazem, že tato platforma není mrtvá, může být i to, že nedávno byl ohlášen vývoj nové a výkonné desky Arduino Galileo, která vzniká ve spolupráci s Intelem. Za osm let vývoje již vzniklo spoustu různých typů Arduina. Jelikož se jedná o opensource projekt, vznikalo společně s hlavní linií projektu i spoustu dalších, neoficiálních typů, takzvaných klonů. Nejdříve si ale představíme oficiální desky.

Obrázek 1.1: Oficiální logo platformy Arduino

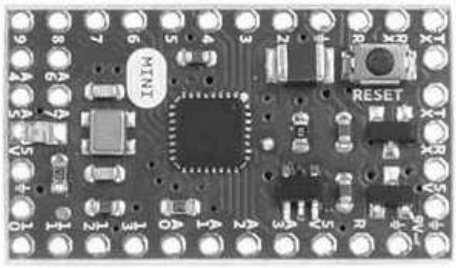
Kapitola 2

Typy desek

Srdcem každého Arduina 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 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 další desky. Na většině desek je mimo hlavního čipu ještě převodník, který umožňuje komunikaci mezi PC (USB) a čipem. Setkám se však 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, jejichž čip má v sobě tento převodník zabudovaný.

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

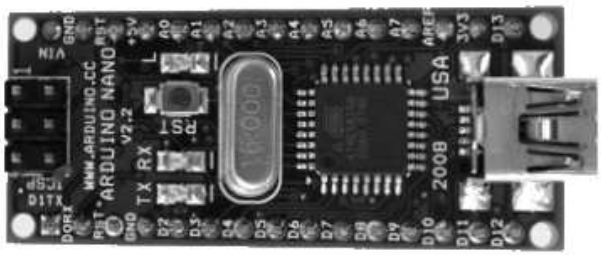
2.1 Arduino Mini



Obrázek 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 2 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 a podobně.

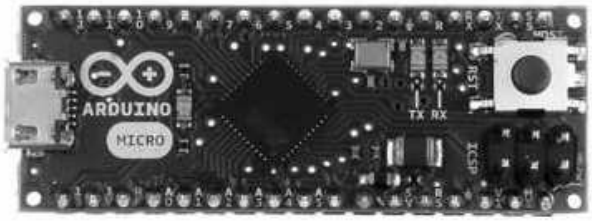
2.2 Arduino Nano



Obrázek 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 tak nutnost mít společně s deskou ještě další programovací prostředek.

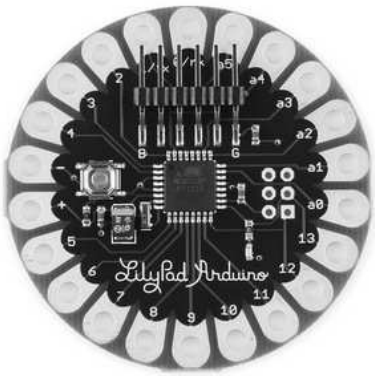
2.3 Arduino Micro



Obrázek 2.3: Arduino Micro

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 a 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.

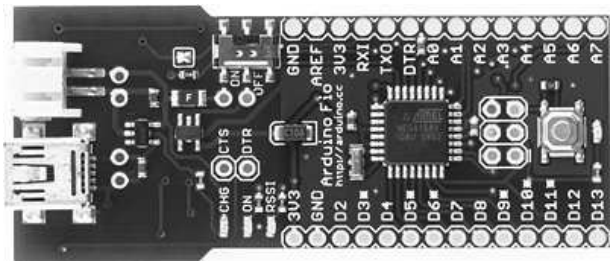
2.4 LilyPad Arduino



Obrázek 2.4: Arduino Lilypad

Již při prvním pohledu je jasné, že Lilypad Arduino není úplně typické. Jedná se totiž o verzi přizpůsobenou k nošení na textilu, kdy jsou spoje tvořeny vodivou nití. Tak se dá vyrobit například cyklistická mikina s přišitými blinkry. Existuje více druhů této desky. Můžeme se setkat s verzí s USB a čipem ATmega32u4, nebo bez USB ve verzi ATmega328 a dalšími.

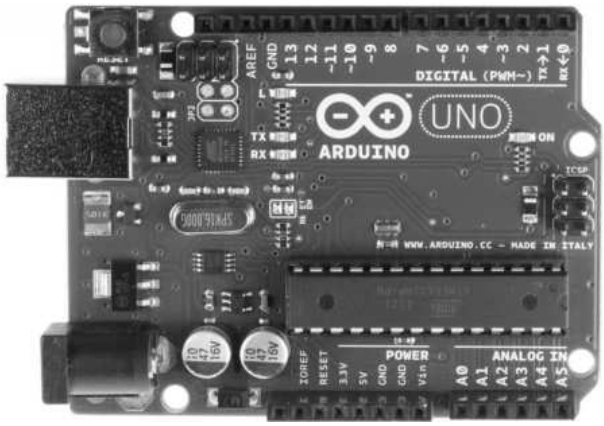
2.5 Arduino Fio



Obrázek 2.5: Arduino Fio

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

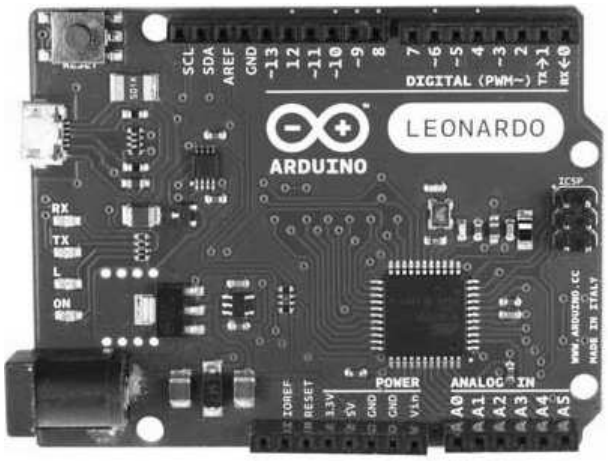
2.6 Arduino Uno



Obrázek 2.6: Arduino Uno

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 nějakého projektu.

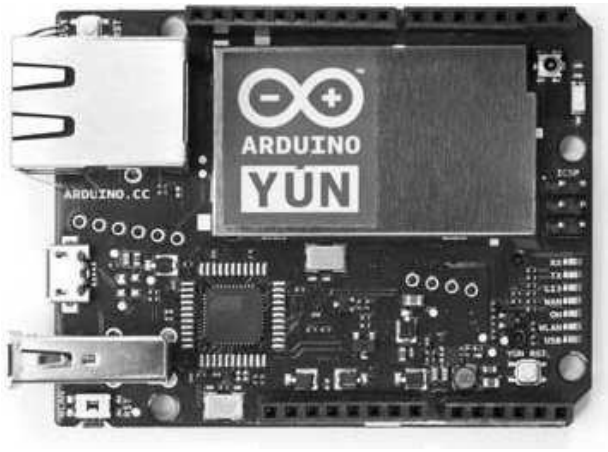
2.7 Arduino Leonardo



Obrázek 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. Specifika tohoto čipu si popíšeme dále.

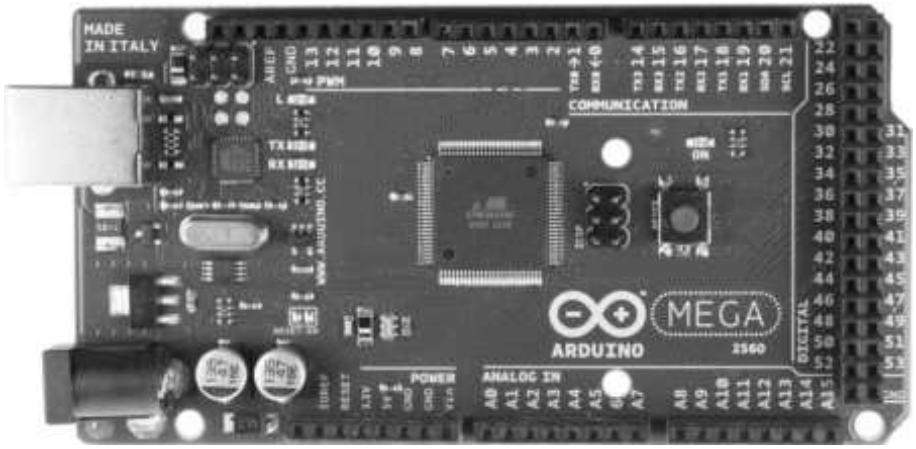
2.8 Arduino Yún



Obrázek 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 linuxu Linino. 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 ATmeaga32u4 i normální USB pro potřeby linuxu a Ethernet port pro připojení k síti. Můžeme tedy například posílat naměřené hodnoty přímo na webový server.

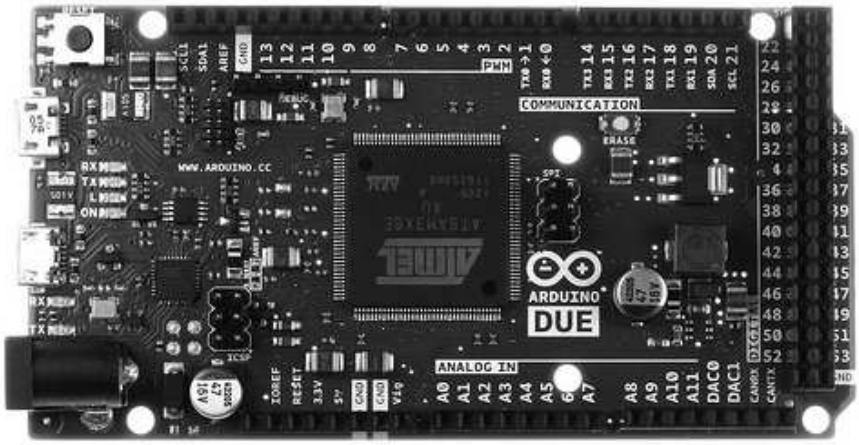
2.9 Arduino Mega2560



Obrázek 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.

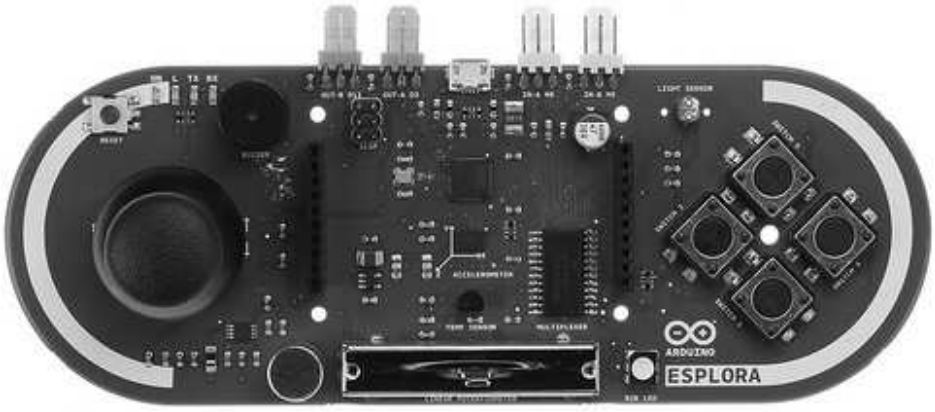
2.10 Arduino Due



Obrázek 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 84Mhz a jeho jádro je 32-bitové, což je oproti ostatním deskám s 8-bity a maximálně 16MHz opravdu velký skok. Na desce nalezneme dva microUSB konektory. Jeden pro programování čipu, druhý pro připojení zařízení, jako jsou myši, klávesnice, telefony a jiné.

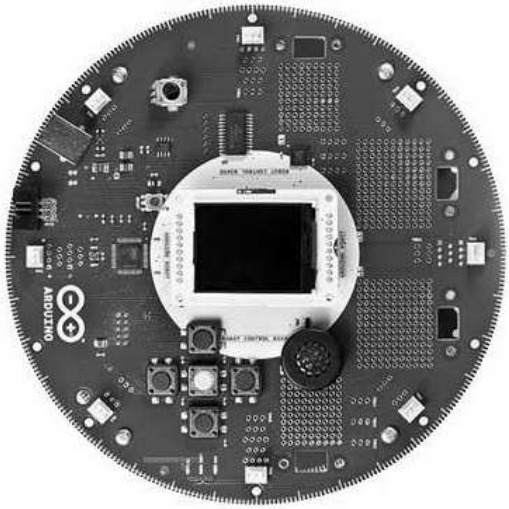
2.11 Arduino Esplora



Obrázek 2.11: Arduino Esplora

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. 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.

2.12 Arduino Robot



Obrázek 2.12: Arduino Robot

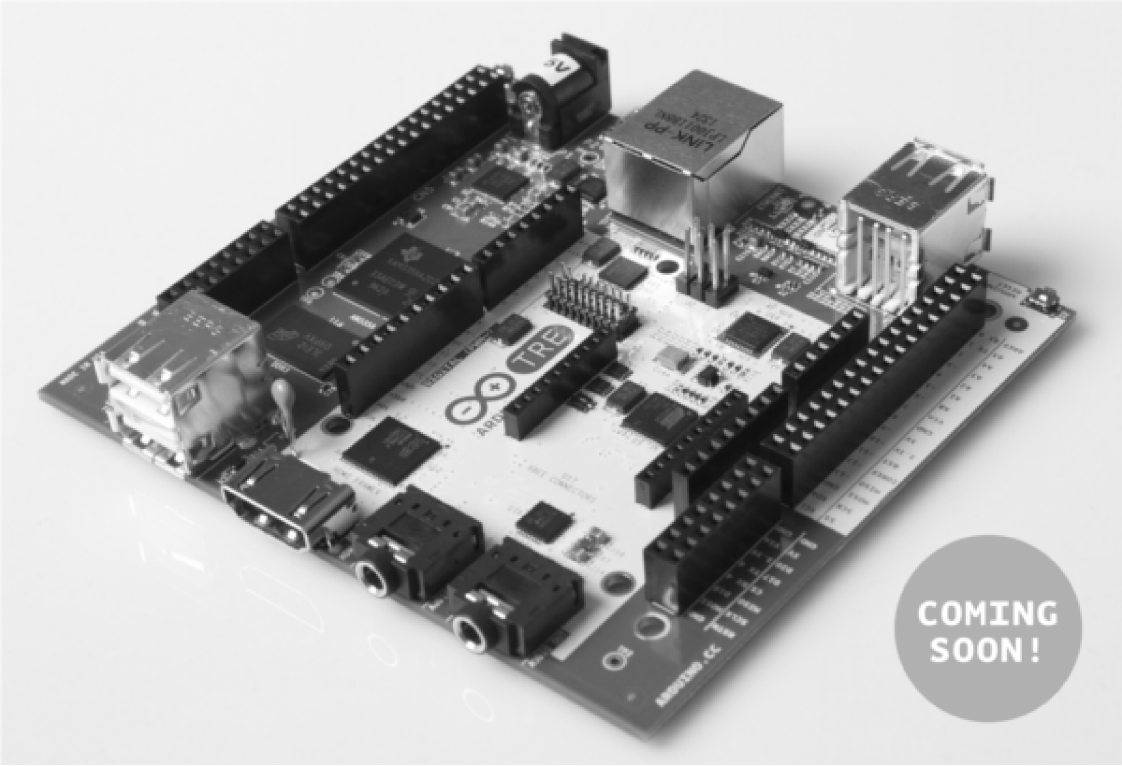
Jak už název napovídá, jedná se o set pro vytvoření vlastního chytrého robota. Jeho mozkem je procesor ATmega32u4. Zajímavostí je přítomnost kompasu.

2.13 Arduino Intel Galileo

Obrázek 2.13: Arduino Intel Galileo

Tato verze vznikla ve spolupráci se společností Intel. Jedná se o první desku, která běží na čipu Intel® Quark SoC X1000, což je 32-bitový 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



Obrázek 2.14: Arduino Tre

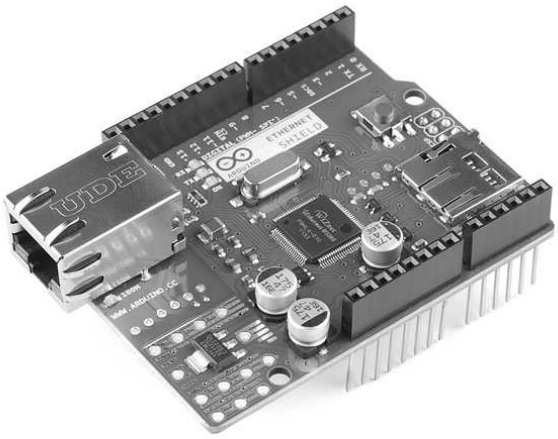
V současné době chystané Arduino Tre bude zatím nejvýkonnější typ. Mělo by obsahovat 1 GHz procesor, schopný běhu velmi náročných výpočetních aplikací. Stejně jako Arduino Yún bude obsahovat dva procesory. Jeden pro jádro Arduina a druhý pro linux. Na desce také nalezneme HDMI port, dva audio konektory, 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. V říjnu 2014 ale stále není jasné, kdy se skutečné začne prodávat.

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. 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 nasunout do zdířek na Arduinu. Stejně jako desek existuje i celá řada shieldů. Z těch oficiálních jsou to ale Ethernet Shield, Wifi Shield, Motor Shield a Další. Při výběru je však nutné dát si pozor na to, aby byl vybraný shield s Arduinem kompatibilní.

Na obrázku vidíte, jak vypadá takový Ethernet shield.



Obrázek 3.1: Arduino Ethernet Shield

Kapitola 4

Arduino klony

Jak už jsem naznačil 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, součástky 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
- A další...

Část II

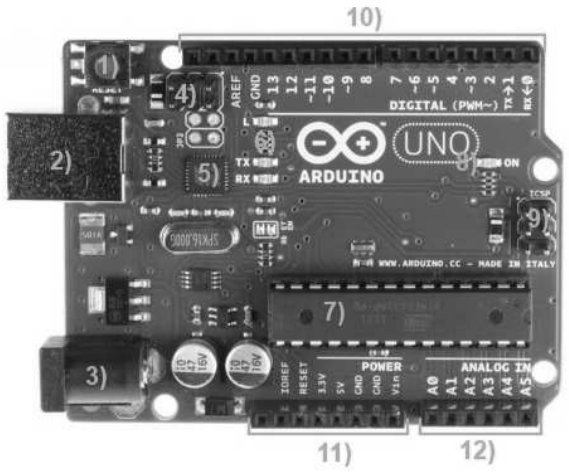
Programujeme Arduino

V první kapitole jsme se mohli dozvědět něco o historii Arduina. Také jsme si ukázali desky z oficiální řady a zmínili jsme i existenci shieldů. V závěru článku přišla řeč na neoficiální klony. Dnes už se tedy můžeme pustit do praktického použití. Na začátku si vybereme vhodné Arduino a na něm si ukážeme, co všechno se dá na desce najít. Řekneme si něco o programovacím jazyku a o vývojovém prostředí pro Arduino. Na závěr si vyzkoušíme první program, který bude blikat LED diodou.

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 počítače. Těmito modely jsou například DUE, 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. Níže uvedené části najdeme v různých obměnách na většině desek.



Obrázek 5.1: Co nalezneme na desce Arduino

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, BT), 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 nenajdeme.

5. USB-serial převodník. Ten se stará o komunikaci mezi hlavním čipem a PC. Plní zde roli překladatele. 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 č. 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

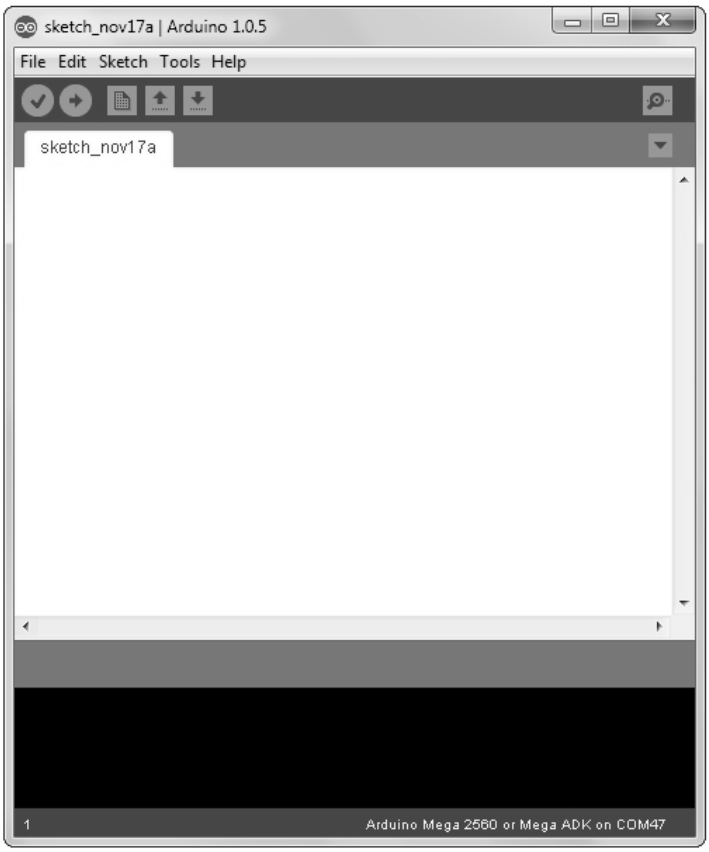
6.1 Historie

Arduino IDE (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 zde [en]. V této době je poslední verzí Arduino 1.0.5 (1.5 je stále ve verzi Beta). Najdeme si tedy na 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[en]. Musíme také zmínit další velký operační systém, jímž je Mac OS. Návod na instalaci nalezneme zde[en]. Následující část článku se bude věnovat použití Arduina s operačním systémem Windows. Předpokládejme, že máme stažený ZIP archiv verze 1.0.6. 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:Usersjmeno_uzivateleDocuments. 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



Obrázek 6.1: Arduino IDE

Vývojové prostředí můžete vidět na obrázku. V prvním řádku navigačních prvků nás bude zajímat pouze rozbalovací nabídka Tools, ve které nalezneme nastavení pro připojení a programování desky. Její funkce si popíšeme později. V dalším řádku nalezneme několik ikon. Jako první z leva nalezneme ikonu s fajfkou – Verify. Ta po kliknutí spustí kontrolu programu a zkontroluje kód. Pokud nalezne nějakou chybu, v syntaxi ji zvýrazní. Vedle nalezneme ikonu s šipkou doprava – 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 v Dokumentech). 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 příště. Velký bílý prostor slouží k zápisu kódu a černý prostor dole zobrazuje informační a chybové výpisy z běhu prostředí.

6.4 Programovací jazyk

Arduino je možné programovat v jazyce C nebo C++. Nejjednodušší je však používat knihovnu Wiring. Ta je v současné době pro programování Arduina velmi rozšířená. Kvůli její komplexnosti se o ní občas mluví jako o samostatném programovacím jazyku. 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 restart, 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. 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 podtržítkem 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:

```
1  jednoradkovy komentar
2  //kdyz radek zacina dvojitym podtrzitkem, jedna se o jednoradkovy komentar
3  //vse, co je v radku za nim program ignoruje
4  to co se vsak nachazi na dalsim radku je brano jako normalni kod
5
6  druhym typem jsou viceradkove komentare
7  /* ty zacinaji podtrzitkem a hvezdickou
8  mohou
9  mit
10 libovolny
11 pocet
12 radku
13 a konci hvezdickou a podtrzitkem */
```

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 tento set.
6. 330 ohm rezistor – zapojení sice bude fungovat i bez něj, ale je nutné ho použít.

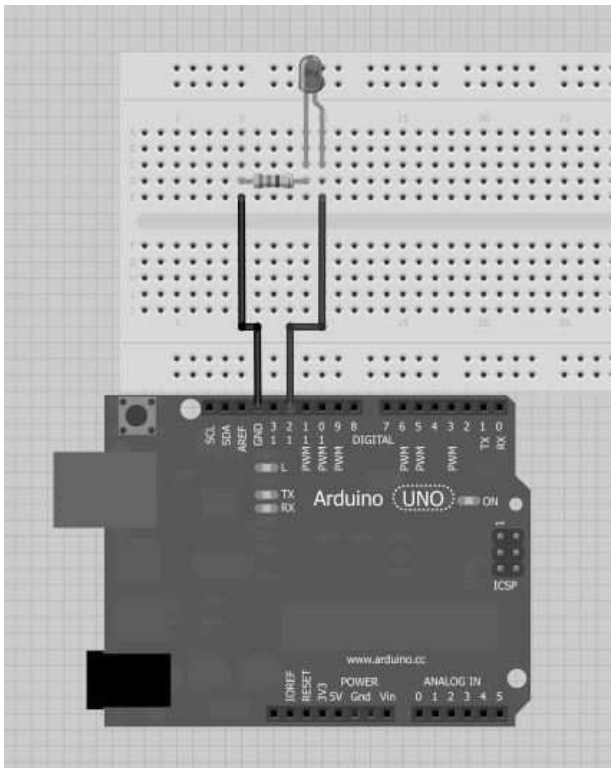
7.2 Připojení Arduina a PC

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 stažená složka Drivers. 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.

7.3 Zapojení

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



Obrázek 7.1: Schéma zapojení příkladu Blink

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() {
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 }
```

Část III

Základní struktury jazyka Wiring

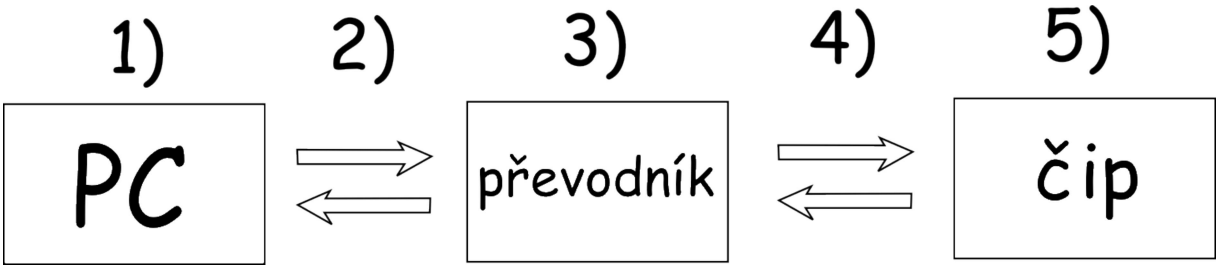
V předchozí části jsme si ukázali první program, ve kterém Arduino blikalo LED diodou. Úvodní seznámení je tedy za námi a můžeme se pustit do dalšího programování. V dnešním článku se podíváme na základní náležitosti jazyka Wiring. Na začátek si vysvětlíme, jak Arduino komunikuje s počítačem. Poté si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

Kapitola 8

Sériová komunikace

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 (BT, Ethernet, Wifi...), ale těmi se dále zabývat nebudeme. Popišme si tedy proces programování, se kterým se setkáme u většiny desek.

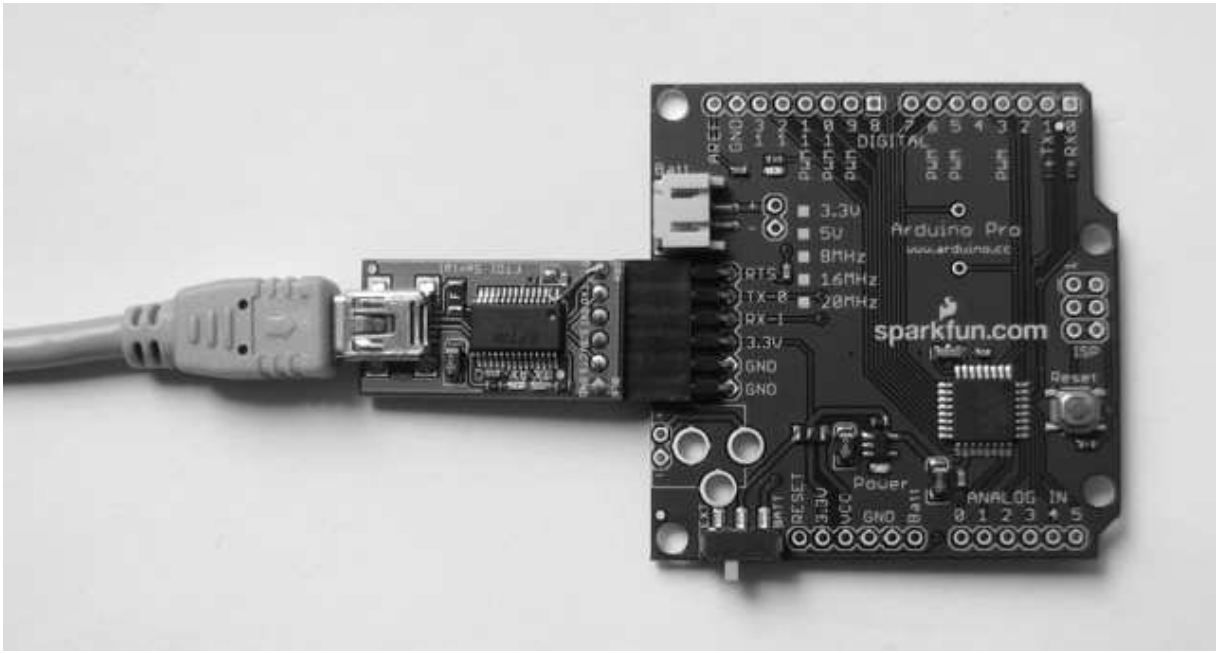
Co je ke komunikaci potřeba:



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 na pevno připájený k základní desce Arduina.
   (b) Převodník, který mají některé čipy (ATmega32u4...) přímo v sobě.
   (c) Externí převodník, který musíme při programování Arduina 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.

Takto vypadá Arduino Pro s připojeným externím převodníkem.



Obrázek 8.2: Arduino Pro s externím převodníkem

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 místo ve kterém se dají uchovávat data. Každá proměnná má vlastní jméno, datový typ a hodnotu. Často se používají například tam, kde se v programu dokola opakují ty samé hodnoty. Praktické využití proměnných si ukážeme na následujícím příkladu napsaném 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.

```
1  int x = 10; //tuto promennou muzeme pouzit vsude
2
3  void setup() {
4      int y = 11;
5      //uvnitr teto funkce muzeme pouzit promenne x a y
6  }
7
8  void loop() {
9      int z = 12;
10     //zde muzeme pouzit promenne x a z
11 }
```

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 jsem naznačil 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.

Číselné datové typy

- byte – Proměnná datového typu byte má velikost 8 bitů a slouží k uchování celých čísel. Její rozsah je 28 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 16-bit hodnotu – tedy od -32,768 do 32,767. U Arduino DUE s procesorem SAM je tato hodnota 32-bitová a může obsahovat čísla od -2,147,483,648 do 2,147,483,647.
- long – Slouží k uchování celočíselných 32 bitových 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 používá desetinná tečka. Jeho velikost je 32 bitů. Můžeme v něm ukládat hodnoty od -3,4028235 * 1038 do 3,4028235 * 1038.

Logický datový typ

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

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.

```
1  //byte
2      byte a = 12;
3  //integer
4      int b = 400;
5  //long
6      long c = 12121212;
7  //float
8      float d = 1.256;
9
10 //boolean
11     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 [EN] 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
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, 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 minulém článku, 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.

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. Ve verzi Arduino UNO tedy můžeme používat piny 0 až 13. Tímto 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 v minulém článku, 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

Ke zjištění, zda proud do vstupu teče, nebo ne se používá funkce digitalRead(). Ta, narozdíl od předchozích dvou funkcí, potřebuje pouze jeden parametr, kterým je číslo pinu. 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
5 //pokud ne, tak LOW
```

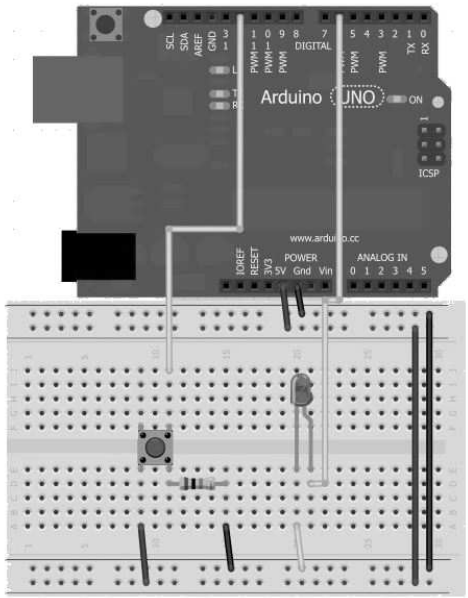
Kapitola 12

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. 10K ohm resistor
6. LED diodu

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, zkontrolujte středníky. Ty se píší na konec řádku za: deklarací proměnné, za přiřazením hodnoty proměnné a za voláním funkce (pinMode...).

Část IV

Pokročilejší struktury jazyka Wiring

V tomto díle navážeme na informace z minula a ukážeme si 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 v minulém dílu. Můžeme je rozdělit do tří skupin.

13.1 Logické konstanty

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

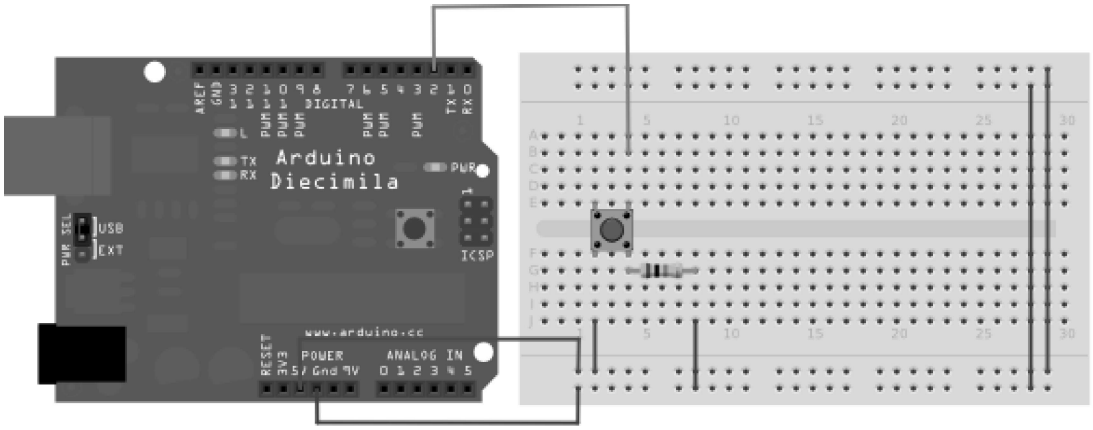
- false
  - Konstanta false má hodnotu 0.
- true
  - U konstanty true je situace trochu komplikovanější. Mohlo by se zdát, že má hodnotu 1, ale není to úplně přesné. Při logických operacích totiž jazyk Wiring vnímá jakékoliv nenulové číslo typu integer jako true.

13.2 Typ digitálního pinu

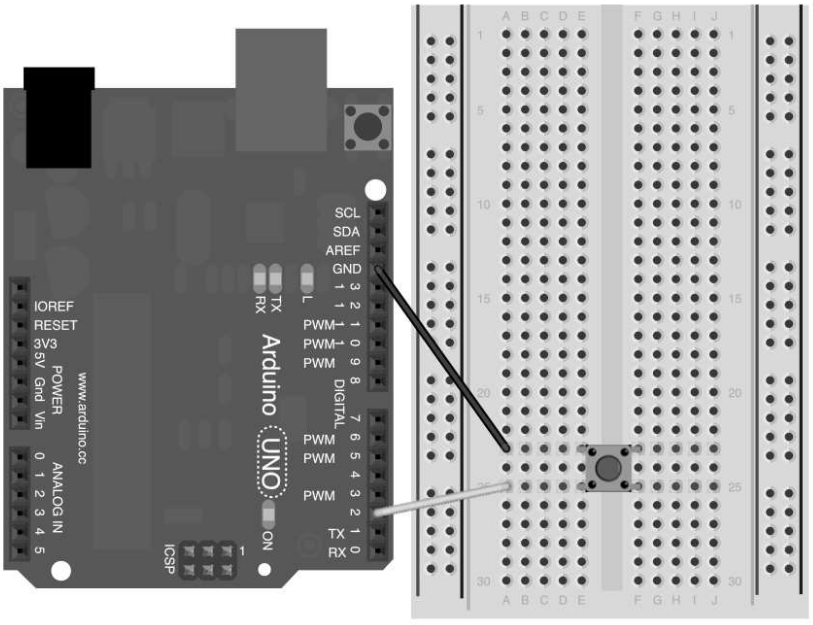
V minulém díle jsme při určování, zda se bude pin chovat jako vstup, nebo jako výstup, používali ve funkci pinMode() dvě konstanty – OUTPUT a INPUT. Dnes si k nim přidáme ještě třetí možnost INPUT_PULLUP.

- 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 ukázali v minulém dílu. V jeho zapojení si všimněme, že je tento pin stále připojen ke GND přes 10k ohm resistor. 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() 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 Proud na digitálních pinech

Jsou pouze dvě možné hodnoty, jaké může mít proud při čtení a zápisu pomocí funkcí digitalRead() a digitalWrite(). Jsou to hodnoty LOW a HIGH.

- HIGH
  - Při čtení pomocí funkce digitalRead() je vyhodnocena hodnota napětí jako HIGH, pokud je větší než 3V. Když použijeme funkci digitalWrite(pin, HIGH), na výstupu bude právě 5V.
- LOW
  - Při čtení je stav napětí vyhodnocen jako LOW, pokud je jeho velikost menší než 2V. Při zápisu je hodnota 0V. Fakticky ale může "přijmout"napětí do velikosti 5V (což nelze, pokud je nastaveno HIGH).

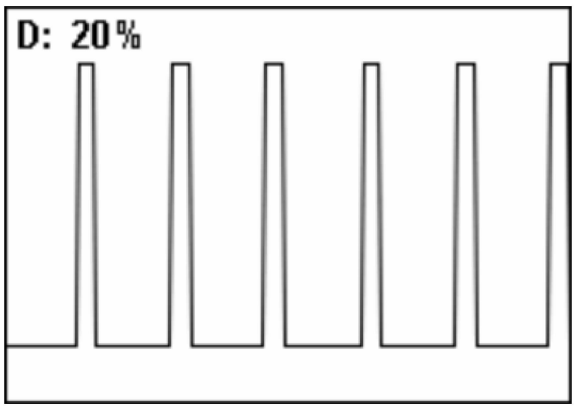
Kapitola 14

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 digitalWrite(číslo_pinu, hodnota), kdy hodnota může být v rozsahu 0 až 255. Slovo analogové jsem dal do uvozovek, protože se ve skutečnosti o žádné analogové hodnoty nejedná. Pokud bychom chtěli skutečně analogovou hodnotu v rozsahu například 0-5V, museli bychom použít externí D/A převodník. Tato funkce totiž na vybraných pinech generuje PWM signál, což je jakási digitální "náhražka"analogového signálu. Ta v praxi funguje tak, že rychle střídá 0 a 5V. 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 +5V ku stavu 0V se pak odvíjí intenzita svícení LED diody a podobně. Při volání funkce analogWrite(pin, 127) je tedy přibližně 50% času nastaveno napětí +5V a 50% času 0V. Průběh napětí při různých hodnotách můžete vidět na obrázku.



Obrázek 14.1: Graf PWM modulace

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 rozlišení 10 bitů, což odpovídá hodnotám od 0 do 1023. Například u Arduino DUE se ale můžeme setkat až s 12-bitovým rozlišením. Zde se dá nastavit požadované rozlišení pomocí funkce analogReadResolution(). My se ale budeme zabývat obyčejným Arduinem. Syntaxe je jednoduchá: proměnná = analogRead(pin). Nejjednodušším příkladem použití je měření hodnot na potenciometru. Pokud bychom chtěli měřit například stav fotoresistoru, museli bychom ho zapojit do děliče napětí s vhodným resistorem. Použití obou analogových funkcí si ukážeme na zapojení s LED diodou a potenciometrem.

Kapitola 15

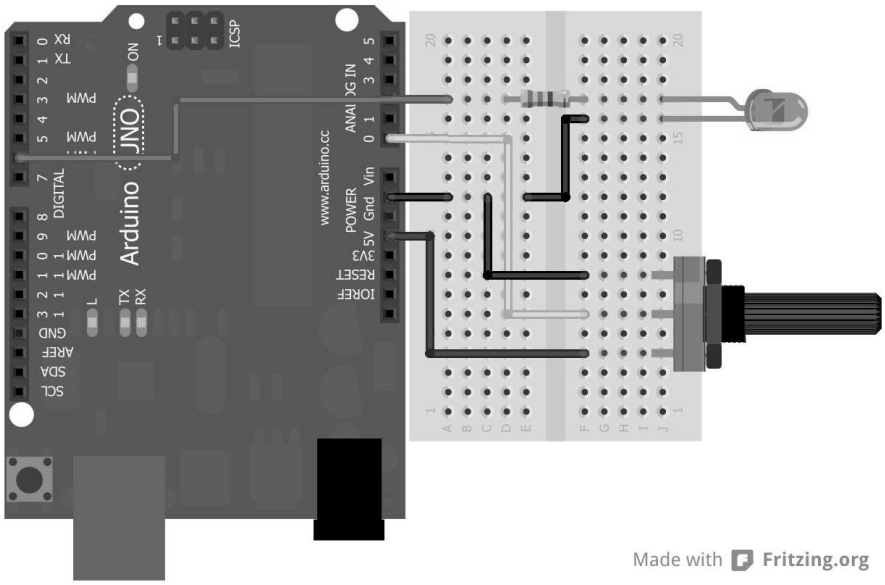
Příklad: Regulace jasu LED

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

Budeme potřebovat:

1. Deska Arduino
2. Nepájivé kontaktní pole s vodiči
3. LED dioda
4. potenciometr
5. 330 ohm resistor

Na trhu nalezneme celou řadu potenciometrů. Nejčastěji se používají ty s odporem kolem 10k ohm 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. Z funkce analogRead() vychází hodnoty 0 až 1023, kdežto analogWrite() čeká 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().

```
1  byte led = 6; //pin s LED diodou
2  byte pot = A0; //pin s pripojenym potenciometrem
3  int val; //promenna pripravena k uchovani hodnot
4
5  void setup() {
6      //sem nic nepiseme
7  }
8
9  void loop() {
10     val = analogRead(pot)/4;
11     //cteni hodnoty na A0 a uprava rozsahu
12     analogWrite(led, val);
13     //generovani PWM
14 }
```

Kapitola 16

Podmínky

Pokud chceme, aby se určitá část kódu provedla pouze v určitých případech, přicházejí na řadu podmínky. Existují tři základní struktury, které rozdělují 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.
- 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).

```
1  //podminky s jednim prikazem
2
3  if(x > 120) digitalWrite(LEDpin, HIGH);
4
5  if(x > 120)
6  digitalWrite(LEDpin, HIGH);
7
8  if(x > 120){ digitalWrite(LEDpin, HIGH); }
9
10 //podminky s vice prikazy - je nutne pouzit slozene zavorky
11 if(x > 120){
12   digitalWrite(LEDpin1, HIGH);
13   digitalWrite(LEDpin2, HIGH);
14   ...
15 }
```

16.4 else if()

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

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

16.6 Switch

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;

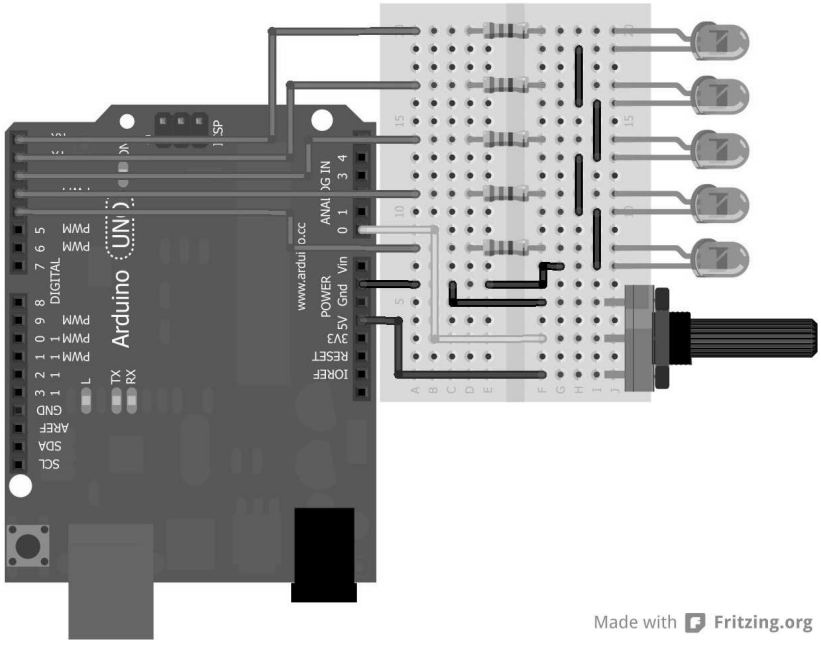
Syntaxe je následující:

```
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 kapitoly si trochu pohrajeme s podmínkami. Vytvoříme aplikaci, která bude číst hodnotu z potenciometru a podle ní vybere led diodu. K tomuto příkladu budeme potřebovat stejné vybavení, jako u toho minulého, jen s větším počtem LED diod a resistorů. Pro předváděcí účely jsem zvolil pět diod.



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);
    }
    else if(val > 600){
        digitalWrite(led[1],HIGH);
    }
    else if(val > 400){
        digitalWrite(led[2],HIGH);
    }
    else if(val > 200){
        digitalWrite(led[3],HIGH);
    }
    else{
        digitalWrite(led[4],HIGH);
    }

    delay(250);
    digitalWrite(led[0],LOW);
    digitalWrite(led[1],LOW);
    digitalWrite(led[2],LOW);
    digitalWrite(led[3],LOW);
    digitalWrite(led[4],LOW);
}
```

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.

Část V

Sériová komunikace a cykly

V této části splním sliby, které jsem v dal dříve. Ukážeme si, jak může Arduino komunikovat přes sériovou linku s jinými zařízeními a deskami Arduino a vysvětlíme si, jak fungují cykly.

Kapitola 18

Sériová komunikace

Ve kapitole Základní struktury jazyka Wiring 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. S počítačem ale komunikuje pouze jedna základní linka.

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.

18.1 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, 14400, 19200, 28800, 38400, 57600 a 115200). 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 }
```

18.2 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  }
```

```
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.

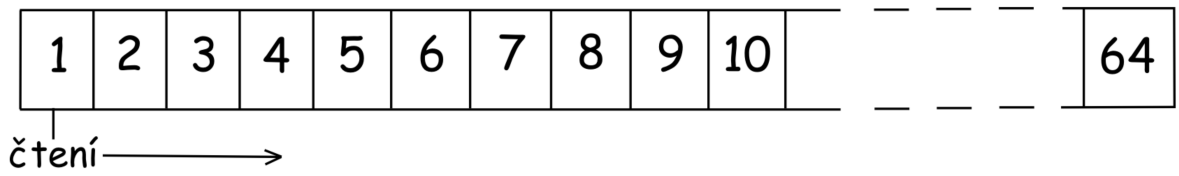
```
1  //pouziti jednoho parametru
2  Serial.print(97); //vypise: 97
3  Serial.print(2.123456); //vypise: 2.12
4  Serial.print('a'); //vypise: "a"
5  Serial.print("ABCDEFGHIJ"); //vypise: "ABCDEFGHIJ"
6
7  //pouziti nepovinneho operatoru pro typ dat
8  Serial.print(97, DEC); //vrati: 97
9  Serial.print(97,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

Před chvílí jsem zmínil, 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); //vrati: 97
2 Serial.print('a', BIN); //vrati: 1100001
3 Serial.print('a', OCT); //vrati: 141
4 Serial.print('a', HEX); //vrati: 61
```

18.3 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 nejnižší 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.



Obrázek 18.1: Buffer

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 čísla) vlastní byte paměti.

Funkce Serial.available() a Serial.read() jsem dal 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 }
```

18.4 Ukončení komunikace – Serial.end()

Tato funkce se v praxi moc často nepoužívá. Při jejím volání ale 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.

Kapitola 19

Cykly

Jistě si vzpomenete, že jsme se v minulých dílech dostali do situace, kdy jsme museli psát prakticky stejnou věc stále dokola, 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.

19.1 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.

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

19.2 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 tohle a vrať se na začátek. Pokud není, skonči“. Cyklus while() se tedy nemusí provést vůbec. Používá se následující syntaxe:

```
1  while(podminka){
2      prikazy...
3  }
```

Příklad 1. – ukázka cyklu

```
1  boolean x = true;
2
3  void setup() {
4    Serial.begin(9600);
5    while(x){
6      Serial.println("OPAKUJI");
7      x = false; //cyklus se tedy provede jednou
8    }
9  }
10
11 void loop() {
12
13 }
```

Příklad 2. – výpis všech čísel od 0 do 99

```
1  byte x = 0;
2
3  void setup() {
4    Serial.begin(9600);
5    while(x < 100){
6      Serial.println(x);
7      x++;
8    }
9  }
10
11 void loop() {
12
13 }
```

19.3 Cyklus do. . . while()

Cyklus do..while() se od while() liší pouze v tom, že se podmínky kontrolují až na konci. Dříve tedy dojde k provedení příkladů a poté až ke kontrole podmínek. 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:

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

Příklad 1. – ukázka cyklu

```
1  boolean x = true;
2
3  void setup() {
4    Serial.begin(9600);
5    do{
6      Serial.println("OPAKUJI");
7      x = false; //cyklus se take provede pouze jednou
8    } while(x);
9  }
10
11 void loop() {
12
13 }
```

Příklad 2. – počítání od 0 do 99

```
1  byte x = 0;
2
3  void setup() {
4    Serial.begin(9600);
5    do{
6      Serial.println(x);
7      x++;
8    }while(x < 100);
9  }
10
11 void loop() {
12
13 }
```

19.4 Cyklus for()

Tento cyklus se používá asi nejčastěji. Jedná se většinou o případy, kdy známe počet opakování cyklu. Také se od ostatních cyklů liší tím, že potřebuje mít pro svoji funkci vlastní proměnnou, která hlídá počet opakování.

```
1  for(inicializace promenne a nastaveni hodnoty; podminka; operace provadene pri kazdem opakovani){
2      prikazy...
3  }
```

Příklad 1. – výpis čísel od 0 do 99

```
1  void setup() {
2      Serial.begin(9600);
3      for(int i = 0; i < 100; i++){
4          Serial.println(i);
5      }
6  }
7
8  void loop() {
9
10 }
```

Kapitola 20

Příklad: Had z LED diod

V tomto díle si vytvoříme hada z pěti LED diod (počet je však volitelný) a ukážeme si, jak se dá cyklus for využít v praxi. Budeme potřebovat:

1. Desku Arduino
2. PC
3. Nepájivé kontaktní pole s vodiči
4. 5x 330 ohm resistor
5. 5x LED diodu

```
1  byte led[] = {2,3,4,5,6}; //piny s LED diodami
2  byte pocet = 5; //pocet diod
3  int rychlost = 1000; //jakou prodlevu maji jednotliva bliknuti
4
5  void setup() {
6      for(int i = 0; i < pocet; i++){
7          pinMode(led[i], OUTPUT); //nastaveni pinu
8      }
9  }
10
11 void loop() {
12     for(int i = 0; i < pocet; i++){
13         digitalWrite(led[i], HIGH);
14         delay(rychlost/2);
15         digitalWrite(led[i], LOW);
16         delay(rychlost/2);
17     }
18 }
```

Část VI

Užitečné funkce

V této části si ukážeme, jak může Arduino vnímat čas. Podrobněji se podíváme na funkce čekání (delay) a další. Poté si ukážeme možnosti matematických operací a na závěr si představíme funkce pro generování náhodných čísel.

Kapitola 21

Čas

V základní výbavě mají desky Arduino čtyři funkce pro práci s časem. Jsou to funkce delay(), delayMicroseconds(), millis() a micros(). První dvě a druhé dvě fungují na stejném principu, jenom pracují s jinými 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 časovače, který poté znovu začne počítat od nuly. Funkce millis() se využívá například tam, kde je třeba čekat, 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+100){
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 16 MHz procesorů 4 mikrosekundy a u 8 MHz 8 mikrosekund. Výstupem funkce tedy bude násobek čtyř, nebo osmi.

Možná se ptáte, proč jsou maximální hodnoty parametrů, nebo vrácených čísel takové, jaké jsou. Je tomu tak, protože funkce pro práci s časem používají dva datové typy, které jsem ve článku Základní struktury jazyka Wiring neuvedl. Jsou to unsigned int a unsigned long. Datový typ unsigned int má stejný rozsah 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 % (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]; //nejmensi cislo
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 nejm = pole[0]; //nejvetsi cislo
6
7      Serial.begin(9600);
8
9      for(int i = 1; i < delka; i++){
10         nejm = max(nejm, pole[i]);
11     }
12
13     Serial.println(nejm);
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, abs(x) = x. Pokud je x ¡ 0, potom abs(x) = -x. Laicky 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.

```
1  x = constrain(upravovana hodnota, dolni mez, horni mez);
2
3  x = constrain(1,10,100); //x = 10
4  x = constrain(150,10,100); //x = 100
5  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 ořízntutí 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);

```
1  //uprava jasu LED pomoci potenciometru
2
3  int analog, pwm;
4
5  void setup() {
6      Serial.begin(9600);
7  }
8
9  void loop() {
10     analog = analogRead(A0);
11     pwm = map(analog, 0, 1023, 0, 255);
12     Serial.print("Analog: ");
13     Serial.print(analog);
14     Serial.print(" PWM: ");
15     Serial.println(pwm);
16
17     analogWrite(11, pwm);
18 }
```

22.7 pow()

Funkce pro mocnění 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.

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

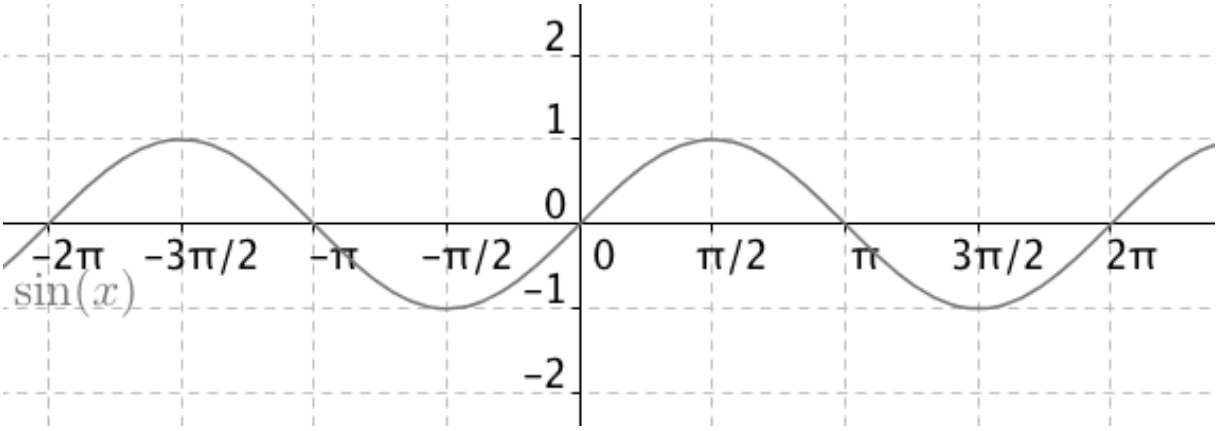
22.9 Goniometrické funkce

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. Plný úhel (360 stupňů) je roven 2*PI radiánů, 180 = PI radiánů atd. Platí mezi nimi převodní vztah: radiány = (stupně * 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 2PI 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 pomocí pomlček přes sériovou linku.

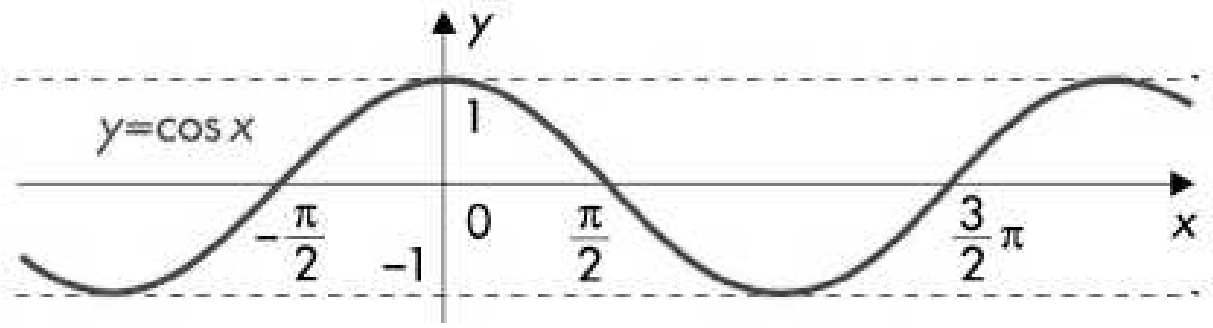


Obrázek 22.1: Graf funkce sinus

```
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 }
```

22.9.2 cos()

Funkce cos() slouží k určení kosinu dané hodnoty. Je definovaná jako délka přilehlé 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 = 2PI, amplituda = 1. Na příkladu můžete vidět výpis kosinusoidy.

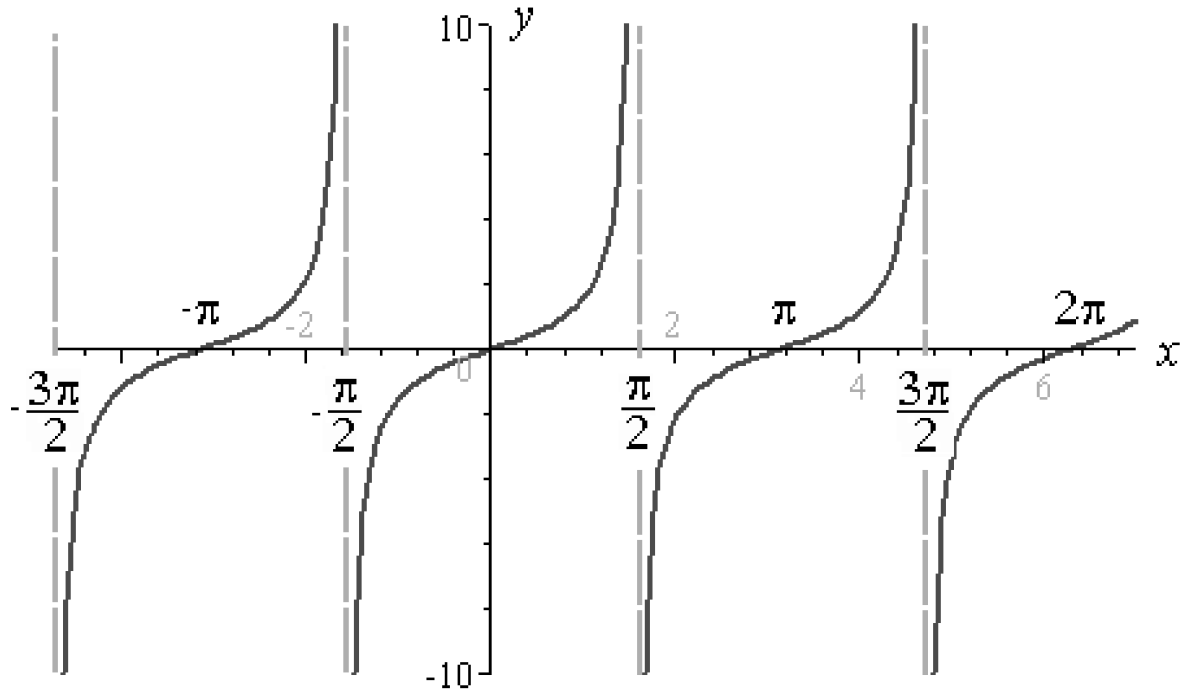


Obrázek 22.2: Graf funkce cosinus

```
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 tangents. Ta je dána jako poměr protilehlé strany ku přilehlé v pravoúhlém trojúhelníku. Má periodu jednoho PI.



Obrázek 22.3: Graf funkce tangens

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 pseudo-náhodná. 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.

23.1 random() a randomSeed()

Tato funkce 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 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     delay(500);
8     Serial.println(random(256));
9 }
```

Kapitola 24

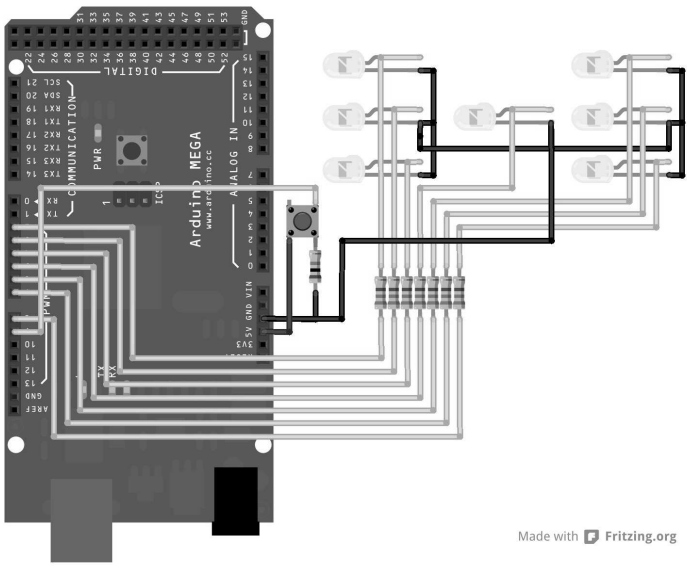
Příklad: Hrací kostka

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. Vždy po jeho zmáčknutí se vygeneruje nové číslo.

Budeme potřebovat:

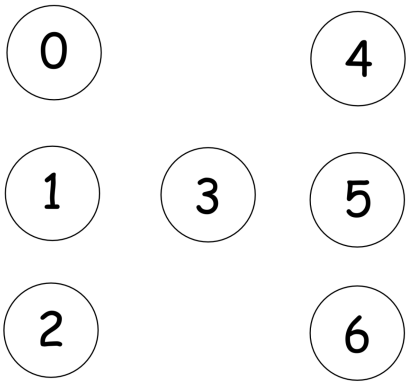
1. Nepájivé kontaktní pole s vodiči
2. 7x LED dioda
3. 7x 330 ohm resistor
4. tlačítko
5. 10 kohm resistor

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



Obrázek 24.1: Hrací kostka

Před uploadem programu do Arduina musíme upravit pole s informacemi o pinech LED diod a o tlačítku. 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.

```
1  //cisla LED s odpovidajicim indexem
2  byte leds[7] = {2,3,4,5,6,7,8};
3
4  //logicke stavy LED pouzitych u jednotlivych cisel
5  byte cisla[7][7] = {{}, /*prazdne pole - 0 se nezobrazuje*/
6                      {0,0,0,1,0,0,0}, /*1*/
7                      {1,0,0,0,0,0,1}, /*2*/
8                      {1,0,0,1,0,0,1}, /*3*/
9                      {1,0,1,0,1,0,1}, /*4*/
10                     {1,0,1,1,1,0,1}, /*5*/
11                     {1,1,1,0,1,1,1}}; /*6*/
12
13 byte tlacitko = 9; //pin s tlacitkem
14
15 byte randn; //promenna pro nahodnou hodnotu
16
17 void setup() {
18     Serial.begin(9600);
19     randomSeed(analogRead(A0)); //inicializace generatoru
20     for(int i = 0; i <= 7; 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 VII

Uživatelsky definované funkce

anglicky user defined functions

Jsou funkce, které může uživatel vytvořit sám. Jak už jsme zjistili v předchozích dílech, 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 na začátku programu. 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:

```
Ahoj Karle
TEXT TEXT
```

```
void setup() {
    Serial.begin(9600);
    zprava("Ahoj", "Karle");
    text();
}

void loop(){
}

void text(){
    Serial.println("TEXT TEXT");
}

void zprava(char a[], char b[]){
    Serial.print(a);
    Serial.print(' ');
    Serial.println(b);
}
```

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.

```
1  String slovo(){
2      return "Ahoj";
3  }
4
5  //volani
6  Serial.println(slovo()); //vypise: Ahoj
```

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

```
1  void setup() {
2      Serial.begin(9600);
3      Serial.println(secti(10,11));
4  }
5
6  void loop(){
7  }
8
9  int secti(int a, int b){
10     int soucet = a + b;
11     return soucet;
12 }
```

Pro výpočet faktoriálu čísla si můžeme sestavit vlastní funkci:

```
1  void setup() {
2      Serial.begin(9600);
3      Serial.println(fact(-2));
4  }
5
6  void loop(){
7  }
8
9  int fact(int n){
10     int vysledek;
11     if(n <= 0){
12         vysledek = 1;
13         //pro n <= se nebude nic provadet
14         //pro zaporne hodnoty neni faktorial definovan
15         //a pro nulu neni potreba nic delat
16     }
17     else{
18         vysledek = n;
19         for(int i = n-1; i > 0; i--){
20             vysledek *= i;
21         }
22     }
23     return vysledek;
24 }
```

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

```
1  void setup() {
2      Serial.begin(9600);
3      Serial.println(euler(10), 10); //Eulerovo cislo s deseti desetinnymi misty
4  }
5
6  void loop(){
7  }
8
9  float fact(float n){
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 Serial.println(byte(a)); //vypise: 255
```

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

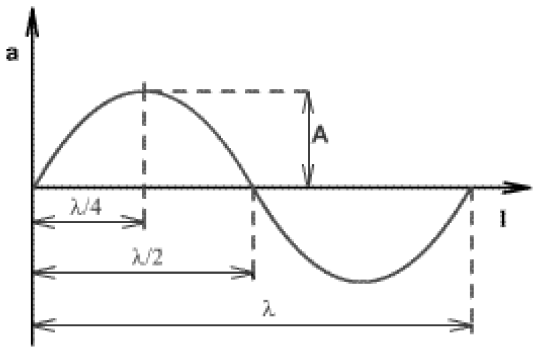
Konverze těchto typů probíhá stejně jako u těch předchozích. Rozdílem je pouze jiný rozsah výchozích hodnot.

```
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

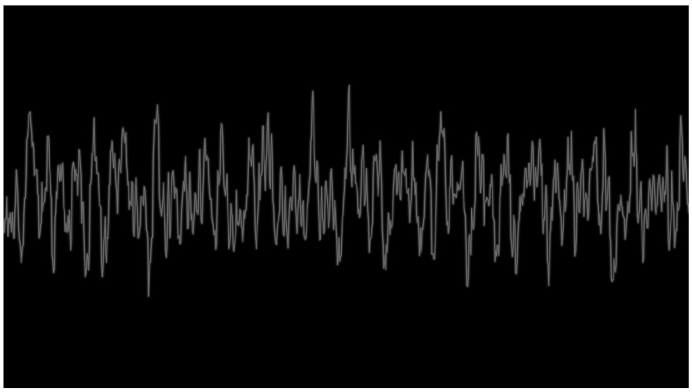
Zvuk a tón

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. Jednoduchým příkladem zvukové vlny může být například sinusoida.



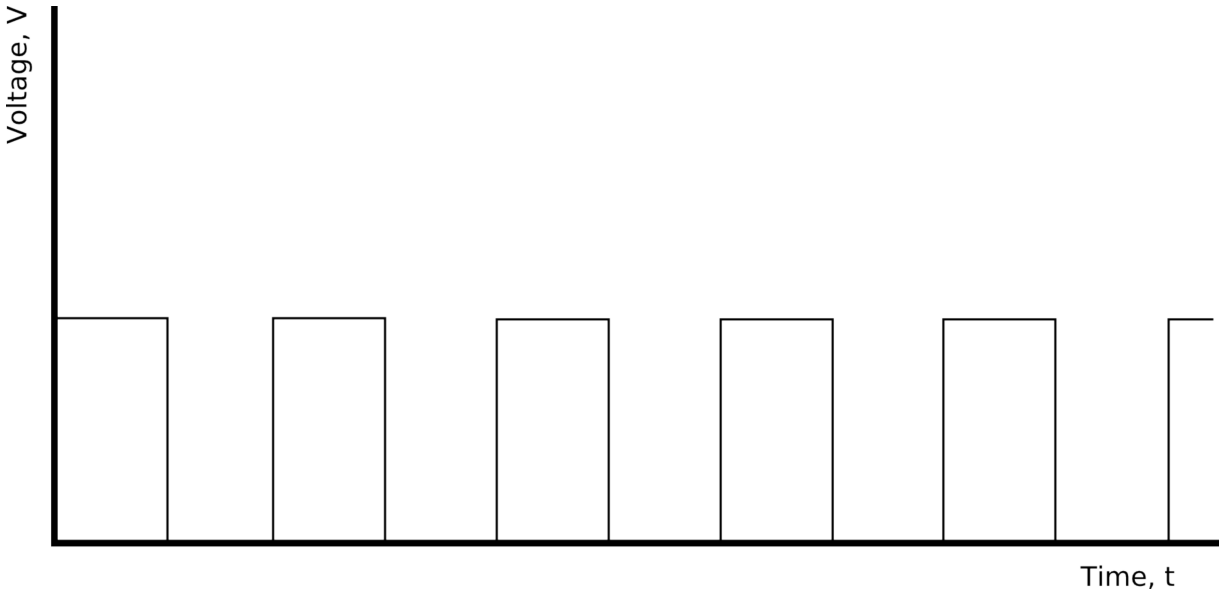
Obrázek 29.1: Graf funkce sinus

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

U Arduina je možné generovat zvuk pouze v nejjednodušší podobě. Neumožňuje totiž generovat analogové hodnoty. Rozlišuje tedy pouze 0V a 5V. Výsledná vlna nazývaná squarewave vycházející z Arduina vypadá přibližně takto:



Obrázek 29.3: Graf Squarewave

Výška tónu závisí na frekvenci, to je počet opakování „hřebenů“ vlny za jednu sekundu, což vztaženo na Arduino znamená počet změn z 0 na 5V 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.

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.

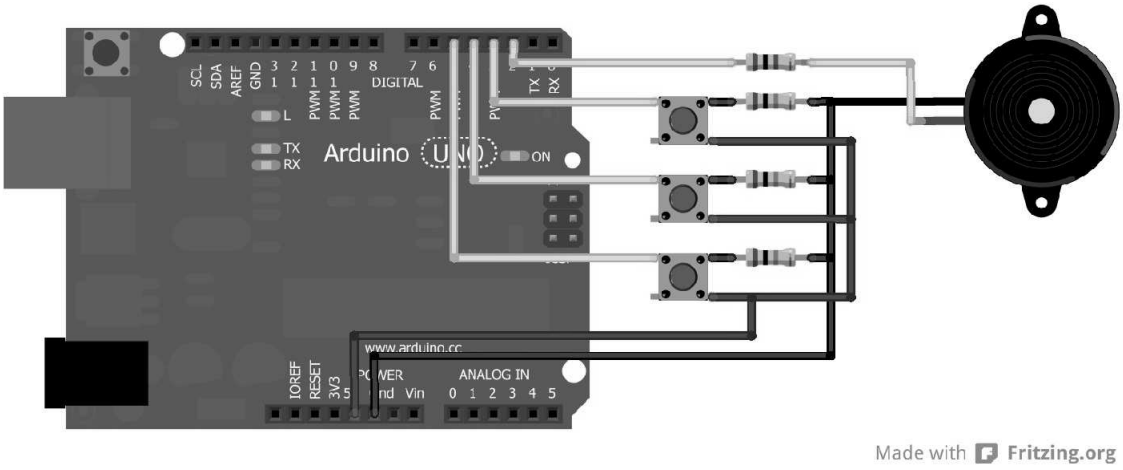
29.2 noTone()

Touto funkcí se vypne generovaný tón 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. 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. 3x tlačítko
5. 3x 10 kohm resistor
6. 1x 100 kohm resistor (doporučený k Piezo)



Obrázek 29.4: Buzzer s tlačítky

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

```
1  #include "pitches.h"
2
3  void setup() {
4  }
5
6  void loop() {
7      if(digitalRead(5) == 1){
8          tone(2, NOTE_A4, 200);
9      }
10     if(digitalRead(4) == 1){
11         tone(2, NOTE_C5, 200);
12     }
13     if(digitalRead(3) == 1){
14         tone(2, NOTE_E5, 200);
15     }
16 }
```

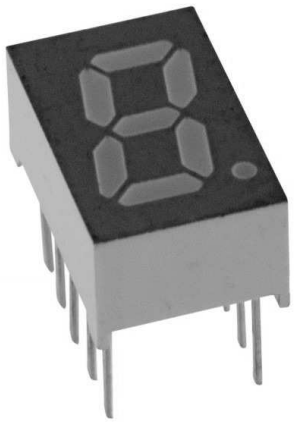
Kapitola 30

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 několik LED diod zalitých v jednom pouzdře, které dohromady vytvářejí znaky. Často mají tyto LED diody společnou jednu z nožiček. Podle typu je to buďto anoda, nebo katoda (typ lze vyčíst v dokumentaci daného displeje). Nejčastějším typem je sedmisegmentový displej, který jistě všichni známe.

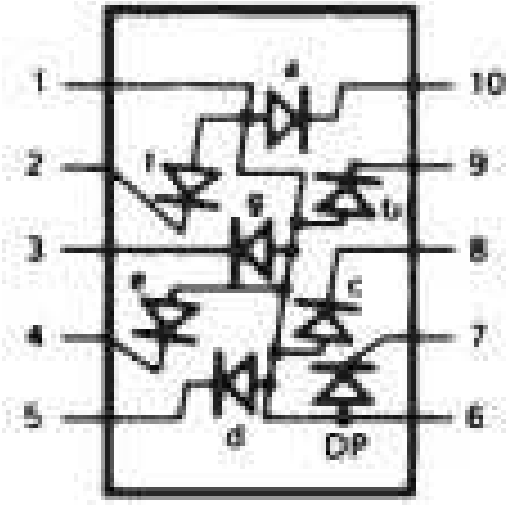
30.1 Sedmisegmentový displej

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.



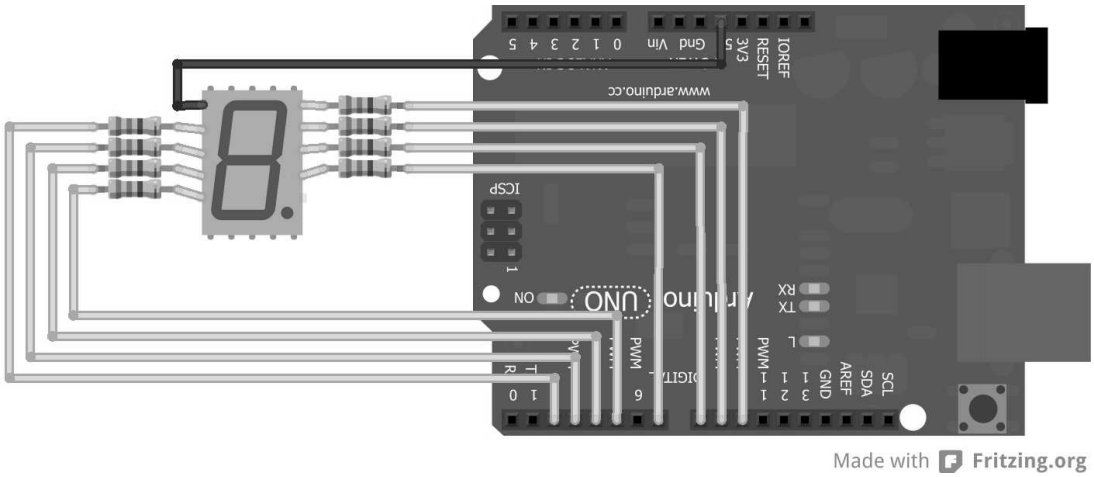
Obrázek 30.1: Sedmisegmentový displej

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



Obrázek 30.2: Vnitřní schéma sedmisegmentového 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.



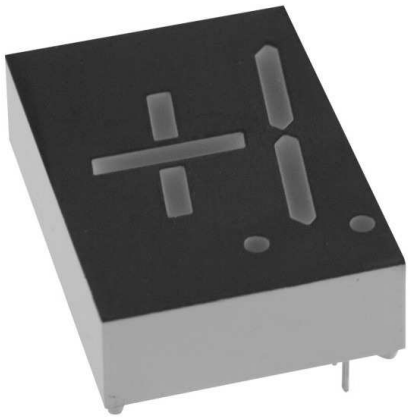
Obrázek 30.3: Zapojení sedmisegmentového displeje

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

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

void loop() {
}
```

30.2 Vícesegmentové displeje

Segmentových displejů existuje celá řada. Může se jednat o displeje schopné zobrazit pouze znak 1, 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 30.4: Dvousegmentový displej

Obrázek 30.5: 16 segmentový displej

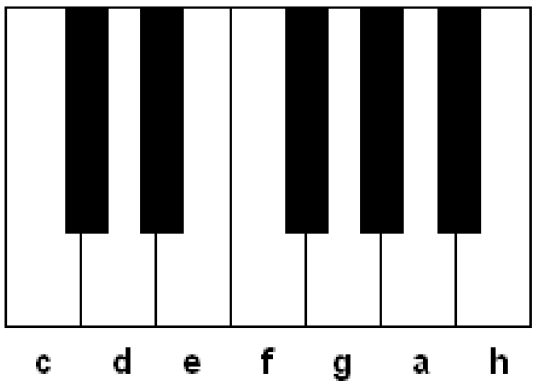
Kapitola 31

Příklad: Klavír

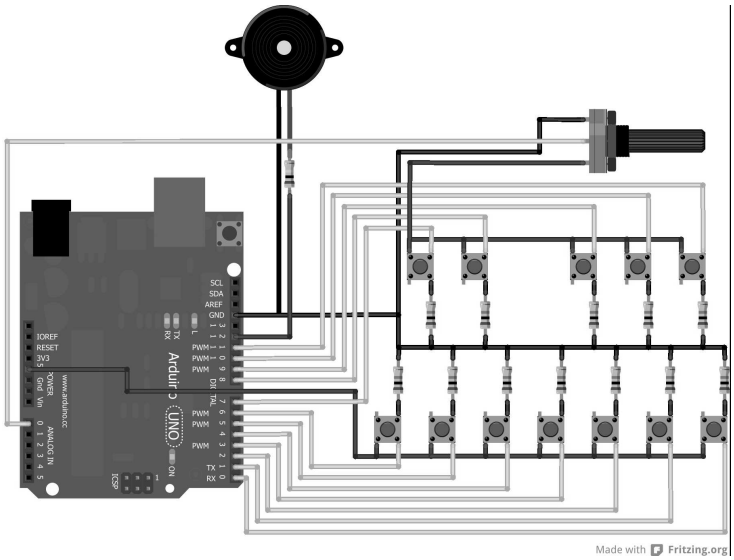
V tomto příkladu si ukážeme, jak vytvořit jednoduchý klavír s možností volby mezi oktávami pomocí potenciometru. Číslo oktávy si necháme posílat pomocí sériové linky. Budeme potřebovat:

1. Arduino
2. Piezzo reproduktor
3. Nepájivé kontaktní pole s vodiči
4. 12x tlačítko
5. 12x 10 kohm resistor
6. 1x 100 kohm resistor (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.

```
#include "pitches.h";
byte oktava;
byte piezo = 12;

byte klavesy[12] = {6,7,5,8,4,3,9,2,10,1,11,0}; //piny jednotlivych tlacitek zleva doprava
//tony v jednotlivych oktavach
int oktavy[4][12] =
{{NOTE_C3, NOTE_CS3, NOTE_D3, NOTE_DS3, NOTE_E3, NOTE_F3,
  NOTE_FS3, NOTE_G3, NOTE_GS3, NOTE_A3, NOTE_AS3, NOTE_B3},
{NOTE_C4, NOTE_CS4, NOTE_D4, NOTE_DS4, NOTE_E4, NOTE_F4,
 NOTE_FS4, NOTE_G4, NOTE_GS4, NOTE_A4, NOTE_AS4, NOTE_B4},
{NOTE_C5, NOTE_CS5, NOTE_D5, NOTE_DS5, NOTE_E5, NOTE_F5,
 NOTE_FS5, NOTE_G5, NOTE_GS5, NOTE_A5, NOTE_AS5, NOTE_B5},
{NOTE_C6, NOTE_CS6, NOTE_D6, NOTE_DS6, NOTE_E6, NOTE_F6,
 NOTE_FS6, NOTE_G6, NOTE_GS6, NOTE_A6, NOTE_AS6, NOTE_B6}};

void setup() {
    tone(piezo, 440, 500);
}

void loop() {
    oktava = map(analogRead(A0),0,1023,0,3);
    for(int i = 0; i < 12; i++){
        if(digitalRead(klavesy[i]) == HIGH){
            tone(piezo, oktavy[oktava][i], 100);
        }
    }
}
```

Pokud se nedaří nahrát program do Arduina, odpojte napájení desky tlačítek. Po uploadu programu je opět připojte.

Část VIII

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

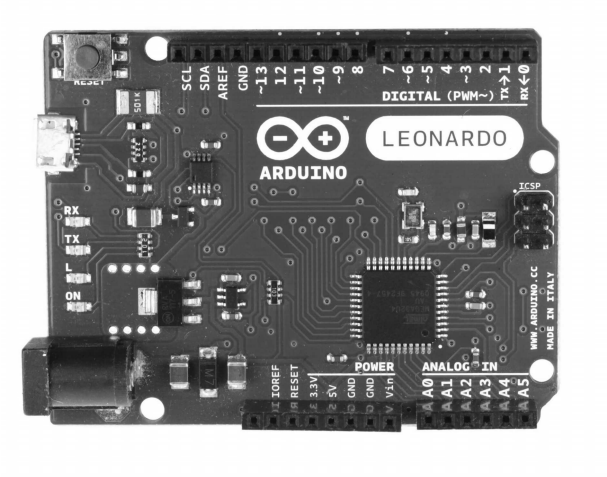
V první kapitole jsem se zmínil o šesti Arduinech, jejichž mozkem je procesor ATmega32u4. Byly to desky Arduino Micro, Lilypad Arduino, Arduino Leonardo, Arduino Yún, Arduino Esplora a Arduino Robot. Jejich největší výhodou je to, že ke komunikaci s PC nepotřebují žádný převodník. Použitý čip má totiž serial-USB převodník přímo v sobě. Díky absenci tohoto prostředníka je daleko jednodušší naprogramovat Arduino tak, aby se vydávalo za myš či klávesnici (nebo oboje dohromady). V dnešním článku se budeme zabývat hlavně deskami Arduino Leonardo a Arduino Esplora. Popsané postupy jsou však většinou použitelné u všech 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 desek jsou většinou 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[EN] – vše však na vlastní nebezpečí.

Kapitola 33

Arduino Leonardo



Obrázek 33.1: Arduino Leonardo

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

Může se stát, 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. Může se tedy stát, že Arduino 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.

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     Keyboard.begin();
3     //prikazy mysi
4     Keyboard.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 10, 9, 8. Tyto tlačítka budou fungovat 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, tedy 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 z leva doprava můžeme tedy využít předchozí zapojení a kód:

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

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 teftbfMouse.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. 7x tlačítko
4. 7x 10 kohm resistor

```
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};
int hodnotySmeru[4][2] = {{0,−1},{0,1},{1,0},{−1,0}};
//nahoru, dolu, doprava, doleva
byte i;
int limitPohyb = 2;
//jak casto se bude pohybovat mysi
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);
    }
}

void loop() {
    for(i = 0; i <= 2; i++){
        if(digitalRead(pinyTlacitka[i]) != posledniStavTlacitek[i]){
            if(digitalRead(pinyTlacitka[i]) == HIGH){
                Mouse.press(tlacitka[i]);
            }
            else{
                Mouse.release(tlacitka[i]);
            }
            posledniStavTlacitek[i] = !(posledniStavTlacitek[i]);
        }
    }

    if(posledniPohyb + limitPohyb < millis()){
        for(i = 0; i <=3; i++){
            if(digitalRead(smery[i]) == HIGH){
                Mouse.move(hodnotySmeru[i][0],hodnotySmeru[i][1],0);
            }
        }
        posledniPohyb = millis();
    }
}
```

Kapitola 35

Keyboard

Jak už název napovídá, druhá část funkcí slouží k odesílání příkazů ve formě úhozů kláves. Jak jistě poznáme, 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. 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 }
```

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. 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.

```
1  byte pinyTlacitek[3] = {11,10,9}; //doleva, nahoru, doprava
2  byte posledniStavTlacitek[3] = {LOW,LOW,LOW};
3  byte tlacitka[3] = {KEY_LEFT_ARROW, 32, KEY_RIGHT_ARROW};
4  byte i;
5
6  void setup() {
7      Keyboard.begin();
8      for(i = 0; i < 3; i++){
9          pinMode(pinyTlacitek[i], INPUT);
10     }
11 }
12
13 void loop() {
14     for(i = 0; i < 3; i++){
15         if(digitalRead(pinyTlacitek[i]) != posledniStavTlacitek[i]){
16             //abychom PC nezahltili,posleme prikaz
17             //pouze kdyz se zmeni stav tlacitka
18             if(digitalRead(pinyTlacitek[i]) == HIGH){
19                 Keyboard.press(tlacitka[i]);
20             }
21             else{
22                 Keyboard.release(tlacitka[i]);
23             }
24             //zmena stavu tlacitka v poli pomoci negace
25             posledniStavTlacitek[i] = !(posledniStavTlacitek[i]);
26         }
27     }
28 }
```

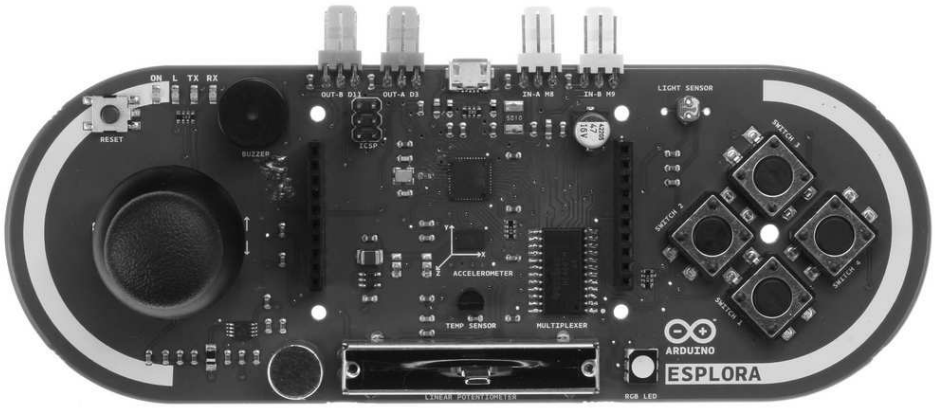
35.3 Keyboard.print() a Keyboard.println()

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, jen s kratším zápisem může vypadat takto:

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

Kapitola 36

Arduino Esplora



Obrázek 36.1: Arduino Esplora

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 pracování s jednotlivými částmi desky.

36.1 Joystick

Joystick jistě všichni znáte z různých herních konzolí. Jedná se o zařízení, které 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 čtyřech 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:

```
#include <Esplora.h>

byte sipky[] = {SWITCH_UP, SWITCH_DOWN, SWITCH_LEFT, SWITCH_RIGHT};
byte zkratky[] = {KEY_UP_ARROW, KEY_DOWN_ARROW, KEY_LEFT_ARROW,
    KEY_RIGHT_ARROW};
byte posledniStav[] = {1,1,1,1};
byte i;

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

void loop(){
    for(i = 0; i < 4; i++){
        if(Esplora.readButton(sipky[i]) != posledniStav[i]){
            if(Esplora.readButton(sipky[i]) == LOW){
                Keyboard.press(zkratky[i]);
            }
            else{
                Keyboard.release(zkratky[i]);
            }
            posledniStav[i] = !(posledniStav[i]);
        }
    }
}
```

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

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.

```
#include <Esplora.h>

int hodnota;

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

void loop(){
    hodnota = Esplora.readSlider();
    Serial.println(hodnota);
    delay(500);
}
```

36.4 Mikrofon

Dalším přídavným hardwarem je standartní 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 1 kHz (kolikrát za sekundu dojde k měření). Pokud bychom takovouto vlnu přehráli, bude velmi zdeformovaná. Pro lidské ucho se totiž doporučuje vzorkovací frekvence 40kHz a vyšší. Spousta zvukových informací 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 fotoresistor. Ke čtení hodnot se využívá funkce Esplora.readLightSensor(), která vrací hodnoty od 0 (pro tmu) do 1023 (pro světlo). Příklad práce s měřením světla je stejný jako u mikrofonu, jen se zaměněním funkcí pro získání hodnot, čili:

```
1  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 $°C = (°F - 32) * (5/9)$. 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.

36.7 Akcelerometr

Akcelerometr je součástka schopná měřit zrychlení. U Arduina Esplora jej nalezneme ve verzi, která je schopná detekovat zrychlení ve třech kolmých osách (x, y, z). Nalezneme ho uprostřed desky. Na zařízení působí zrychlení, i když je v klidu. 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: ");
    Serial.println(zAxis);

    delay(500);
}
```

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 celkově generovat až 2563 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 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. Zde nás budou zajímat hnědě vyznačené piny. Pod jejich číslem je můžeme používat pomocí standartních 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 včetně doporučeného zapojení dalších přídavných komponent.

Část IX

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. 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.

Kapitola 38

Seznámení

Po rozbalení staženého .zip archivu je prostředí plně funkční. Spustíme jej 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): 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() {}
```

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};
```

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()**.

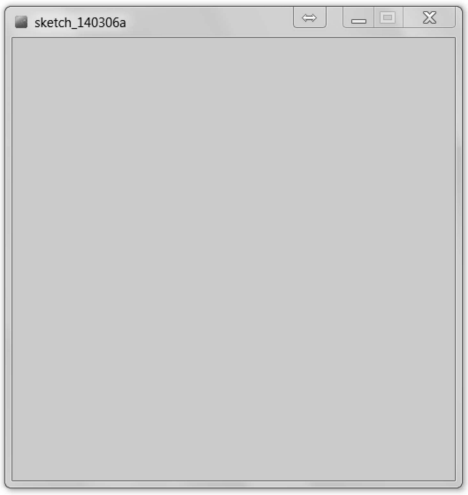
Kapitola 40

Kreslení

40.1 Vytvoření kreslícího plátna

Abychom měli kam kreslit, musíme pro program vytvořit kreslící plochu. To se provádí pomocí příkazu **size(x,y)**. Je jasné, že x a y jsou rozměry v jednotlivých osách. Následující kód tedy vytvoří kreslící plátno o rozměrech 500x500. 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 }
```



Obrázek 40.1: Hlavní okno programu v Processing

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.

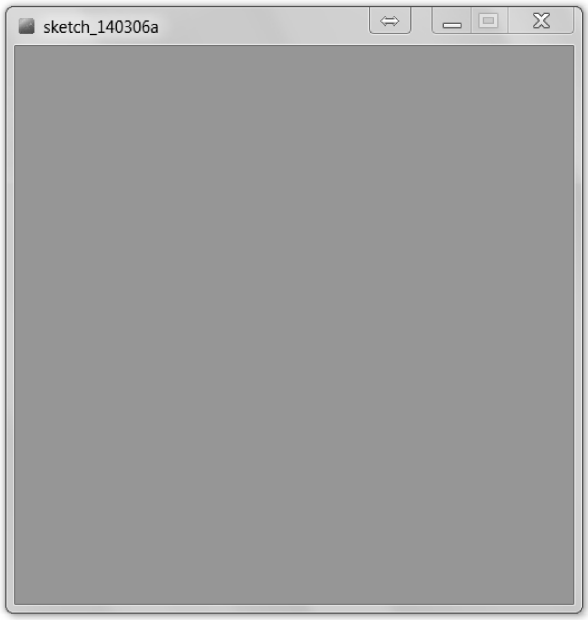
40.2 Barvy

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 kreslící 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(){}
```



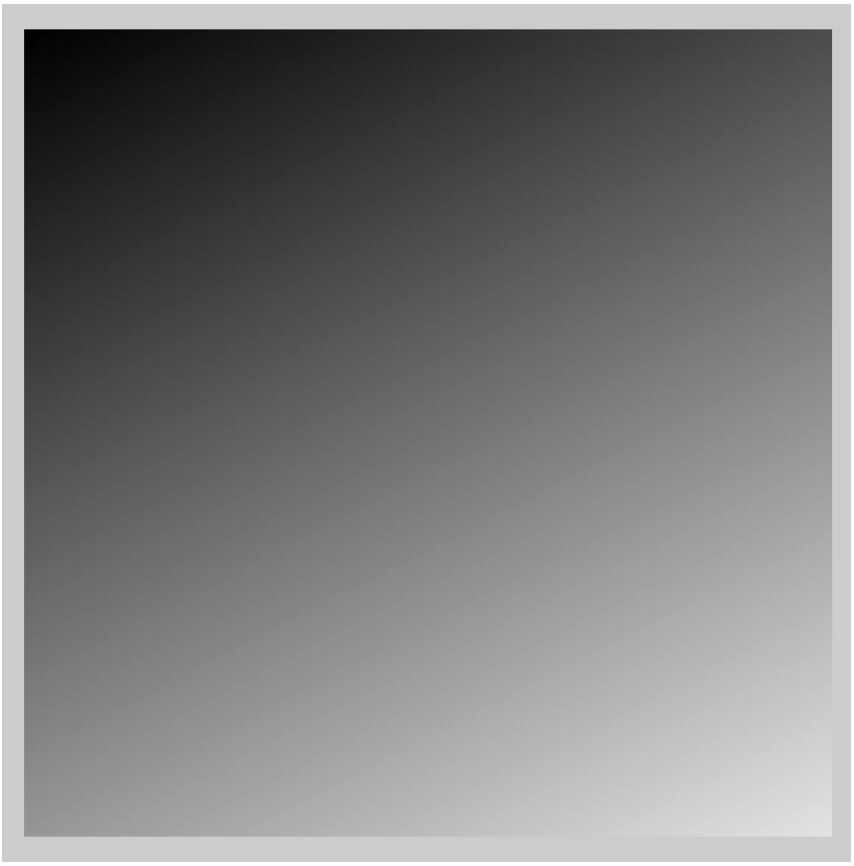
Obrázek 40.2: Funkce background()

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.

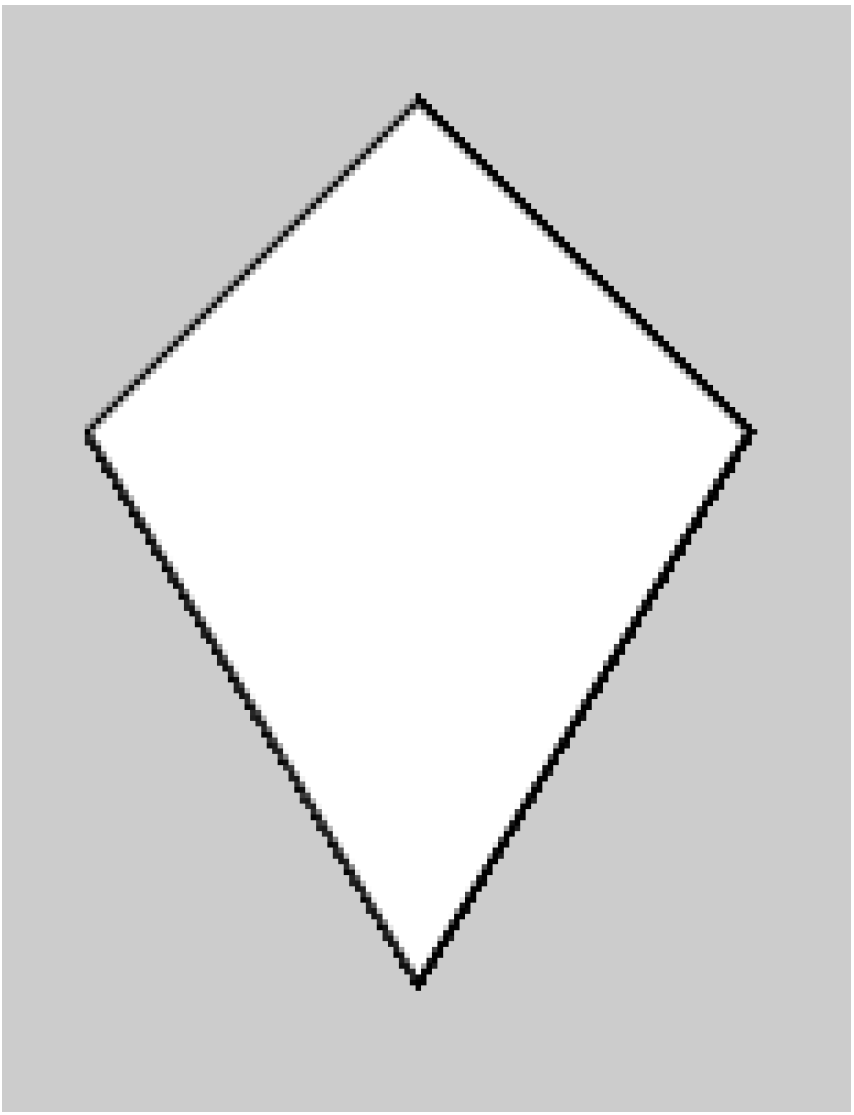
```
1  int delka = 100;
2  int[] pocatek = {20,20}; //pocatecni souradnice x,y
3
4  void setup(){
5      size(296,140);
6  }
7
8  void draw(){
9      for(int i = 0; i<256; i++){
10         stroke(i);
11         line(pocatek[0] + i, pocatek[1], pocatek[0] + i, pocatek[1] + delka);
12     }
13 }
```

Obrázek 40.4: Funkce stroke() a fill()

40.5 Čtyřůhelník

Čtyřúhelník je geometrický útvar daný pomocí čtyř bodů. V Processingu 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 }
```

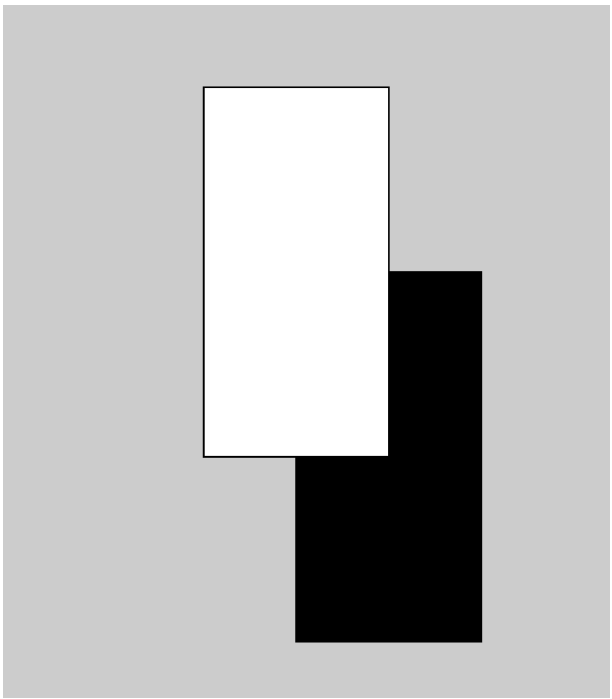


Obrázek 40.5: Funkce quad()

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. Defaultně 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.

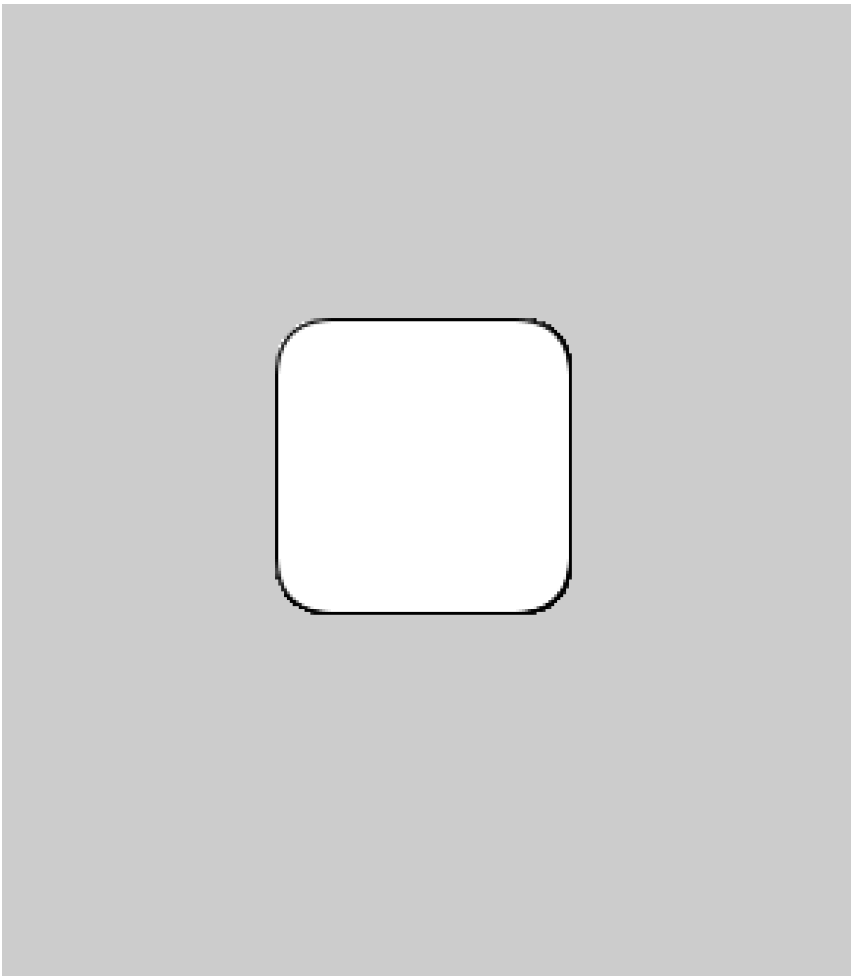
```
1  void setup(){
2      size(500,500);
3  }
4
5  void draw(){
6      fill(0);
7      rectMode(CORNER);
8      rect(250,250,100,200);
9
10     fill(255);
11     rectMode(CENTER);
12     rect(250,250,100,200);
13 }
```



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.

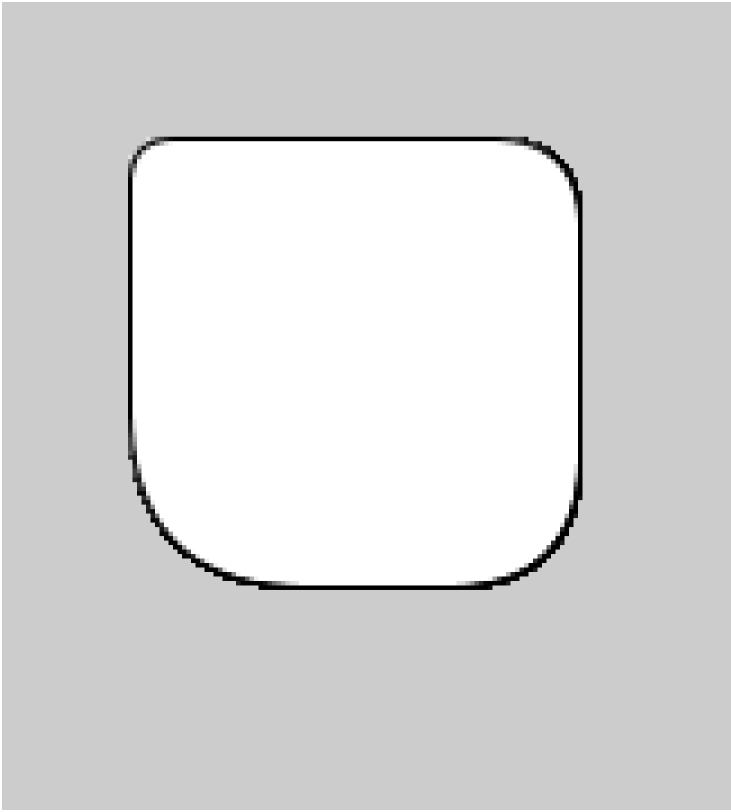
```
1  void setup(){
2      size(500,500);
3  }
4
5  void draw(){
6      fill(255);
7      rectMode(CENTER);
8      rect(250, 250, 100, 100, 20);
9  }
```



Obrázek 40.7: Funkce rect()

U čtyřúhelníka 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  }
```

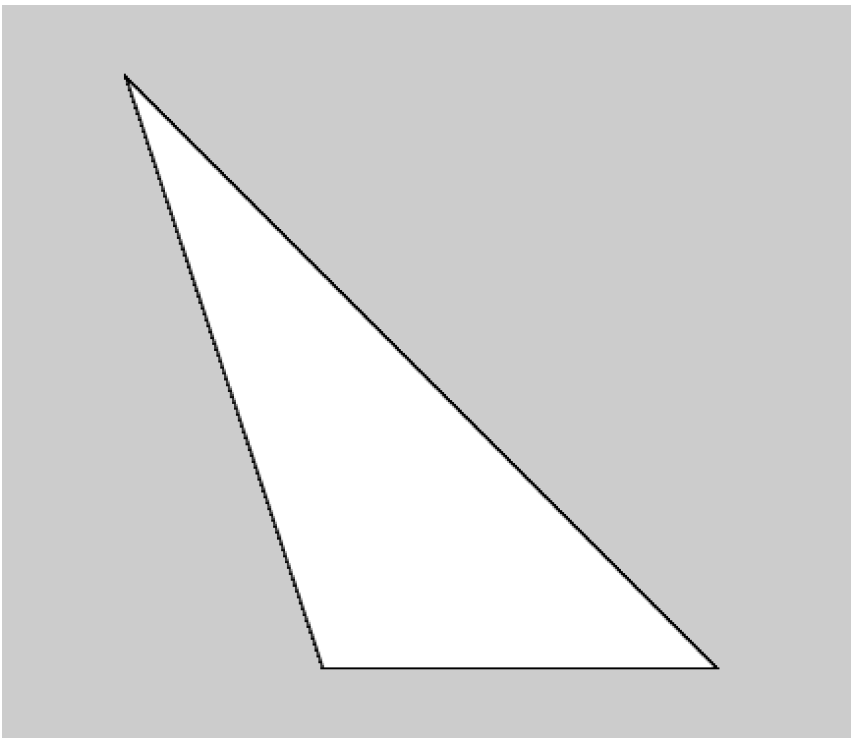


Obrázek 40.8: Funkce rect()

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  }
```

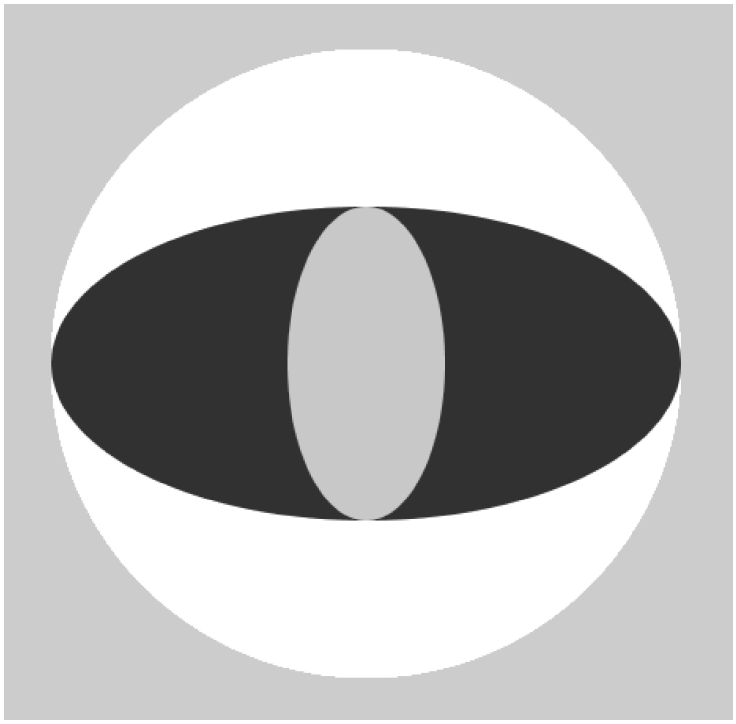


Obrázek 40.9: Funkce triangle()

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). X a y jsou souřadnice středu elipsy.

```
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 }
```

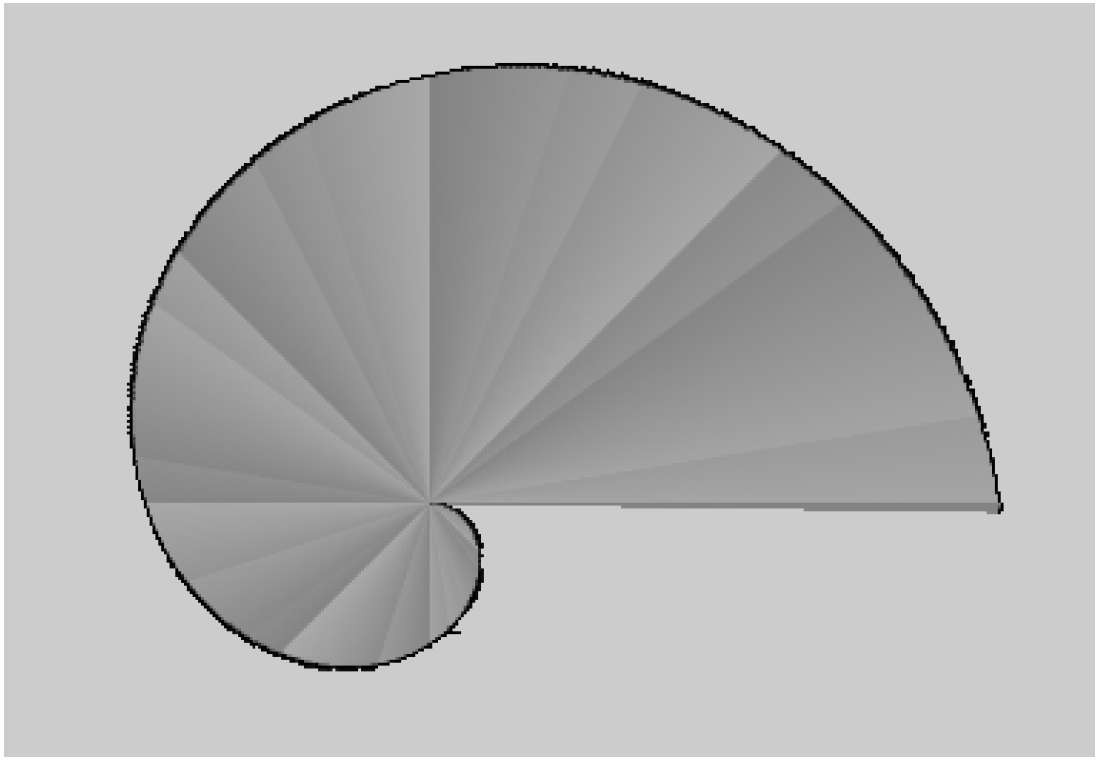


Obrázek 40.10: Funkce ellipse()

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 PI.

```
1  int casti = 400;
2
3  void setup(){
4      size(500,500);
5  }
6
7  void draw(){
8      for(int i=1; i<=casti; i++){
9          fill(color(i%30,i%50+200,i%40+100));
10         arc(250,250,400/casti*i, 400/casti*i, PI*2*i/casti, PI*2*i/casti+PI*2/casti);
11     }
12 }
```



Obrázek 40.11: Funkce arc()

V referenci jazyka Processing nalezneme i funkce pro práci s křivkami (Curves). Těmi se ale dnes 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.

41.1 Tlačítka myši

Základní funkce, která zjišťuje, jestli bylo vůbec nějaké tlačítko stisknuto je funkce **mouseClicked()**. Jak už jsem naznačil, 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).

```
1 void setup(){}
2 void draw(){ }
3
4 void mouseClicked(){
5     //kod, ktery se provede kdyz je stisknute libovolne tlacitko mysi
6 }
```

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.

```
1  int casti = 400;
2  int vypln = 0;
3
4  void setup(){
5      size(500,500);
6  }
7
8  void draw(){
9      for(int i=1; i<=casti; i++){
10         fill(vypln);
11         noStroke();
12         arc(250,250,400/casti*i, 400/casti*i, PI*2*i/casti, PI*2*i/casti+PI*2/casti);
13     }
14 }
15
16 void mouseClicked(){
17     if(mouseButton == LEFT){
18         vypln = 255;
19     }
20     else{
21         vypln = 0;
22     }
23 }
```

Nalezneme zde také další funkce, které s tlačítky myši souvisejí. 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()**, který se vykoná, pokud není zmáčknuto žádné tlačítko a myš se hýbe. funkce **mousePressed()** se provede, pokud je stisknuto libovolné tlačítko a **mouseRelased()** 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 souřadnice v osách x a y. Jsou to **mouseX** a **mouseY**. Ukažme si program, který vykreslí úsečku danou jedním pevným bodem a kurzorem myši.

```
1  void setup(){
2      size(500,500);
3  }
4
5  void draw(){
6      background(200);
7      line(250,250,mouseX,mouseY);
8  }
```

To by bylo pro dnešek vše. Možná si říkáte, proč se v seriálu 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 X

Arduino a Processsing

V minulém díle jsme si představili prostředí Processing. I dnes se jím budeme zabývat, ale ukážeme si, jak může Arduino s tímto prostředím komunikovat. Na příkladech si ukážeme 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 komunikace a 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, avšak za cenu omezení funkcí, které umí. 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 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 zkoumá všechny digitální i analogové piny. Pokud je na nějakém hodnota HIGH, vybarví se jemu odpovídající čtvereček. U analogových pinů zkoumá jejich 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 }
```

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 jej vrací 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 57600 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  }
```

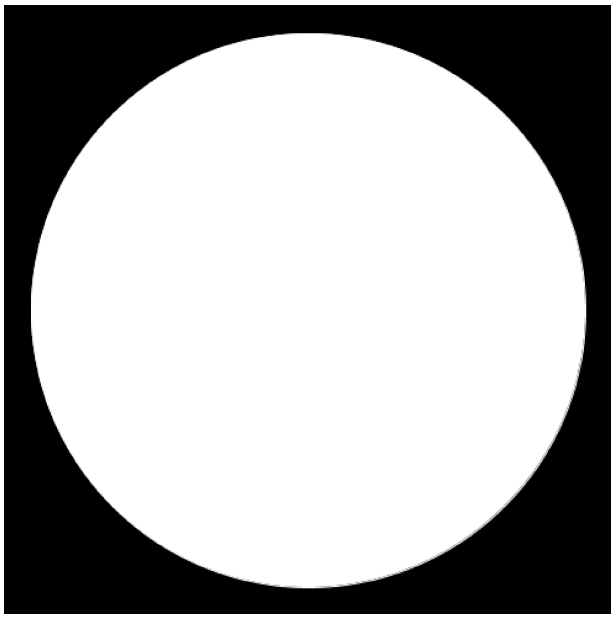
Nyní už máme Arduino i Processing připravené ke spolupráci a můžeme se pustit do programování. Sami zjistíte, že dostupné funkce jsou téměř totožné s těmi, které už známe.

43.3 Programování

Knihovna arduino s sebou přináší několik konstant. Patří mezi ně: **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, u nás tedy 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.

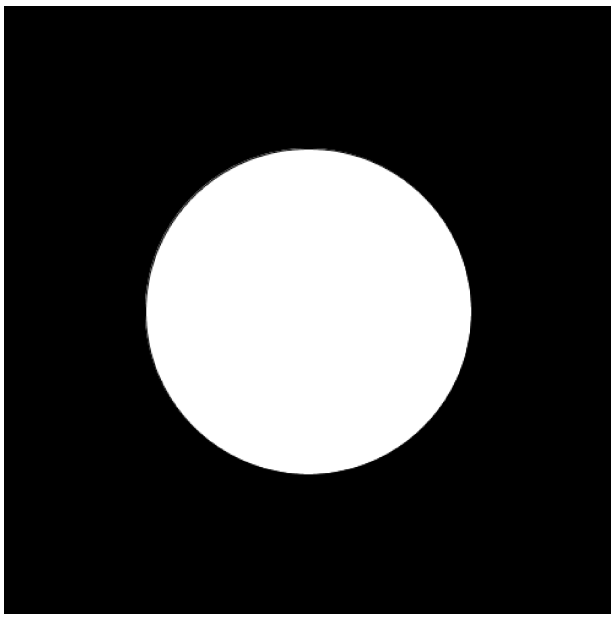
```
1  import processing.serial.*;
2  import cc.arduino.*;
3
4  //vytvoreni objektu mojeArduino
5  Arduino mojeArduino;
6  int led = 13;
7  int tlacitko = 2;
8
9  void setup() {
10     size(500,500);
11
12     println(Arduino.list()); //vypise dostupna zarizeni
13     //vytvoreni noveho objektu
14     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 A0 a podle toho rozsvěcovat LED diodu 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 jsem nezmínil funkci servoWrite(). Servomotory jsme se totiž ještě nezabývali, takže se jí nebudeme zdržovat.

Kapitola 44

Vlastní způsob komunikace

Jak už jsem řekl 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 budu používat Arduino Esplora. 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 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)**.

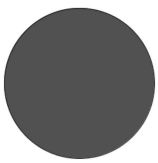
```
1  import processing.serial.*;
2
3  Serial port;
4
5  void setup() {
6      size(500,500);
7
8      println(Serial.list());
9      port = new Serial(this, Serial.list()[0], 57600);
10 }
11
12 void draw() {
13
14 }
```

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:

```
#include <Esplora.h>

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

void loop(){
    Serial.println(Esplora.readSlider());
    delay(100);
}
```

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){
            background(255);
            //funkce trim() orizne prebytecne 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 dostáváme se k odesílání informací 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. V první ukázce Processing odešle Arduinu Esplora hodnotu červené barvy, podle které se poté rozsvítí LED na desce.

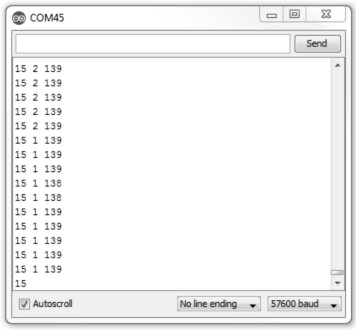
```
1  //Arduino program
2
3  #include <Esplora.h>
4
5  void setup(){
6      Serial.begin(57600);
7  }
8
9  void loop(){
10     if(Serial.available() > 0){
11         Esplora.writeRGB(Serial.read(),0,0);
12     }
13 }
```

```
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 Wiring 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.

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

Kapitola 45

Příklad: Ovládání bodu joystickem

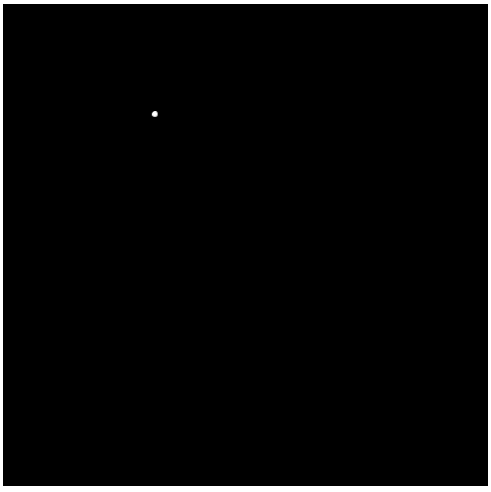
V prvním příkladu si ukážeme, jak se dá navigovat po plátně pomocí joysticku u Esplory. Budeme potřebovat:

- 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 tečku (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;
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);
24          y = port.readStringUntil(eol);
25          if(x != null){
26               if(y != null){
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

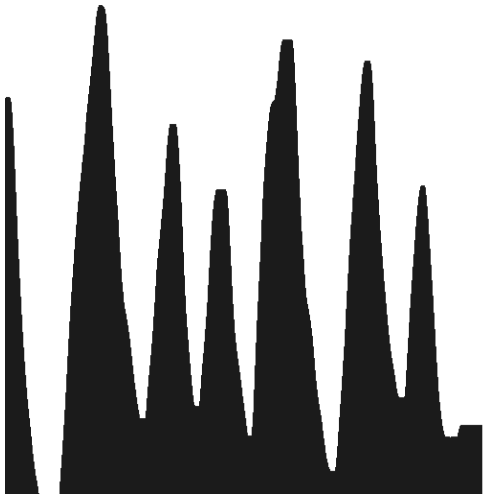
Kapitola 46

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 XI

Propojujeme Arduino s jinými zařízeními

V dnešním článku 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. Dnes 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 (receive = přijmout) a Tx (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 jsem zmínil 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 napájená 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. Pro komunikaci přes USB využijeme funkci začínající Serial., pro hardware komunikaci (náš případ) využijeme Serial1..

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 jsem napsal před chvílí – zapojení je velmi jednoduché. Rx jedné desky připojíme na Tx druhé a GND a 5V obou desek připojíme k sobě.

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".

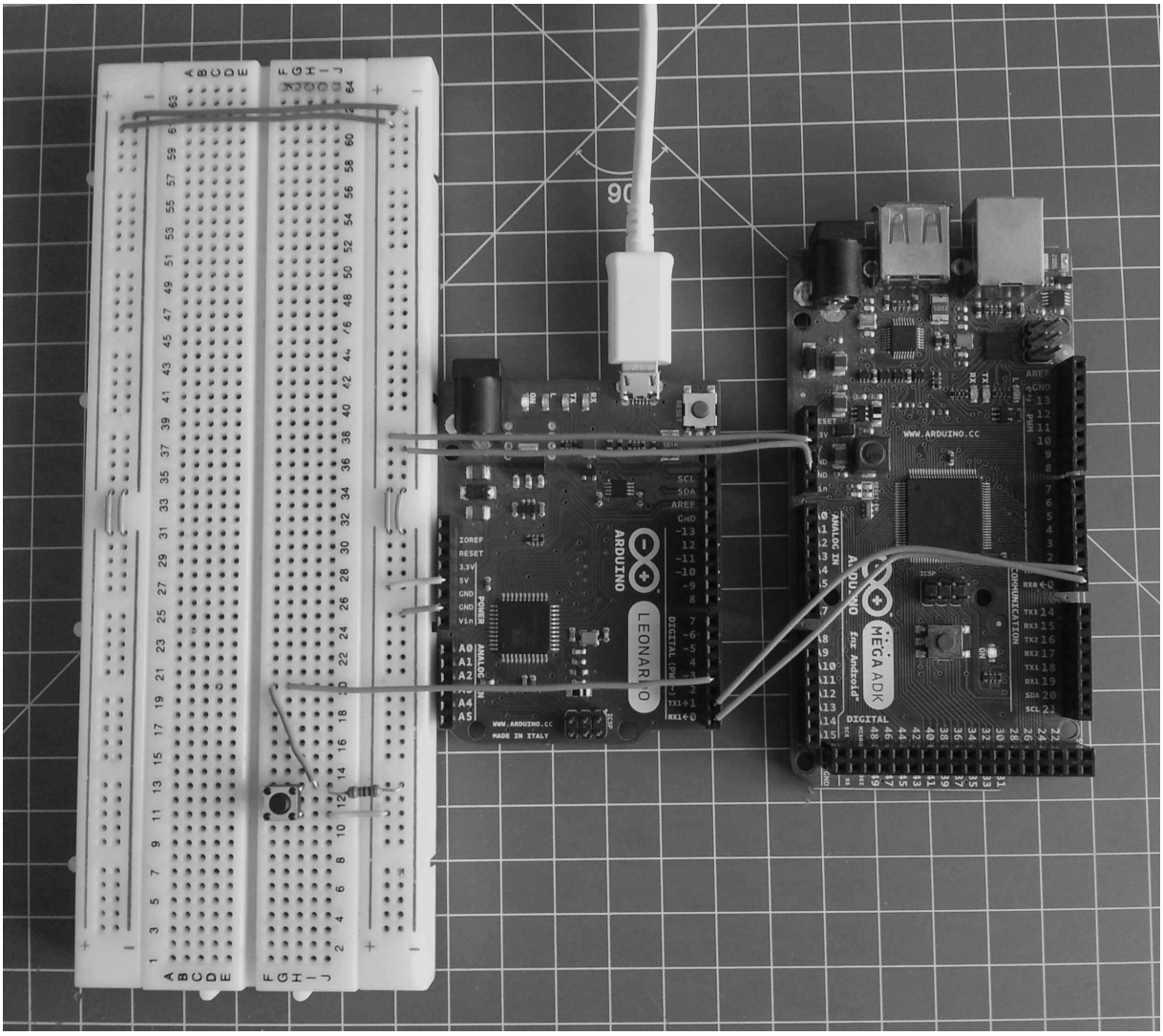
```
byte tl = 3; //tlacitko

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 BT modulem.

Na Arduino Esplora piny Rx a Tx bohužel nenajdete.



Obrázek 47.1: Propojení Arduina Leonardo s Arduinem Mega

Kapitola 48

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 5V 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.

Obrázek 48.1: BlueSMiRF Silver

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.

48.1 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 115200 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 BT 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. Stačí ale na dobu nahrávání odpojit modul od Arduina, čímž by se měl problém vyřešit.

Kapitola 49

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.

Obrázek 49.1: SensoDuino

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. Tyto 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.begin(9600);
15 }
16
17 void loop(){
18     while(Serial.available() > 0){
19         a = Serial.parseInt(); //typ senzoru
20         b = Serial.parseInt(); //cislo mereni
21         c = Serial.parseFloat(); //prvni hodnota
22         d = Serial.parseFloat(); //druha hodnota
23         e = Serial.parseFloat(); //treti hodnota
24
25         Serial.println(a);
26         Serial.println(b);
27         Serial.println(c);
28         Serial.println(d);
29         Serial.println(e);
30     }
31 }
```

To není vskutku žádný div. Je to ale vše, co potřebujeme k dalšímu programování. Ještě si ale musíme uvést, jaké označení mají jaké senzory (typ senzoru). 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 rychlost připojení, jméno, nebo PIN se používají tzv. AT commands. Ty ale pro základní činnost nejsou potřeba. Jejich výpis naleznete například zde.

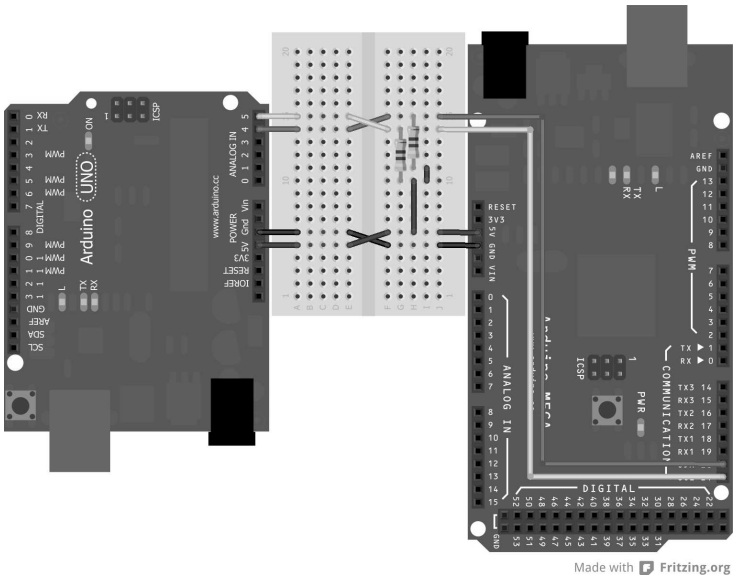
Kapitola 50

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 7-bitovou adresu). Vodič, který udává takt komunikace, se nazývá SCL (clock line). Druhý vodič přenášející data je označován SDA (data line). U tohoto typu komunikace je vždy jedno zařízení řídící (master) a ostatní jsou řízená (slave). Zařízení master nemusí mít definovanou adresu. 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 asi 4700 Ohm. Jeden zapojíme mezi SDA a +5V a druhý mezi SCL a +5V.



Obrázek 50.1: Zapojení i2c sběrnice

Tím máme za sebou zapojení obou desek. Nyní ž se můžeme podívat na software.

50.1 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.

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)	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 funkce, která je volána, pokud jsou od zařízení master přijata nějaká data. Volaná funce 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.

50.2 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     }
22
23     a = ' ';
24     delay(500);
25 }
```

```
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 }
```

50.3 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. 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.

```
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 }
```

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

Stejně jako u sériových modulů, i zde existuje celá řada součástek, vybavená možností komunikovat přes i2c. Patří mě například různé typy pamětí, převodníků, řadičů ale i displejů. Tím se nám otevírá celá škála dalších možností, co s Arduinem dělat.

Část XII

Arduino a displeje

Ve článcích o displejích si ukážeme různé způsoby grafického výstupu z Arduina. V prvním dílu 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 51

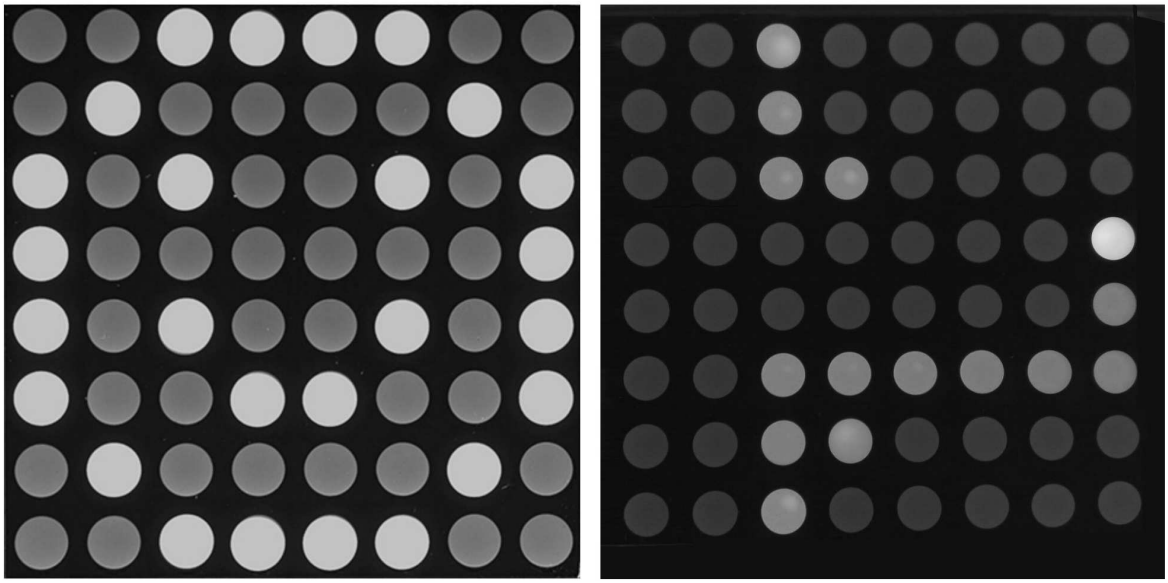
Úvod

U velké části projektů se dostaneme do situace, kdy se nám hodí mít možnost něco zobrazit. Ať už jde o vykreslování grafu, 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 diody.

Kapitola 52

Maticové LED displeje

anglicky matrix display

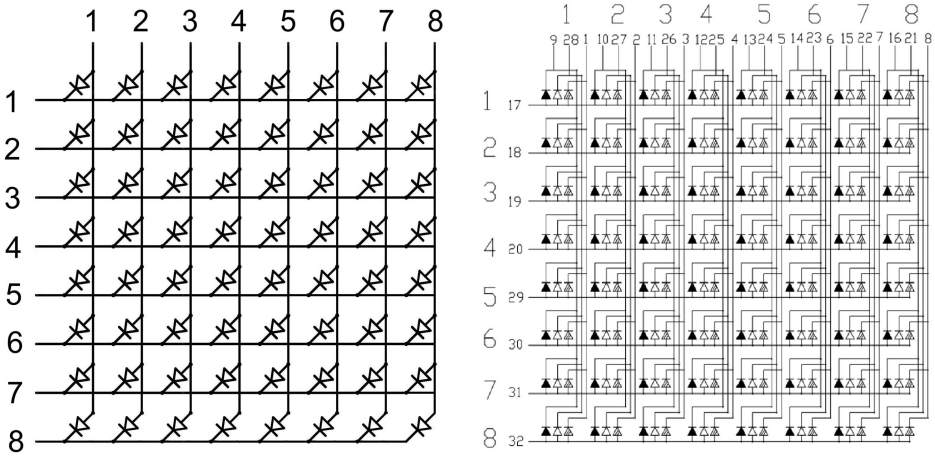


Obrázek 52.1: Maticové displeje

Tento typ displejů využívá k zobrazování klasické LED diody. 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 5x7, 8x8 a další.

52.1 Teorie řízení

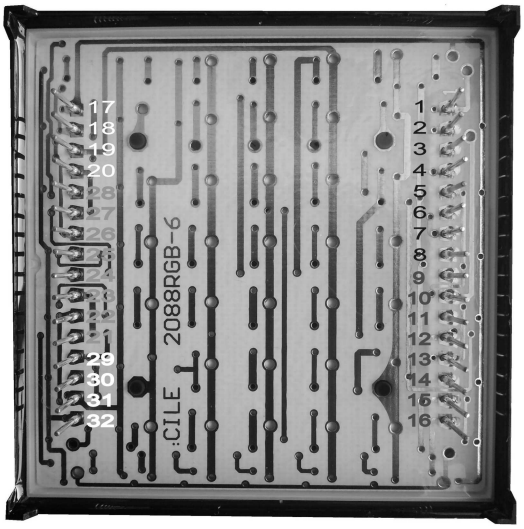
Schéma jednobarevného a trojbarevného maticového displeje. Rozložení LED je stejné, jako při pohledu na displej zepředu.



Obrázek 52.2: Schéma mono a RGB maticového displeje

V tomto článku budeme používat tento RGB maticový displej, jehož schéma můžete vidět na obrázku vpravo. 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. Všechny anody jsou spojené v řádcích. 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 datasheetu.



Obrázek 52.3: 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.

52.2 Zapojení

Nyní už se můžeme pustit do spojení Arduina s displejem. V této části pracuji s Arduinem Leonardo. Budeme používat zelenou barvu RGB displeje. Také bychom měli použít 330 ohm resistor 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

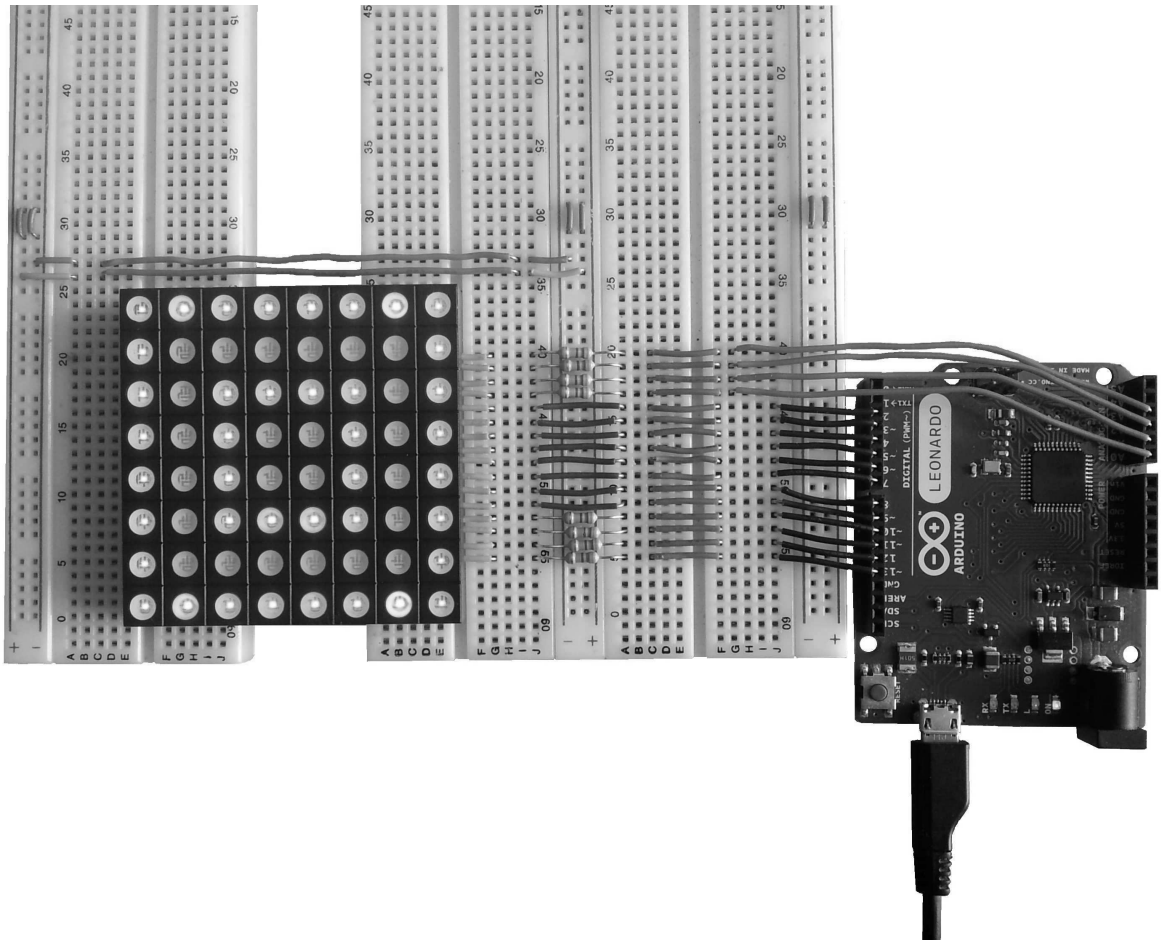
52.3 Programování

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 požadovaném obrázku. 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
        for(int j = 0; j < 8; j++){
            digitalWrite(sloupce[j], HIGH);
        }

        //vypneme radek i
        digitalWrite(radky[i], LOW);
    }
}
```

A výsledek?

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



Obrázek 52.4:

Kapitola 53

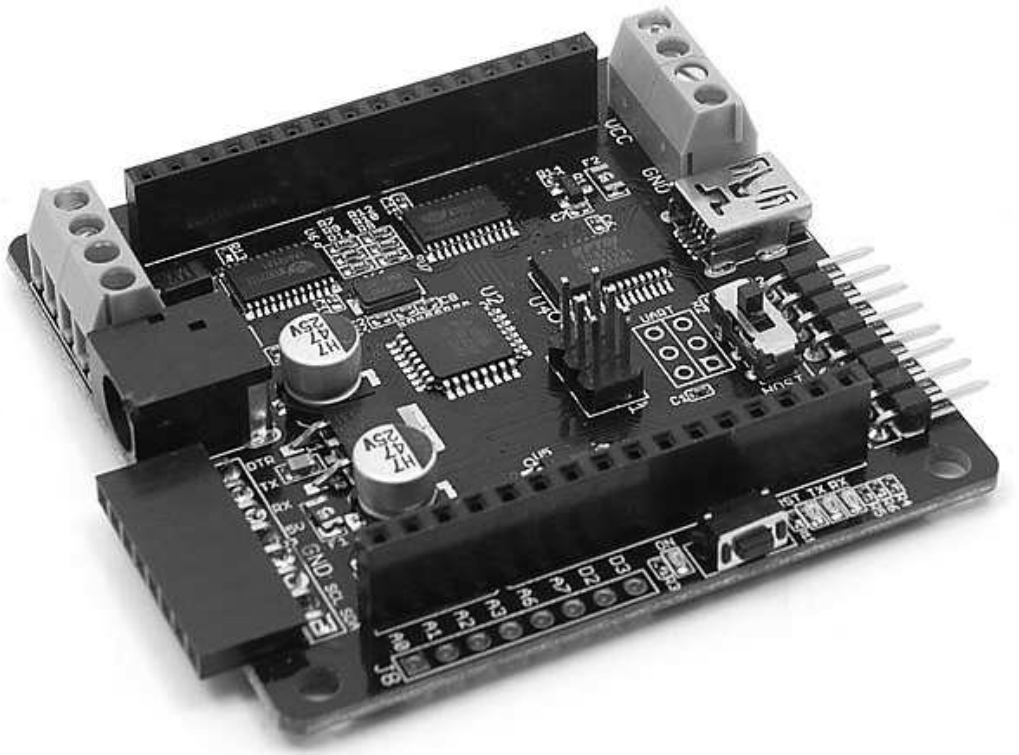
RGB teoreticky

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. 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 16777216 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.

Kapitola 54

Rainbowduino



Obrázek 54.1: Rainbowduino

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 8x8, nebo dokonce RGB kostku 4x4x4 (oboje se společnou anodou). My si je 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.

54.1 Funkce

V první řadě musíme Arduino IDE naučit s deskou Rainbowduino pracovat. To uděláme jednoduše – stáhneme knihovnu (pro IDE 1.0+) 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 umístěna v setup().
Rb.setPixelXY(x,y,r,g,b)	Nastaví barvu pixelu určeného souřadnicemi. Pixel může mít souřadnice od 0 do 7. Když si desku natočíme tak, aby USB bylo dole, je bod [0,0] v levém spodním rohu.
Rb.blankDisplay()	Vypne všechny LED diody.
Rb.drawChar(znak, x, y, RGBbarva)	Vypíše na displej znak na daných souřadnicích. Barva musí být zadána ve 32 bitovém 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 6x8, 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.

```
#include <Rainbowduino.h>

char a;

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

void loop(){
    while(Serial.available() > 0){
        a = Serial.read();
        Rb.blankDisplay();
        Rb.drawChar(a,0,1,random(0xFFFFFF));
        delay(500);
    }
}
```

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).

54.2 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(0/1):x:y:r:g:b, 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.

```
//master
#include <Wire.h>
#include <Rainbowduino.h>

byte c,x,y,r,g,b;

void setup(){
    Rb.init();
    Serial.begin(9600);
    Wire.begin();
}
void loop(){
    while(Serial.available() > 0){
        c = Serial.parseInt(); //maji se zhasnout dosud svitici LED?
        x = Serial.parseInt();
        y = Serial.parseInt();
        r = Serial.parseInt();
```

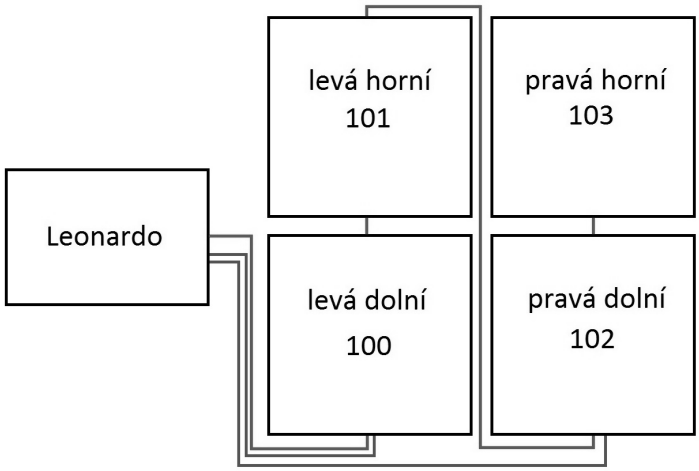
```
18          g = Serial.parseInt();
19          b = Serial.parseInt();
20          if(x >= 8 || y >= 16 || r >= 256 || g >= 256 || b >= 256){
21              Serial.println("Chybna data, nebo neplatny rozsah.");
22          }
23          else{
24              if(c == 1){
25                  Rb.blankDisplay();
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 }
```

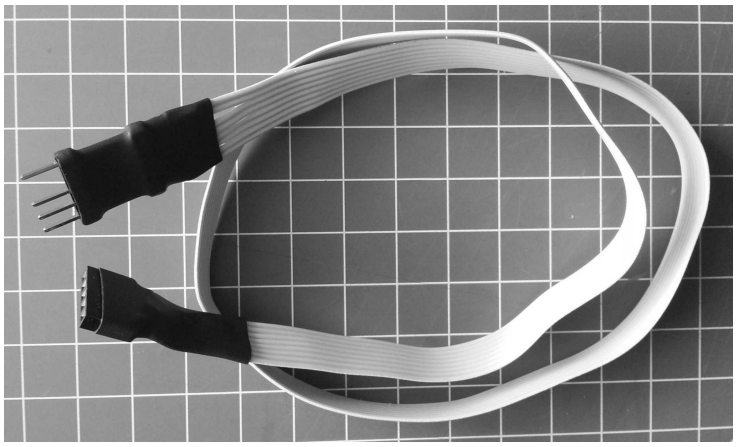
54.3 Zobrazení obrázku z PC

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 rozkouzkujeme 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 vidíte rozmístění displejů a jejich adresy.



Obrázek 54.2: Propojení Rainbowduin

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. Leonardo je zde připojeno k jednomu sloupci a ten je připojen k dalšímu. Ve druhém způsobu (modře) 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. Mě se osvědčil plochý víceżilový kabel se zdířkami na jedné straně a piny na druhé. Piny i dutinky jsou k dostání zde. Pro i2c komunikaci potřebujeme čtyři vodiče – SDA, SCL, +5V a GND. Výsledek může vypadat třeba takto – takovéto propojky budeme potřebovat dvě.



Obrázek 54.3: Propojovací kabel

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 16x16 pixelů a může to být třeba obyčejný smajlík: 

Princip jednotlivých částí je popsán v kódu.

```
1  import processing.serial.*;
2
3  PImage obr;
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; //otoceni osy y
36
37                 while(millis() - last < 5){ //chvilku pocka
38                 }
39
40                 port.write(0+":"+x+":"+y+":"+r+":"+g+":"+b+":");
41
42                 println(x+","+y+","+r+","+g+","+b);
43                 last=millis();
44             }
45             fill(0);
46             image(obr,0,0);
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.

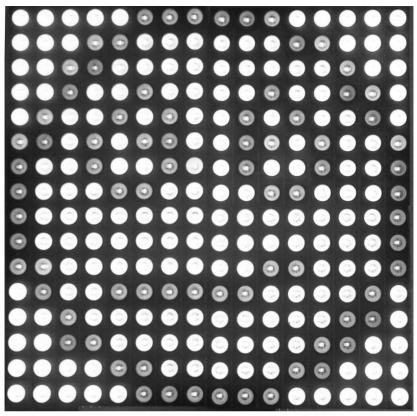
```
1  //master - Leonardo
2
3  #include <Wire.h>
4
5  byte c,x,y,r,g,b;
6
7  void setup(){
8      Serial.begin(19200);
9      Wire.begin();
10 }
11
12 void loop(){
13     while(Serial.available() > 0){
14         c = Serial.parseInt(); //maji se zhasnout dosud svitici LED?
15         x = Serial.parseInt();
16         y = Serial.parseInt();
17         r = Serial.parseInt();
18         g = Serial.parseInt();
19         b = Serial.parseInt();
20
21         if(x >= 16 || y >= 16 || r >= 256 || g >= 256 || b >= 256){
22             Serial.println("Chybna data, nebo neplatny rozsah.");
23         }
24         else{
25             if(c == 1){
26                 clearDisplay(100);
27                 clearDisplay(101);
28                 clearDisplay(102);
29                 clearDisplay(103);
30             }
31             if(x < 8 && y < 8){
32                 //levy spodni
33                 setPixel(100);
34             }
35             else if(x < 8 && y >= 8){
36                 //levy horni
37                 y = y%8;
38                 setPixel(101);
39             }
40             else if(x >= 8 && y < 8){
41                 //pravy spodni
42                 x = x%8;
43                 setPixel(102);
44             }
45             else if(x >= 8 && y >= 8){
46                 //pravy horni
47                 x = x%8;
48                 y = y%8;
49                 setPixel(103);
50             }
51         }
52     }
53 }
```

```
54 void clearDisplay(int addr){
55     Wire.beginTransmission(addr);
56     Wire.write(1); //pouze zhasne svitici LED
57     Wire.write(0);
58     Wire.write(0);
59     Wire.write(0);
60     Wire.write(0);
61     Wire.write(0);
62     Wire.endTransmission();
63 }
64
65 void setPixel(int addr){
66     Wire.beginTransmission(addr);
67     Wire.write(c);
68     Wire.write(x);
69     Wire.write(y);
70     Wire.write(r);
71     Wire.write(g);
72     Wire.write(b);
73     Wire.endTransmission();
74 }
```

Poslední a nejdůležitější částí jsou displeje. Kód všech bude téměř totožný – bude se lišit pouze v i2c adrese.

```
1  //slave – Rainbowduino
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
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.



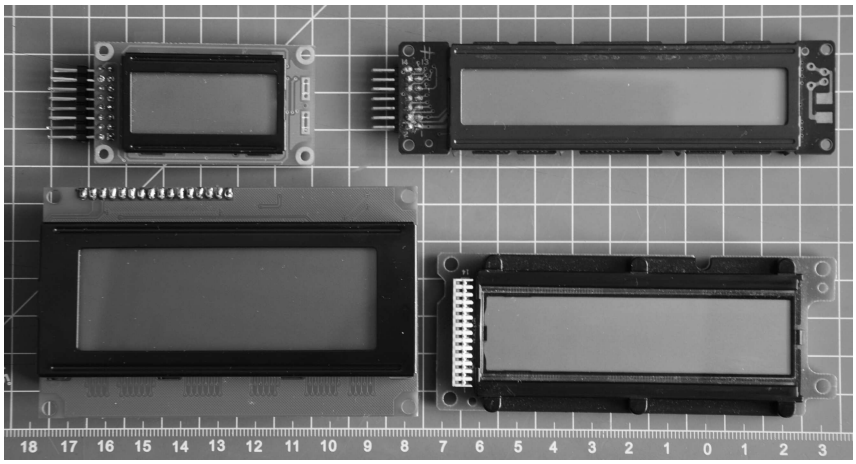
Obrázek 54.4: Výsledný obrázek

Kapitola 55

LCD displeje

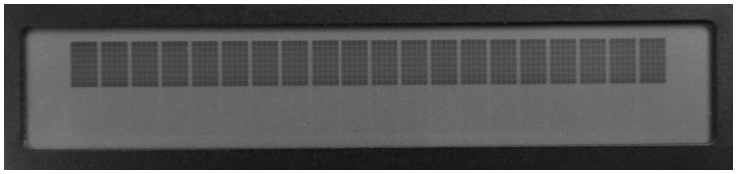
Jak už jsem zmínil před chvílí, existuje celá řada LCD displejů. Všeobecně se užívají dva typy dělení. První z nich rozděluje LCD displeje do dvou skupin na znakové a grafické. Druhý z nich je dělí na barevné a monochromatické (jednobarevné). My si nyní jednotlivé skupiny představíme.

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 8x1 (jeden řádek s osmi znaky) až 40x4 (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 55.1: LCD displeje

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



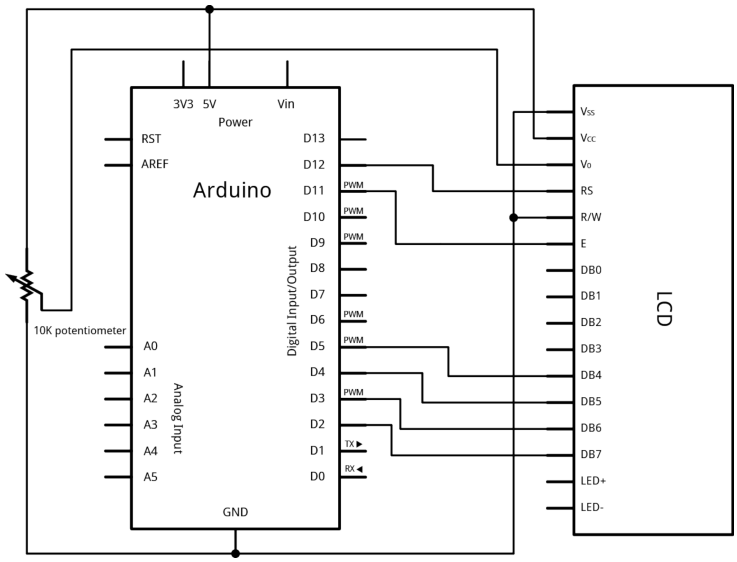
Obrázek 55.2: Maximální kontrast

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 displejů je. Tyto displeje mají většinou šestnáct pinů. V tomto příkladu budeme pracovat s tímto 20x4 LCD displejem. Pokud se podíváme na jeho zadní stranu, nalezneme zde piny popsané 1 a 16. Pojďme si je nyní představit.

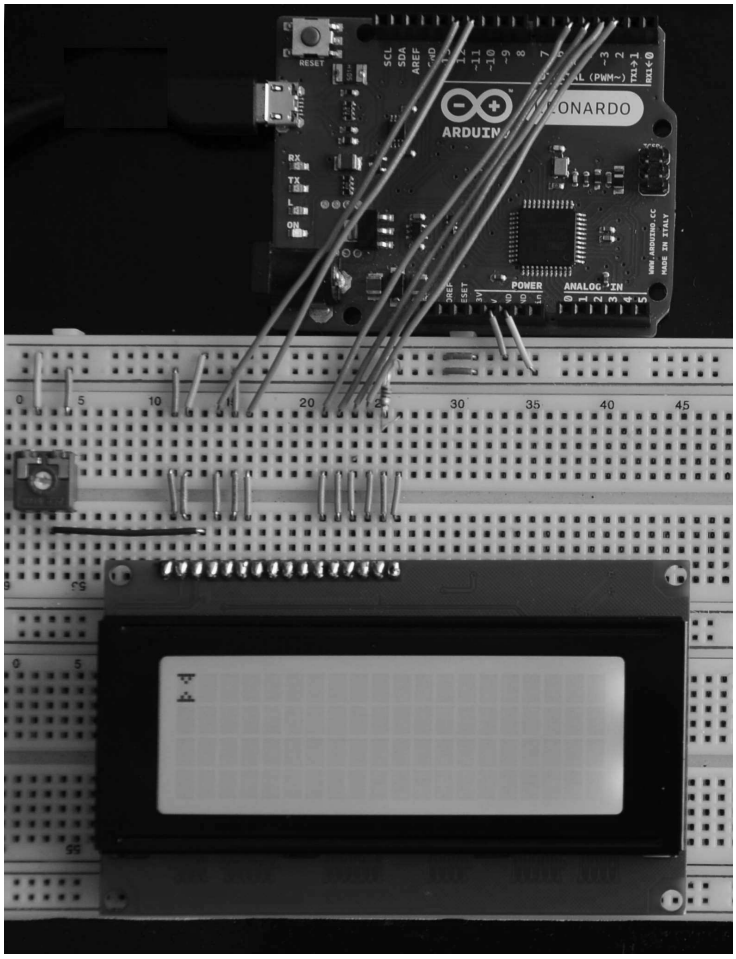
Číslo pinu	Symbol	Popis
1	VSS, GND	GND napájení displeje
2	VDD, VCC	+5V napájení displeje
3	V0	Pin pro nastavení kontrastu LCD (bude vysvětleno později)
4 – 6	RS, R/W, E	Řízení řadiče
7 – 14	DB0 – DB7	Datové piny
15	LED+	Anoda podsvícení displeje
16	LED-	Katoda displeje

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

Displej zapojíme podle schématu. Také můžeme přes 10 Ohm rezistor připojit napájení k podsvícení. Potenciometr zde slouží k nastavení kontrastu displeje. Rezistor i potenciometr jsou součástí naší sady.



Obrázek 55.3: Zapojení znakového LCD displeje



Obrázek 55.4: Znakový LCD displej

Jak už jsem řekl 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
lcd.begin(s,ř)	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,ř)	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.
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

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

void loop() {
    if(millis() % 1000 == 0){
        lcd.clear();
        lcd.setCursor(9,1);
        lcd.print(millis()/1000);
    }
}
```

Nyní si ukážeme, jak se používá funkce .createChar(). Pod hodnotu 1 si vytvoříme 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 B0 (= 0) až B11111111 (= 255). Konstanta B00000010 tedy odpovídá hodnotě 2.

```
#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() {}
```

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.

```
1  #include <LiquidCrystal.h>
2
3  LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
4
5  int i = 0;
6
7  void setup() {
8      Serial.begin(9600);
9      lcd.begin(20,4);
10 }
11
12 void loop() {
13     while(Serial.available() > 0){
14         lcd.setCursor(i%20, (i−i%20)/20);
15         lcd.write(Serial.read());
16         i++;
17         if(i == 80){
18             i = 0;
19             lcd.clear();
20         }
21     }
22 }
```

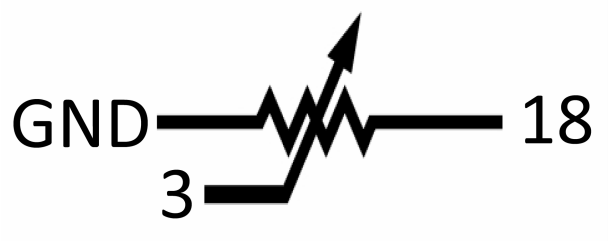
55.1 Grafické monochromatické LCD

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 tomto článku si ukážeme použití displeje ATM12864D (128 x 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. Dalším důležitým dokumentem je datasheet displeje, 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		

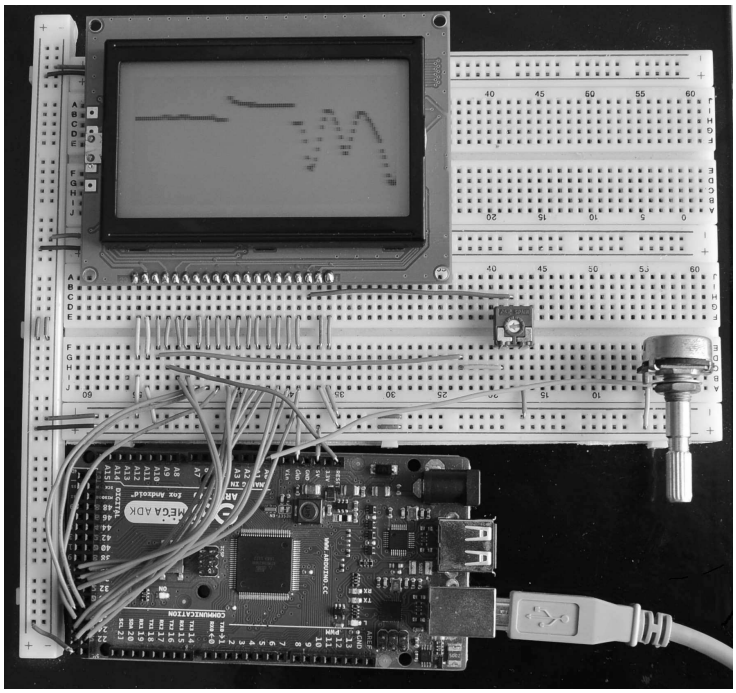
Piny 3, 18 a 19 nejsou připojeny k Arduinu. Zapojení pinů 3 a 18 s potenciometrem je naznačeno na následujícím obrázku. Pin 19 je připojen na +5V přes 220 ohm Rezistor.



Obrázek 55.5: 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(){
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]);
14         data[i] = data[i-1];
15     }
16
17     delay(40);
18
19     GLCD.ClearScreen();
20 }
21
22 void bod(int x, int y){
23     y = 63 - y; //prevraceni os
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 55.6: Graf naměřených hodnot

55.2 Barevné grafické LCD

Nyní už se konečně dostáváme k barevným displejům. Představíme si dotykový displej s úhlopříčkou 2,8 palce a 320x240 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.

Obrázek 55.7: Arduino a TFT Shield

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.

```
#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 ohm a u y 20 ohm. 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.

```
1  void loop(){
2      Point p = ts.getPoint();
3
4      x = map(p.x, min\_x, max\_x, 0, 239);
5      y = map(p.y, min\_y, max\_y, 0, 319);
6
7      if(p.z > 10){
8          Serial.print(x);
9          Serial.print(':');
10         Serial.println(y);
11     }
12
13     delay(500);
14 }
```

Kalibrace dotykové plochy displeje se dá provést získáním naměřených hodnot pomocí funkce .getPoint a postupním 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 důležitější) část je práce se samotnou zobrazovací plochou. I k tomuto účelu existuje pro náš displej knihovna, kterou stáhneme zde. 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 níže.
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]
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 řetězce dané barvy na souřadnicích [x,y] velikosti v.
Tft.setDisplayDirect(smer)	Nastaví směr textu. Smer může mít hodnoty: LEFT2RIGHT, RIGHT2LEFT, DOWN2UP a UP2DOWN.

Displej umí pracovat s 2^{16} (65536) různými barvami. Informace o barvě jednoho pixelu tedy zabere 16 bitů. Červené a modrá barva mají každá pět bitů, zelená má bitů šest, protože je na ní lidské oko více citlivé, tudíž rozezná více jejích rozdílů. Tento způsob míchání barev se nazývá 16-bit high color a více se o něm můžete dočíst na anglické Wikipedii. 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. Jedná se o trošku pokročilejší programování, na které přijde řeč možná až časem. 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.

```
1  #include <stdint.h>
2  #include <TFTv2.h> //knihovna displeje
3  #include <SPI.h> //knihovna pro komunikaci s displejem
4
5  void setup(){
6      Tft.TFTinit();
7
8      Tft.drawCircle(120, 60, 20, barva(31,0,0));
9      Tft.drawCircle(120, 160, 20, barva(0,63,0));
10     Tft.drawCircle(120, 260, 20, barva(0,0,31));
11 }
12
13 void loop(){
14
15 }
16
17 uint16_t barva(int r, int g, int b){
18     r = r % 32;
19     g = g % 64;
20     b = b % 32;
21
22     r = r << 11;
23     g = g << 5;
24
25     return r | g | b;
26 }
```

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;

int x, y;

void setup(){
    Tft.TFTinit();

    Tft.fillScreen(10,230,10,230, WHITE);
    Tft.fillScreen(10,230,240,310, GRAY2);

    Tft.drawString("RESET", 45, 255, 5, BLACK);
}
```

```
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){
50         if(x >= 10 && x <= 220 && y >= 250 && y <= 310){
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 55.8: Výsledný program

Část XIII

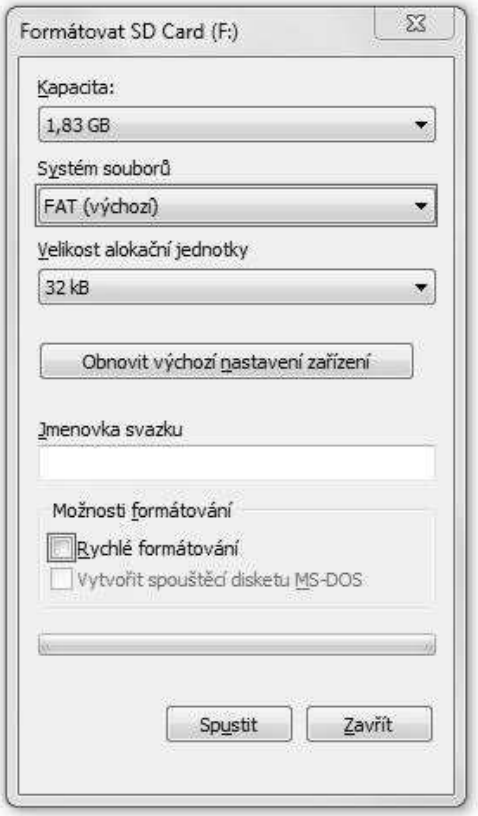
Projekt: 2048

V dnešním díle 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 56

SD karta

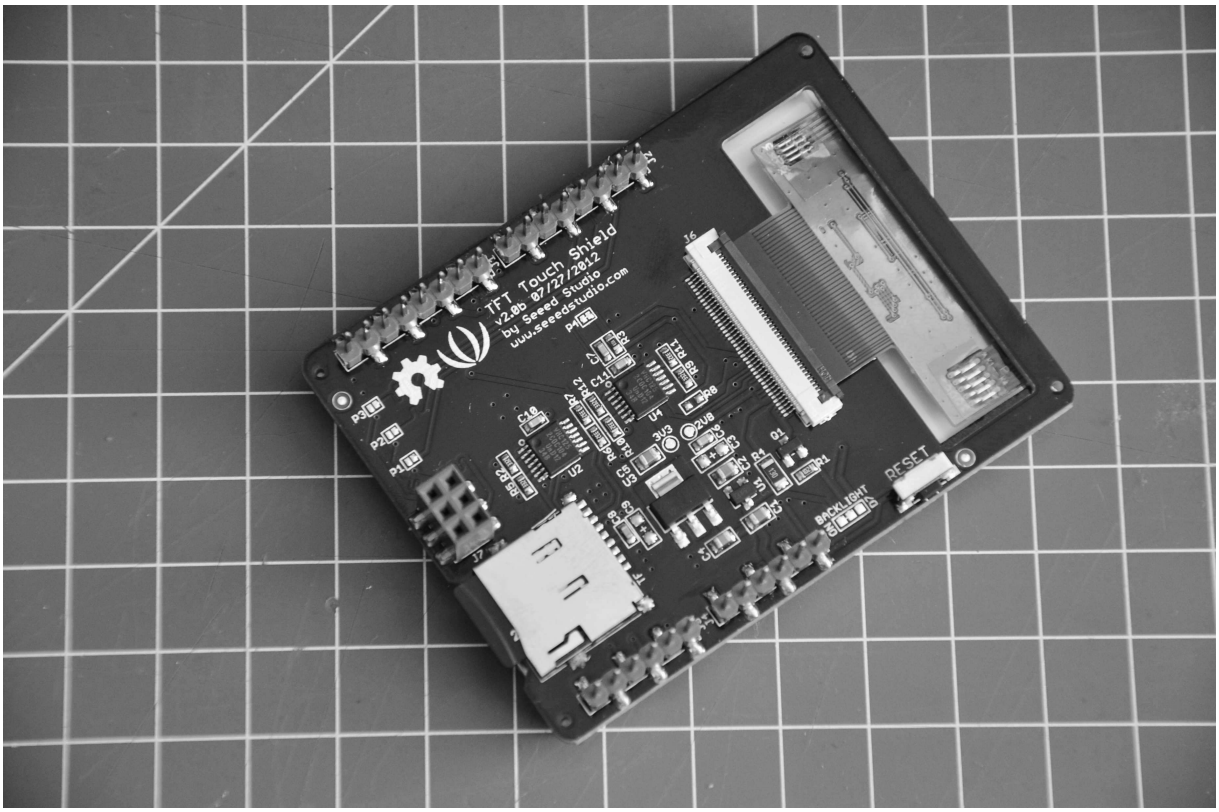
Arduino IDE obsahuje již od základu knihovnu pro obsluhu SD karet. Ta nám umožňuje pracovat s kartami, která mají formáty souborového systému FAT16 a FAT32. Název souboru nesmí mít více než 8 znaků a přípona musí mít znaky tři. Pokud má karta, se kterou chceme pracovat, jiný, než požadovaný typ, musíme 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 56.1: Formátovací dialog

56.1 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). Také existuje celá řada shieldů, které SD čtečku obsahují. Jsou jimi například SD Card Shield V4.0, Arduino Ethernet Shield SD a další. V tomto článku 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.



Obrázek 56.2:

56.2 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¿.

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. My si představíme ty nejužitečnější.

Funkce pro práci s umístěním složek a souborů

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 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 (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 jsou tedy pro práci s objektem soubor získaným takto:

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

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 dovolím použít příklady z dokumentace.

56.3 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
12     Serial.print("Initializing SD card...");
13     pinMode(10, OUTPUT); //SS pin desky se kterou pracujeme
14
15     //je karta pritomna a v poradku?
16     if (!SD.begin(chipSelect)) {
17         Serial.println("Card failed, or not present");
18         return;
19     }
20     Serial.println("card initialized.");
21 }
22
23 void loop(){
24     //vytvori retezec pro shromazdovani dat
25     String dataString = "";
26
27     //precti 3 vstupy a vysledek zapis do retezce
28     for (int analogPin = 0; analogPin < 3; analogPin++) {
29         int sensor = analogRead(analogPin);
30         dataString += String(sensor);
31         if (analogPin < 2) {
32             dataString += ",";
33         }
34     }
35
36     //otevre soubor
37     File dataFile = SD.open("datalog.txt", FILE_WRITE);
38
39     // pokud se otevreni povedlo, data se zapisi do slozky
40     if (dataFile) {
41         dataFile.println(dataString);
42         dataFile.close();
43         Serial.println(dataString);
44     }
45     else {
46         Serial.println("error opening datalog.txt");
47     }
48 }
```

56.4 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 57

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 4x4 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 a čtečku karet pro uložení postupu.

57.1 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 4x4 dosáhnout maximální hodnoty 17. Princip zjištění maximální hodnoty je vidět na obrázku.

4	5	13	14
3	6	12	15
2	7	10	16
1	8	9	17

Obrázek 57.1: Maximální hodnoty

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

57.2 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.

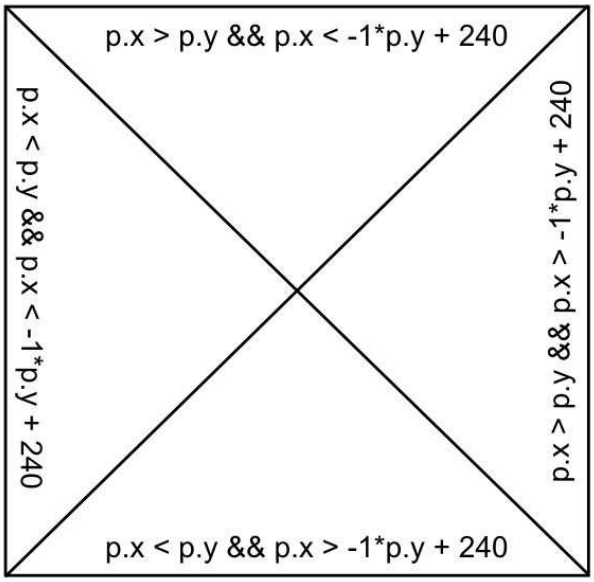
```
74 void loop(){
75     if(rewrite_all){
76         Tft.fillRectangle(0,0,239,319, barvy[0]);
77     }
78
79     //uvodni obrazovka
80     if(stat == 0){
81         if(rewrite_all){
82             Tft.fillRectangle(10,10,220,145, barvy[2]);
83             Tft.fillRectangle(10,165,220,145, barvy[2]);
84             Tft.drawString("LOAD", 35, 55, 7, BLACK);
85             Tft.drawString("NEW", 55, 210, 7, BLACK);
86             rewrite_all = false;
87         }
```

```
88          checkP();
89
90          if(p.z > 10){
91              //tlacitko LOAD
92              if(p.y < 160){
93                  rewrite_all = true;
94                  rewrite_game = true;
95                  if(!SD.exists("G2048.TXT")){
96                      stat = 0;
97                      Tft.fillRectangle(20,95,200,100, barvy[17]);
98                      Tft.drawString("ERROR", 28, 123, 6, BLACK);
99
100                     delay(500);
101                 }
102                 else{
103                     stat = 1;
104                     soubor = SD.open("G2048.TXT");
105                     int i = 0;
106                     byte p;
107
108                     for(int i=0; i < 4; i++){ //vycistime plochu
109                         for(int j = 0; j < 4; j++){
110                             plocha[i][j] = 0;
111                         }
112                     }
113
114                     while(soubor.available()){
115                         p = soubor.read();
116                         plocha[(i - i%4) /4][i % 4] = p;
117                         i++;
118                     }
119                     soubor.close();
120                 }
121             }
122
123             //tlacitko NEW
124             else if(p.y >= 160){
125                 rewrite_all = true;
126                 rewrite_game = true;
127                 moved = false;
128                 added = 0;
129                 gEnd = false;
130                 gWin = false;
131                 stat = 1; //pokracovat budeme hrou
132
133                 //cela plocha se nastavi na 0
134                 for(int i = 0; i < 4; i++){
135                     for(int j = 0; j < 4; j++){
136                         plocha[i][j] = 0;
137                     }
138                 }
139
140                 addTile();
141                 addTile();
142             }
143         }
144     }
```

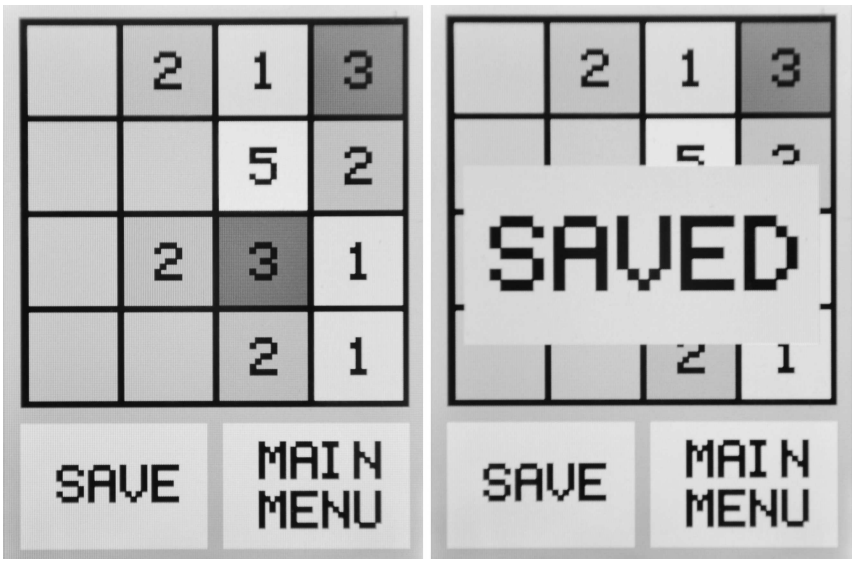
Pokračovat budeme případem normální hry. Pokud je proměnná rewrite_game true, dojde k přepsání herní plochy. Dále program detekuje, jestli došlo k dotyku v oblasti herní plochy a tlačítek. Ovládání směru je nastaveno tak, jak vidíte na obrázku. Detekce dotyku v oblasti směrových šipek vychází z funkce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

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.



Obrázek 57.2: Segmenty displeje



Obrázek 57.3: Herní plocha

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

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

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

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

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

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.

```
368 //nastavi aktualni pozici dotyku uzivatele
369 void checkP(){
370     p = ts.getPoint();
371     p.x = map(p.x, min_x, max_x, 0, 240);
372     p.y = map(p.y, min_y, max_y, 0, 320);
373 }
374
375 //prida novou dlazdici
376 void addTile(){
377     byte r1, r2;
378
379     do{
380         r1 = random(4);
381         r2 = random(4);
382     }while(plocha[r1][r2] != 0);
383
384     plocha[r1][r2] = random(1,3);
385 }
386
387 void goUp(){
388     for(int x = 0; x <= 3; x++){
389         if(plocha[0][x] == plocha[1][x] && plocha[2][x] == plocha[3][x]){
390             max_added = 2;
391         }
392         else{
393             max_added = 1;
394         }
395
396         for(int y = 0; y <= 2; y++){
397             for(int y_p = y; y_p >= 0; y_p--){
398                 if(plocha[y_p][x] == plocha[y_p+1][x] && plocha[y_p][x] != 0 && added < max_added){
399                     plocha[y_p][x]++;
400                     plocha[y_p+1][x] = 0;
401                     added++;
402                     moved = true;
403                 }
404                 if(plocha[y_p][x] == 0 && plocha[y_p+1][x] != 0){
405                     plocha[y_p][x] = plocha[y_p+1][x];
406                     plocha[y_p+1][x] = 0;
407                     moved = true;
408                 }
409             }
410         }
411         added = 0;
412     }
413 }
```

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).

```
414 void goUp(){
415     for(int x = 0; x <= 3; x++){
416         if(plocha[0][x] == plocha[1][x] && plocha[2][x] == plocha[3][x]){
417             max_added = 2;
418         }
419         else{
420             max_added = 1;
421         }
422 
423         for(int y = 0; y <= 2; y++){
424             for(int y_p = y; y_p >= 0; y_p--){
425                 if(plocha[y_p][x] == plocha[y_p+1][x] && plocha[y_p][x] != 0 && added < max_added){
426                     plocha[y_p][x]++;
427                     plocha[y_p+1][x] = 0;
428                     added++;
429                     moved = true;
430                 }
431                 if(plocha[y_p][x] == 0 && plocha[y_p+1][x] != 0){
432                     plocha[y_p][x] = plocha[y_p+1][x];
433                     plocha[y_p+1][x] = 0;
434                     moved = true;
435                 }
436             }
437         }
438         added = 0;
439     }
440 }
441 
442 void goDown(){
443     for(int x = 0; x <= 3; x++){
444         if(plocha[0][x] == plocha[1][x] && plocha[2][x] == plocha[3][x]){
445             max_added = 2;
446         }
447         else{
448             max_added = 1;
449         }
450 
451         for(int y = 3; y >= 1; y--){
452             for(int y_p = y; y_p <= 3; y_p++){
453                 if(plocha[y_p-1][x] == plocha[y_p][x] && plocha[y_p][x] != 0 && added < max_added){
454                     plocha[y_p-1][x]++;
455                     plocha[y_p][x] = 0;
456                     added++;
457                     moved = true;
458                 }
459                 if(plocha[y_p][x] == 0 && plocha[y_p-1][x] != 0){
460                     plocha[y_p][x] = plocha[y_p-1][x];
461                     plocha[y_p-1][x] = 0;
462                     moved = true;
463                 }
464             }
465         }
466         added = 0;
467     }
468 }
```

```
469 void goRight(){
470     for(int y = 0; y <= 3; y++){
471         if(plocha[y][0] == plocha[y][1] && plocha[y][2] == plocha[y][3]){
472             max_added = 2;
473         }
474         else{
475             max_added = 1;
476         }
477         for(int x = 3; x >= 1; x--){
478             for(int x_p = x; x_p <= 3; x_p++){
479                 if(plocha[y][x_p-1] == plocha[y][x_p] && plocha[y][x_p] != 0 && added < max_added){
480                     plocha[y][x_p-1]++;
481                     plocha[y][x_p] = 0;
482                     added++;
483                     moved = true;
484                 }
485                 if(plocha[y][x_p] == 0 && plocha[y][x_p-1] != 0){
486                     plocha[y][x_p] = plocha[y][x_p-1];
487                     plocha[y][x_p-1] = 0;
488                     moved = true;
489                 }
490             }
491         }
492         added = 0;
493     }
494 }
495
496 void goLeft(){
497     for(int y = 0; y <= 3; y++){
498         if(plocha[y][0] == plocha[y][1] && plocha[y][2] == plocha[y][3]){
499             max_added = 2;
500         }
501         else{
502             max_added = 1;
503         }
504         for(int x = 0; x <= 2; x++){
505             for(int x_p = x; x_p >= 0; x_p--){
506                 if(plocha[y][x_p] == plocha[y][x_p+1] && plocha[y][x_p] != 0 && added < max_added){
507                     plocha[y][x_p]++;
508                     plocha[y][x_p+1] = 0;
509                     added++;
510                     moved = true;
511                 }
512                 if(plocha[y][x_p] == 0 && plocha[y][x_p+1]){
513                     plocha[y][x_p] = plocha[y][x_p+1];
514                     plocha[y][x_p+1] = 0;
515                     moved = true;
516                 }
517             }
518         }
519         added = 0;
520     }
521 }
```

Celý program je možné stáhnout ze stránky github.

Část XIV

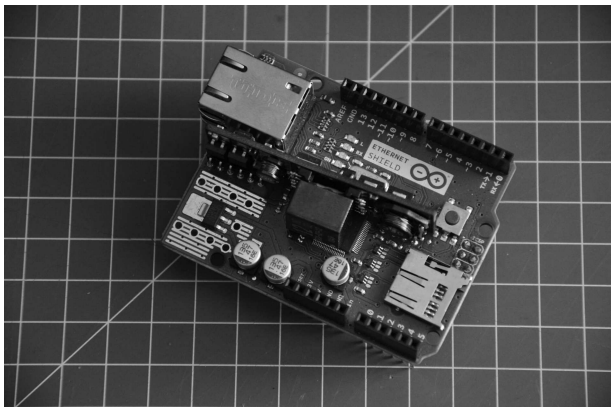
Arduino a Ethernet shield

V dnešním článku 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 58

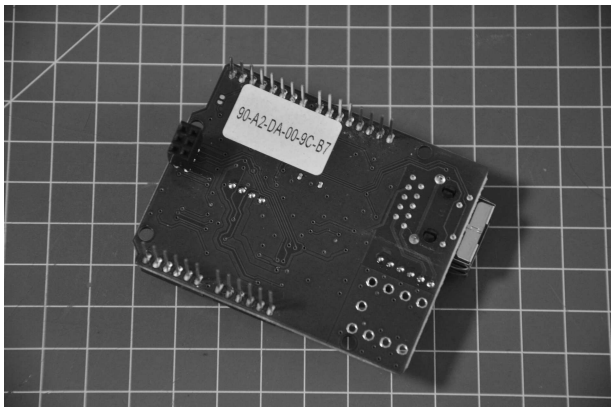
Ethernet Shield

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



Obrázek 58.1: Ethernet shield

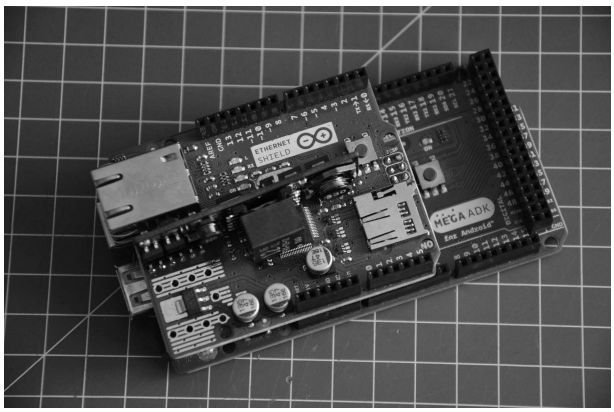
Dominantním prvkem desky je RJ45 konektor pro připojení Ethernet kabelu. Mimo něj ale na shieldu nalezneme i slot na SD kartu, jejíž používání jsme si popsali již dříve. Ovládání čipu (W5100) i SD karty probíhá přes SPI rozhraní. Rychlost síťové komunikace 10/100 MB není v dnešní době zrovna strhující,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



Obrázek 58.2: 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 našem případě Arduino Mega).



Obrázek 58.3: Arduino Mega s připojeným Ethernet shieldem

58.1 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()	-	Vrátí IP adresu shieldu. Tato funkce se nepoužívá, pokud IP adresu přiřazujeme manuálně, ale když ji necháme přes DHCP přidělit automaticky.
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 ve tvaru a.b.c.d (např. 10.0.0.1) se jménem mojeIP se provede vytvořením 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	-	Vytvoří klienta s názvem ja.
ja	while(!ja)delay(1)	Počká, dokud není klient připojen.
ja.connected()	-	Vrátí true, pokud klient odesílá data (v dobu zpracování už nemusí být přítomen, ale musí být od něj přijaty 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.

58.2 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
```

Tato zpráva obsahuje vše potřebné pro server pro odeslání vodných 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, jejichž seznam nalezneme například na Wikipedii. Hlavička odpovědi tedy vypadá takto (opět s volným řádkem).

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

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

```
1  <!DOCTYPE HTML> - rikame prohlizeci, ze pracujeme s HTML
2  <html>
3      <head>
4          mezi znacky head se pisi zakladni informace o strance (kodovani, CSS styly...)
5          <title>Titulek stranky </title> - zobrazi se v zalozce
6      </head>
7      <body>
8          sem patri obsah stranky (ten, ktery vidime v prohlizeci)
9      </body>
10 </html>
```

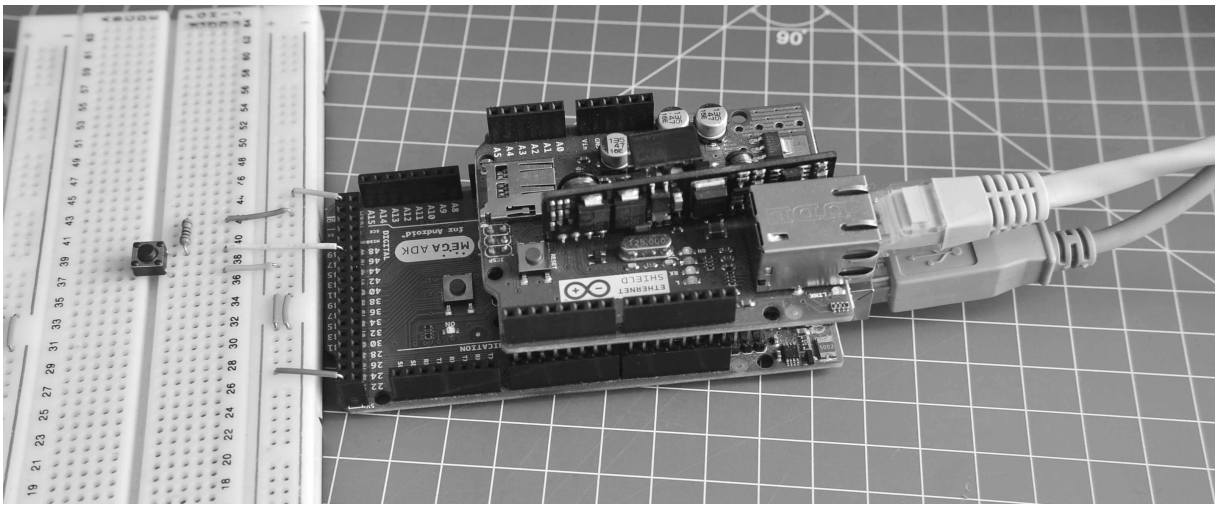
Se všemi získanými informacemi už můžeme poskládat program jednoduchého serveru, který bude měnit barvu pozadí pomocí css stylů podle toho, jestli je nebo není stisknuto tlačítko připojené k Arduinu na pinu 45. Tlačítko budeme kontrolovat každou vteřinu. Mimo tlačítka budeme potřebovat ještě 10k resistor 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 elementu <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)
            char c = client.read(); //precti byte od klienta

            if(c == 'n' && prazdnyRadek){
                client.println("HTTP/1.1 200 OK");
                client.println("Content-Type: text/html");
                client.println("Connection: close");
                client.println("Refresh: 1");
                client.println();
                client.println("<!DOCTYPE HTML>");
                client.println("<html>");
                client.println("<head>");
                client.println("<title>Zkoumame HTML a HTTP</title>");
                client.println("</head>");
                if(digitalRead(45) == HIGH){
                    client.println("<body style="background:green">");
                }
                else{
                    client.println("<body style="background:red">");
                }
```

```
43                      client.println("</body>");
44                      client.println("</html>");
45                  }
46
47                  if(c == 'n'){
48                      prazdnyRadek = true;
49                  }
50                  else if(c != 'r'){
51                      prazdnyRadek = false;
52                      /*dokud klient neposle dvakrat za sebou r a n
53                      znamena to, ze stale odesila data*/
54                  }
55              }
56              delay(1); //dame klientovi cas na zpracovani
57              client.stop(); //komunikace je u konce
58          }
59  }
```



Obrázek 58.4: Arduino Mega s Ethernet shieldem a připojeným tlačítkem

58.3 Sosáme 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 ahwk.ic.cz, po kterém budeme chtít soubor ahoj.txt. Ten je uložen v kořenovém adresáři serveru, tedy přímo na adrese ahwk.ic.cz/ahoj.txt. Na začátek si sestavíme HTTP požadavek.

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

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

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

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

char server[] = "ahwk.ic.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: ahwk.ic.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
    }
}
```

58.4 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 ?. My tedy budeme v HTTP požadavku klienta hledat ? a poté čísla za ním následující. Náš program poté rozsvítí postupně LED diody daných čísel. Pokud napíšeme do prohlížeče: 10.0.0.15/?3456, HTTP požadavek odeslaný na server vypadá nějak takto.

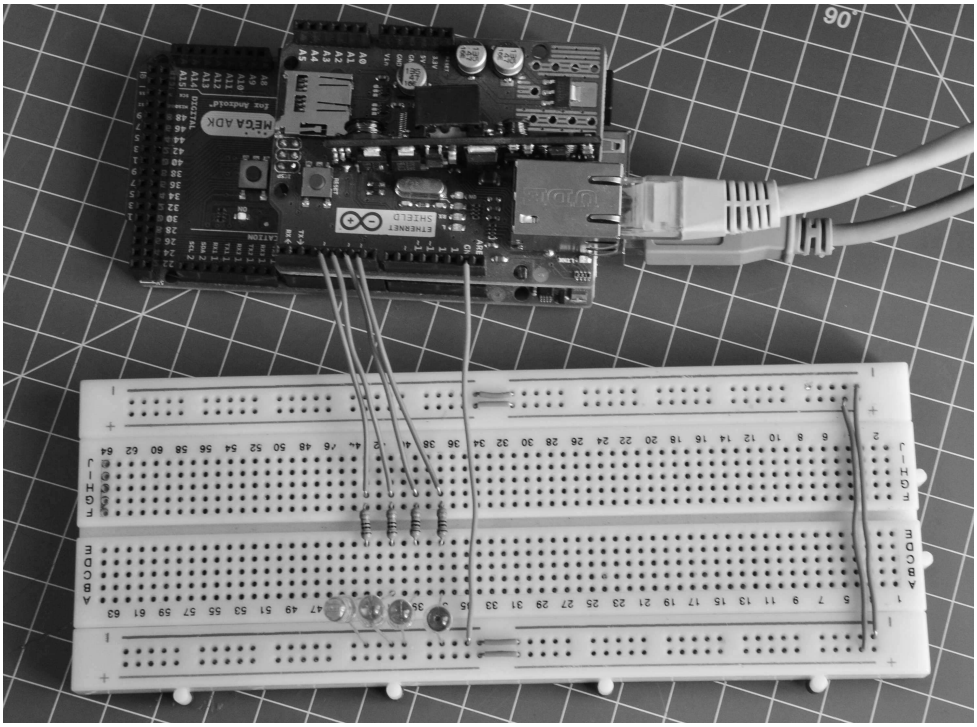
```
1 GET /?3456 HTTP/1.1
2 Host: 10.0.0.15
3 Connection: keep-alive
4 Cache-Control: max-age=0
5 Accept: text/html,application/xhtml+xml, application/xml;q=0.9, image/webp,*/*;q=0.8
6 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.114
      Safari/537.36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

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.

```
1  #include <Ethernet.h>
2  #include <SPI.h>
3
4  boolean zacatekCteni = false;
5
6  byte mac[] = {0x90, 0xA2, 0xDA, 0x00, 0x9C, 0xB7 };
7  IPAddress ip(10,0,0,15);
8
9  EthernetServer mujSvr = EthernetServer(80);
10
11 void setup(){
12   pinMode(3, OUTPUT);
13   pinMode(4, OUTPUT);
14   pinMode(5, OUTPUT);
15   pinMode(6, OUTPUT);
16
17   Ethernet.begin(mac, ip);
18
19   mujSvr.begin();
20 }
21
22 void loop(){
23     EthernetClient client = mujSvr.available();
24
25     if(client){
26         boolean prazdnyRadek = true;
27         boolean hlavickaPoslana = false;
```

```
28          while(client.connected() && client.available()){
29              if(!hlavickaPoslana){ //jednou posleme hlavicku
30                  client.println("HTTP/1.1 200 OK");
31                  client.println("Content-Type: text/html");
32                  client.println();
33                  hlavickaPoslana = true;
34              }
35
36              char c = client.read();
37              if(zacatekCteni && c == ' '){ //ukonci cteni
38                  zacatekCteni = false;
39              }
40              if(c == '?'){ //zacne cteni
41                  zacatekCteni = true;
42              }
43
44              if(zacatekCteni){
45                  if(c == '3'){
46                      blikni(3, client);
47                  }
48                  else if(c == '4'){
49                      blikni(4, client);
50                  }
51                  else if(c == '5'){
52                      blikni(5, client);
53                  }
54                  else if(c == '6'){
55                      blikni(6, client);
56                  }
57              }
58
59              if (c == 'n') {
60                  prazdnyRadek = true;
61              }
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 oficiální dokumentaci.



Obrázek 58.5: Ovládání LED diod přes Ethernet Shield

Část XV

Náš první klon Arduina

V dnešním dílu 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 článku napovídá, vytvoříme vlastní klon Arduina.

Už na začátku jsem se zmínil 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 59

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

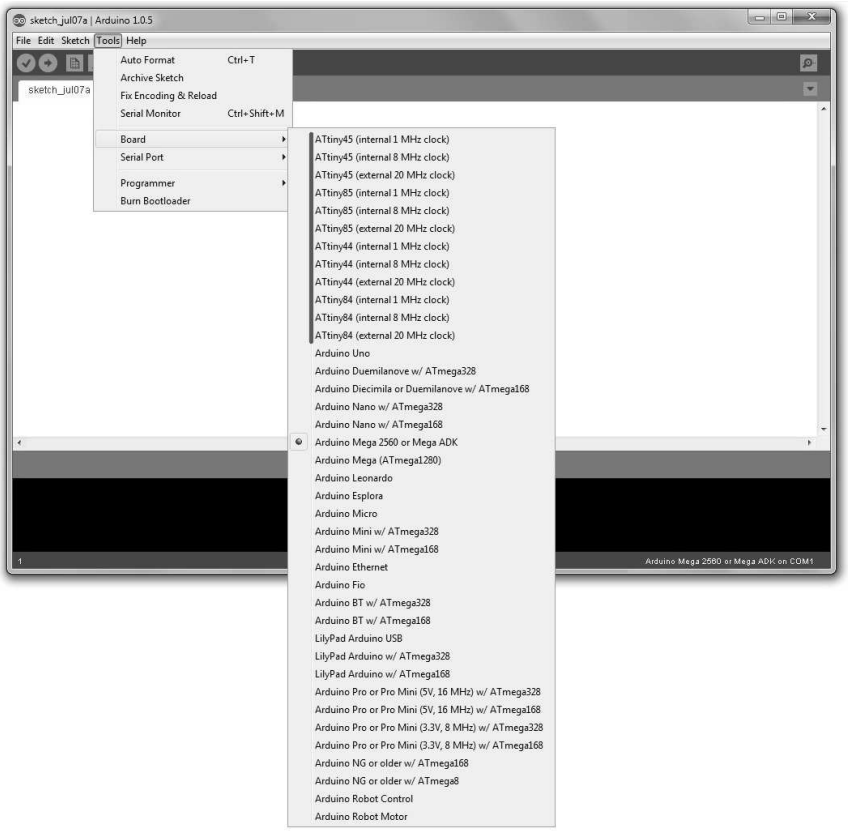
Informace o pinech jsou k nalezení v dokumentaci každé desky. Některé verze ale nemusejí mít tyto piny lehce dostupné. Například Leonardo je nemá vyvedeny standardně. O jeho použití jako ISP programátor pojednává tento článek.

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 60

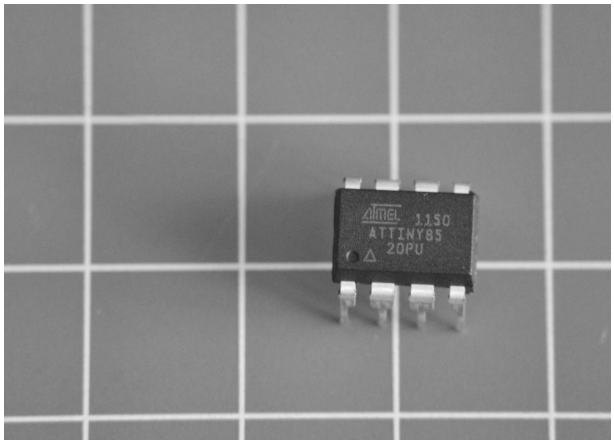
Čipy ATtiny

Jak už jsem zmínil 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 s 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 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.

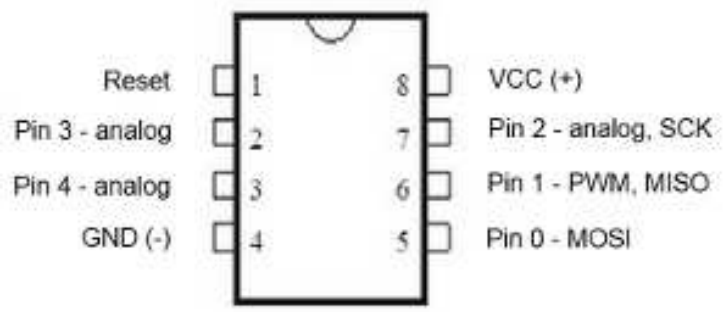


Obrázek 60.1: Menu Boards

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 3. Orientaci čipu určíme buď podle půlkruhového symbolu na jedné ze stran (viz obr. 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).



Obrázek 60.2: ATtiny 85



Obrázek 60.3: ATTiny 85 pinout

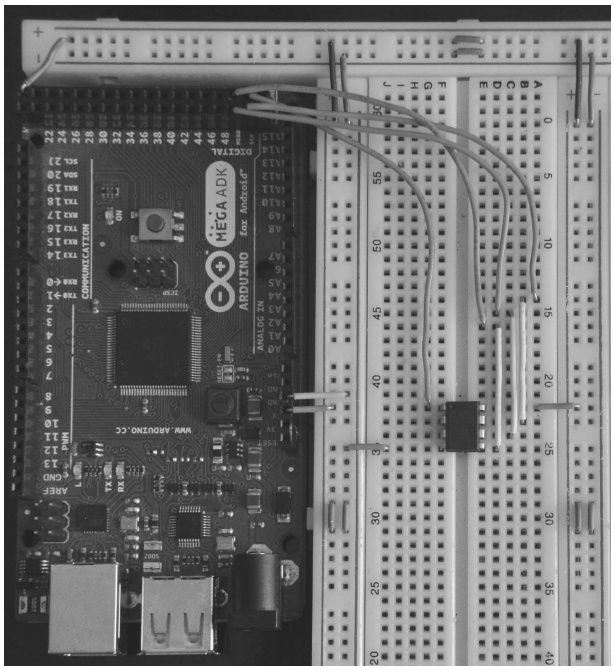
Jelikož se jedná o malé čipy s menším výkonem, i škála použitelných funkcí je zde omezena. Většinu základních funkcí zde ale najdeme. Jsou to:

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

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 60.4: Zapojení

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

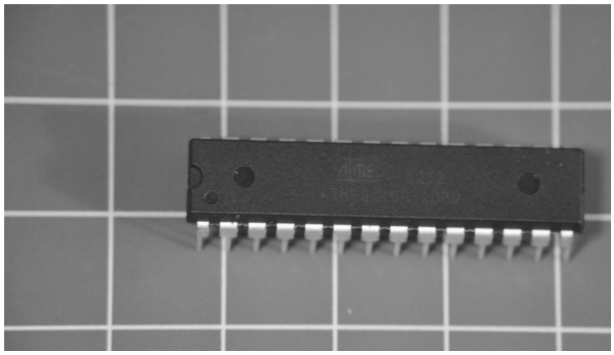
```
1  int led = 0;
2
3  void setup() {
4    pinMode(led, OUTPUT);
5  }
6
7  void loop() {
8    digitalWrite(led, HIGH);
9    delay(1000);
10   digitalWrite(led, LOW);
11   delay(1000);
12 }
```

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.

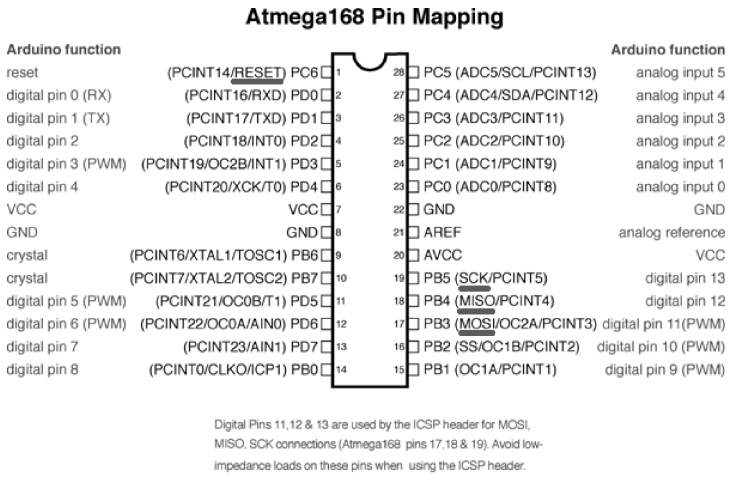
Kapitola 61

Čipy ATmega

Pokud se chceme od malých čipů ATtiny přesunout k něčemu výkonnějšímu, 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 piny, ke kterým nalezneme patřičnou modifikaci souboru boards.txt. Jako nejjednodušší se jeví použití těch čipů, na kterých jsou založeny standartní desky Arduina, tedy ATmega168, ATmega328 a další. My si ukážeme použití čipu ATmega168. Stejně jako u ATtiny, i zde musíme najít potřebné piny.

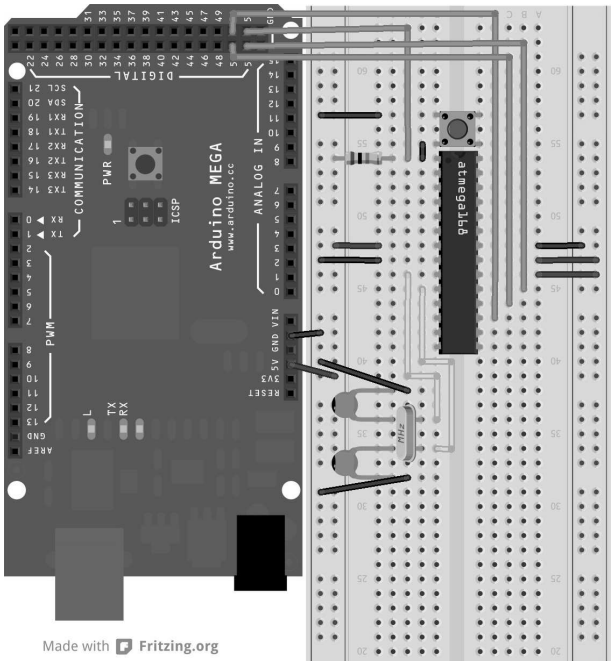


Obrázek 61.1: ATmega 168



Obrázek 61.2: ATmega 168 pinout

Abychom využili celý potenciál čipu, budeme potřebovat ještě 16 MHz oscilátor, 10 kOhm resistor, dva 22 pF kondenzátory a spínač (tlačítko). Také budeme potřebovat USB-serial převodník. Na výběr máme z více možností – FTDI Basic Breakout – 5V, USB 2 Serial Converter a další. Vše zapojíme podle schématu.



Obrázek 61.3: Arduino on board

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 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.

Část XVI

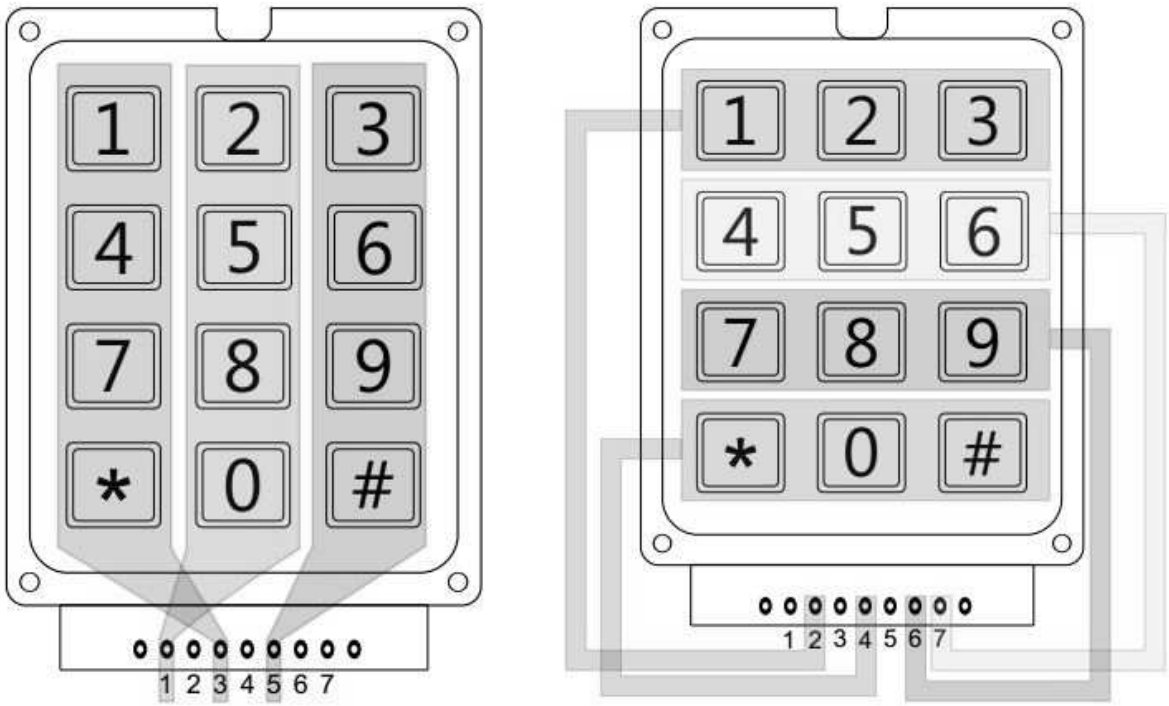
Projekt: Programátorská klávesnice

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 – vytvoříme si klávesnici pro programátory obsahující všechny potřebné závorky.

Kapitola 62

Keypad

Pro naše účely použijeme tento dvanáctitlačítkový keypad. Jeho tlačítka jsou uspořádány do matice 3 x 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.

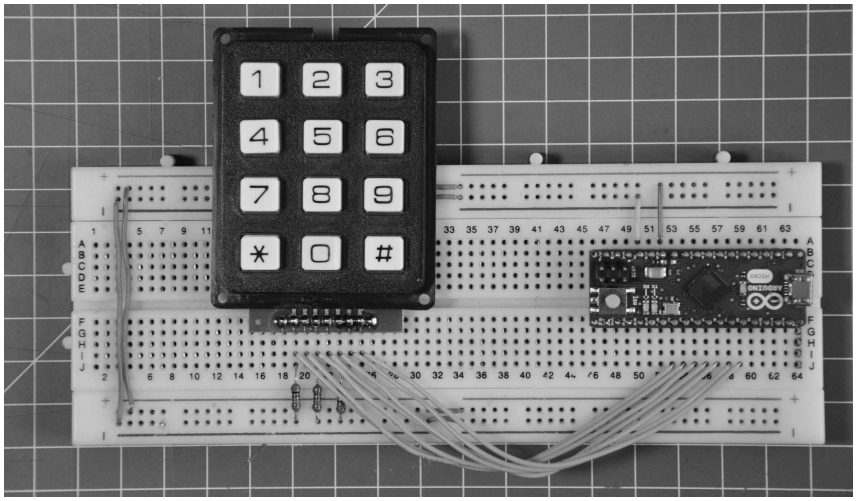


Obrázek 62.1: Vnitřní propojení keypadu

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 10k ohm resistor GND a také vstup (standardní připojení 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 10k ohm resistor na GND. Druhou skupinu připojíme na výstupy. Do těch budeme stále dokola střídavě pouštět proud, který poté budeme detekovat na vstupech. K dispozici pak máme dvě informace: na jaký pin pouštíme proud a na kterém jsme ho detekovali. Z těchto informací už můžeme zjistit, jaké tlačítko je stisknuto.

62.1 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 10k resistor 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. 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.



Obrázek 62.2: Zapojení Keypadu

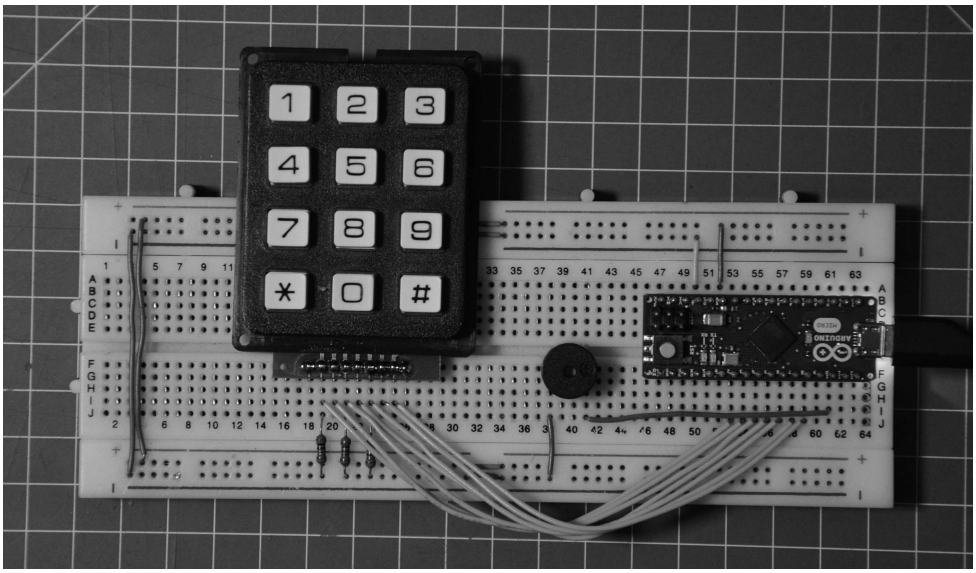
Na začátku musíme programu říct, jaké piny jsou připojeny na co. To budeme mít uloženo ve dvou polích – sloupce a radky. Navíc si sloupce a řádky seřadíme z leva doprava a odshora dolů. Vytvoříme si dvojrozměrné pole se znaky odpovídajícími klávesám. Poté si piny odpovídající řádkům nastavíme jako výstupy a sloupce jako vstupy. V těle funkce setup() si ještě nastavíme aktuální čas. Ve funkci loop() budeme neustále dokola pouštět proud do jednotlivých řádků a detekovat jeho průchod na sloupcích.

```
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 63

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 63.1: 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++){
            if(digitalRead(sloupce[b]) == HIGH
            && millis() - cekej > cas){
                tone(10,440,100);
                cas = millis();

                if(znaky[a][b] == '#'){
                //pokud najde krizek, vymaze kod
                    vynuluj();
                }
                else if(znaky[a][b] == '*'){
                //kdyz najde hvezdicku, zkontroluje shodu
                    for(int c = 0; c < 100; c++){
                        if(kod[c] != spravnyKod[c]){
                            chyby++;
                        }
                    }
                    if(chyby > 0){
                        vynuluj();
                        for(int d = 0; d < 10; d++){ //sirena
                            for(int c = 200; c < 2000; c++){
                                tone(10, c, 5);
                            }
                            for(int c = 2000; c > 200; c--){
                                tone(10, c, 5);
                            }
                        }
                    }
```

```
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 64

Programátorská klávesnice

Posledním příkladem využívajícím keypad je slibovaná programátorská klávesnice. Ta naše bude používat jenom osm kláves keypadu – pro každý typ závorek dvojici kláves (pravá a levá 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 počítače odešleme pomocí funkce Keyboard.print() (viz Arduino jako klávesnice a myš). 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. Kód klávesnice vypadá následovně.

```
1  byte radky[4] = {3,8,7,5};
2  byte sloupce[3] = {4,2,6};
3
4  char znaky[4][3] = {{40,41,' '},
5                      {123,125,' '},
6                      {91,93,' '},
7                      {60,62,' '}};
8
9  int cekej = 200;
10 long cas;
11
12 void setup(){
13     for(int i = 0; i < 4; i++){
14         pinMode(radky[i], OUTPUT);
15     }
```

```
16      for(int i = 0; i < 3; i++){
17          pinMode(sloupce[i], INPUT);
18      }
19
20      cas = millis();
21  }
22  void loop(){
23      for(int a = 0; a < 4; a++){
24          digitalWrite(radky[a],HIGH);
25          for(int b = 0; b < 3; b++){
26              if(digitalRead(sloupce[b]) == HIGH
27              && (millis() - cekej) >= cas){
28                  zmackniCesky(znaky[a][b]);
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 smajlíková klávesnice.

Část XVII

Projekt: Robotická ruka

Kapitola 65

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:

- **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
7  void setup(){
8      motor.attach(10); //pripojime servo na pin 10
9  }
10
11
12 void loop(){
13     h = map(analogRead(A5),0,1024,0,180);
14
15     motor.write(h);
16 }
```

To je k teorii řízení serv vše. Jednoduché že? Nyní se tedy můžeme pustit do projektu robotické ruky.

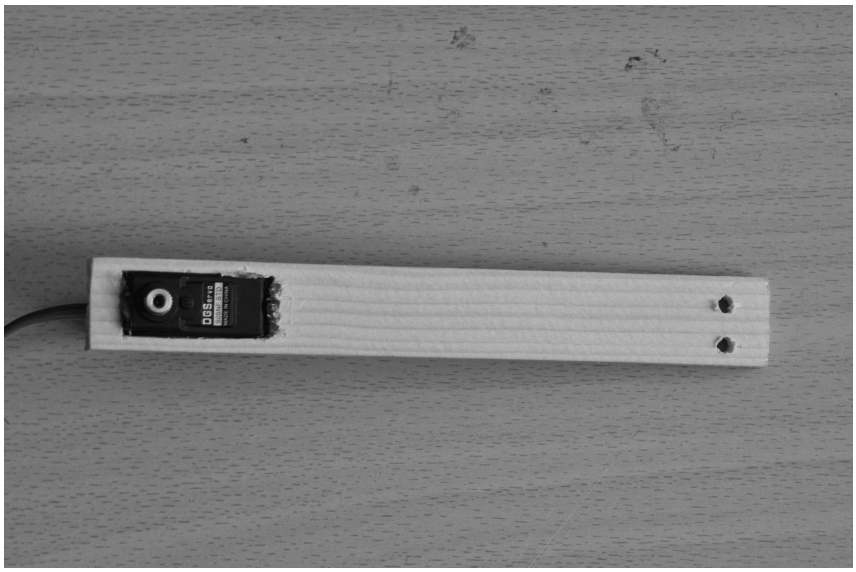
Kapitola 66

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:

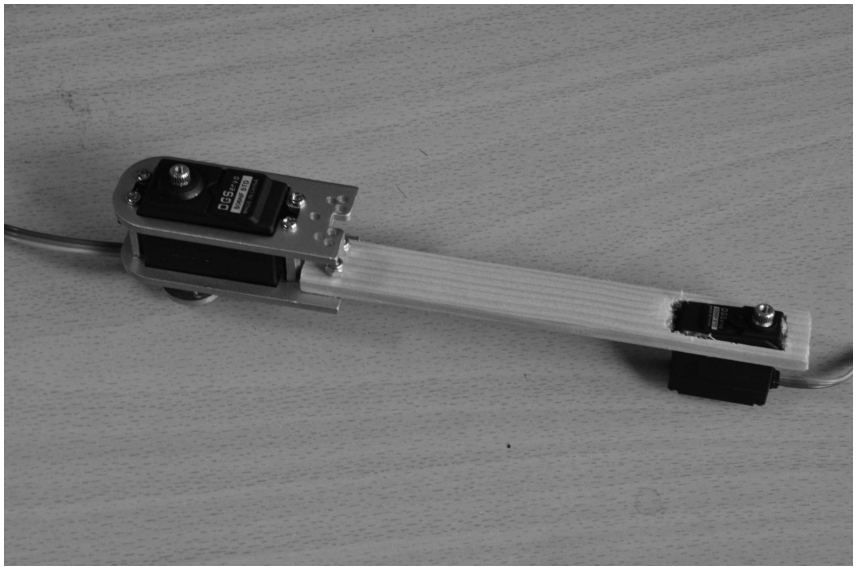
1. Dřevo na mechanické konstrukce – v našem případě dřevěná laťka s průřezem 20x5mm.
2. Tavná pistole (nebo šrouby a herkules či jiné lepidlo – v závislosti na provedení)
3. Vodiče
4. Piny
5. Dutinkovou lištu – třípinovou
6. Kousek prototypovacího plošného spoje
7. 2x 10k potenciometr s lineárním průběhem
8. velké servo s konstrukcí
9. střední servo
10. Klip na papíry o velikosti pera, se kterým budeme kreslit
11. Dřevěnou desku o rozměrech alespoň 30x40cm

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.

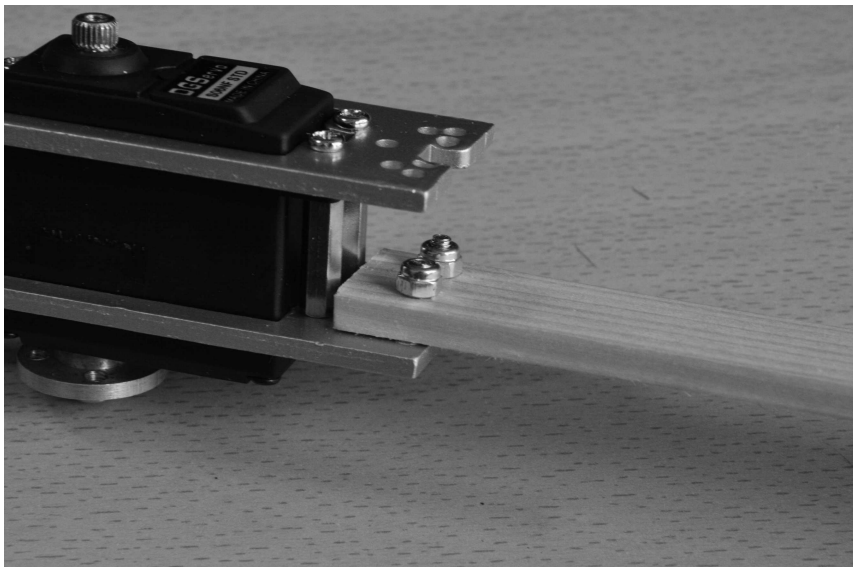


Obrázek 66.1: Menší rameno

V dalším kroku přišroubujeme vytvořené rameno ke konstrukci většího serva.

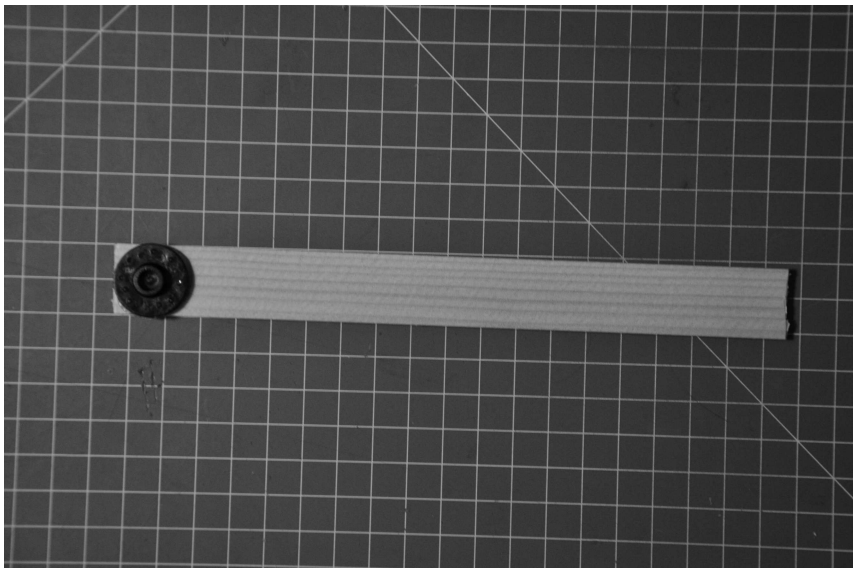


Obrázek 66.2: Kratší rameno s připevněnými motory

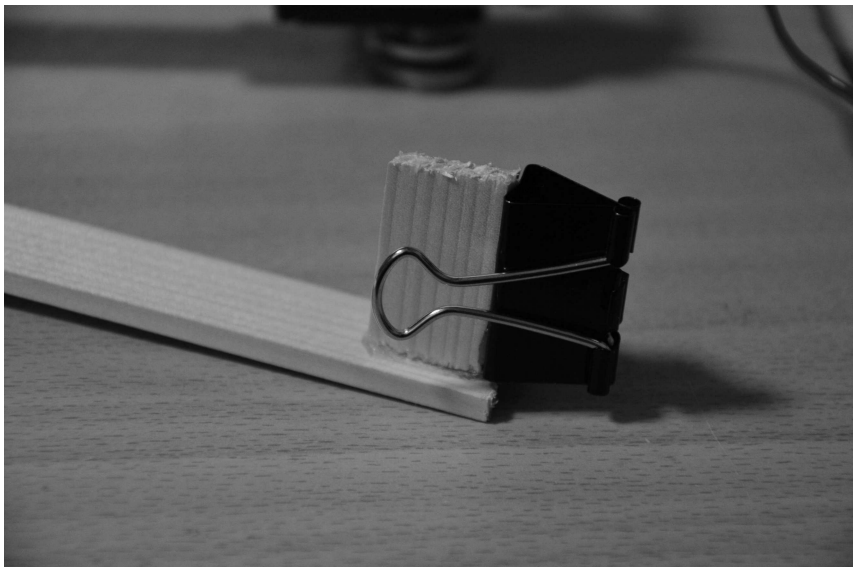


Obrázek 66.3: Detail

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.

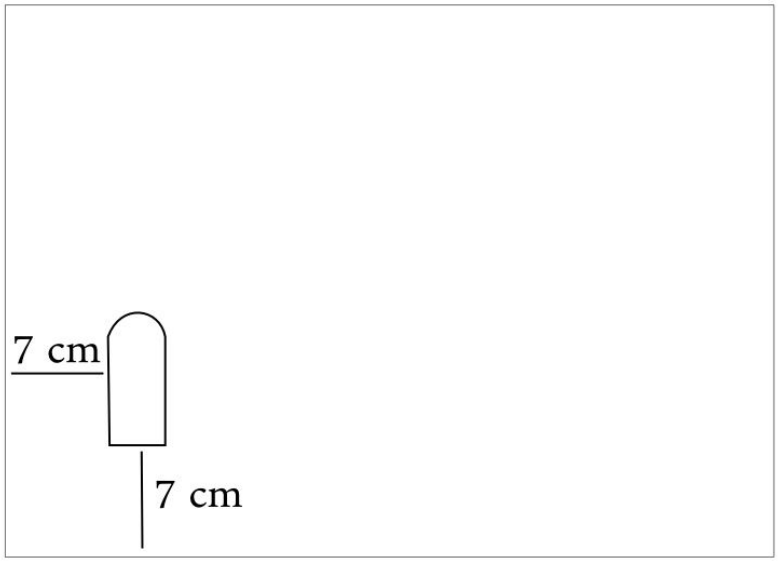


Obrázek 66.4: Delší rameno

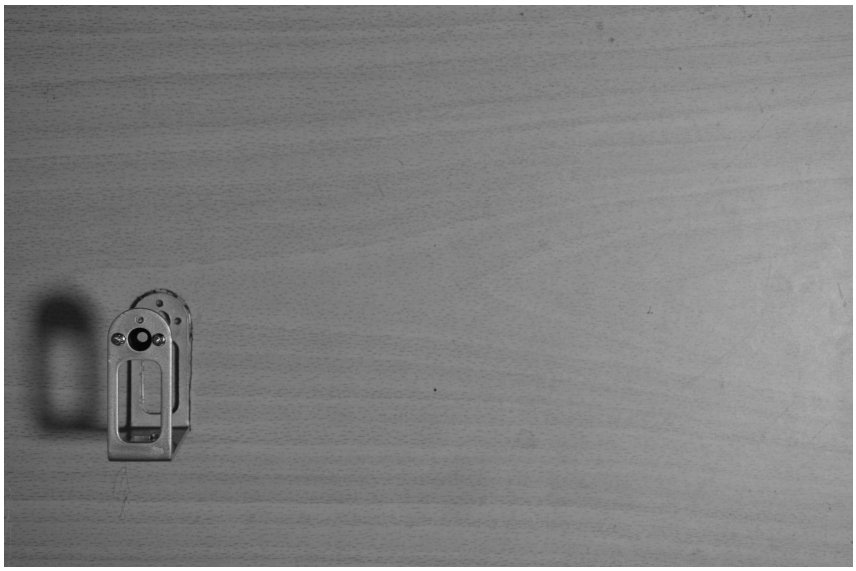


Obrázek 66.5: Detail

Na dřevěnou desku přilepíme, nebo přišroubujeme část konstrukce velkého serva (plastovou částí nahoru). Rozměry můžete vidět na obrázku.

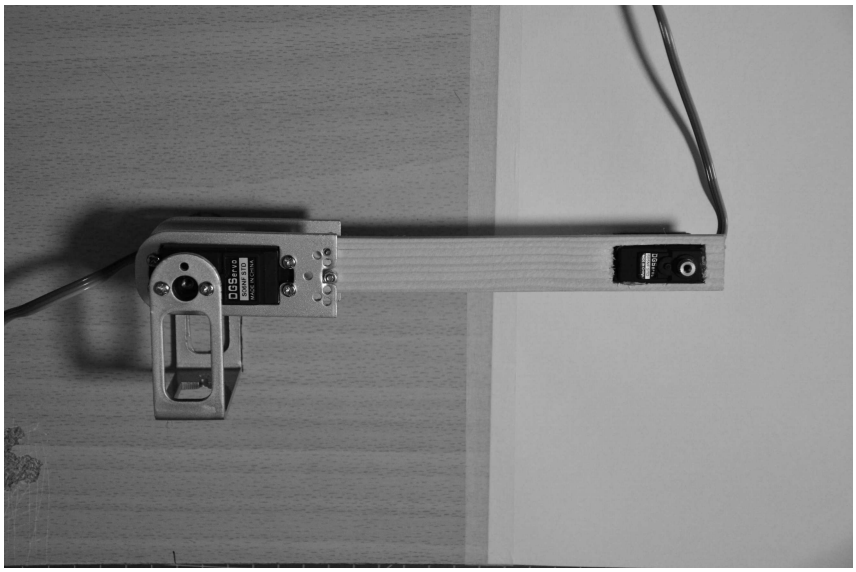


Obrázek 66.6: Nákres kreslící plochy



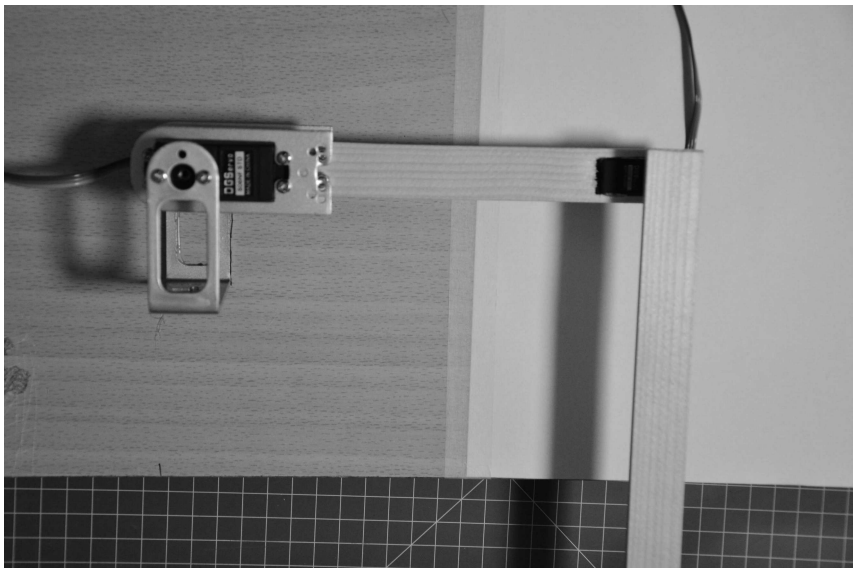
Obrázek 66.7: Foto kreslící plochy

Velké servo připojíme k Arduinu a funkcí .write() ho nastavíme na polohu 105. Poté nasadíme první rameno rovnoběžně s delším okrajem.



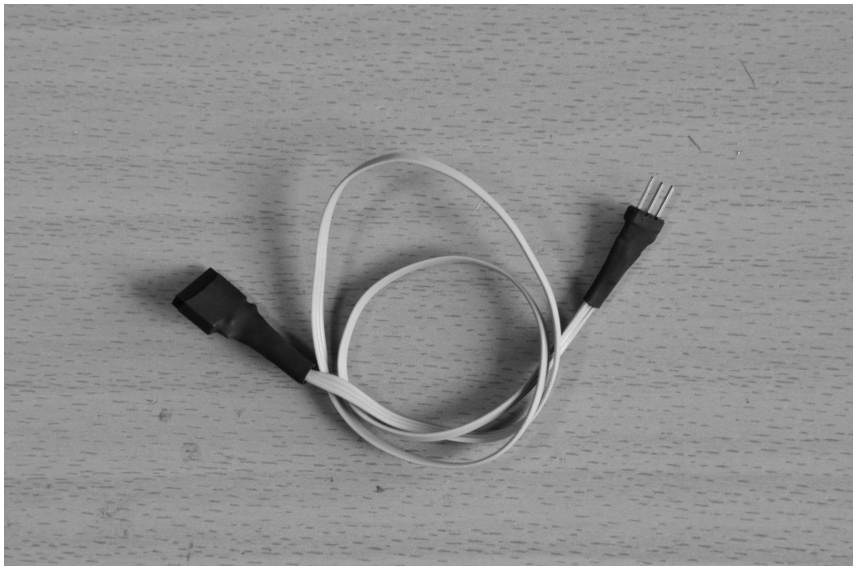
Obrázek 66.8: Připevnění kratšího ramene

Připojíme malé servo a nastavíme ho na hodnotu 60. Poté připojíme druhé rameno kolmo k prvnímu.



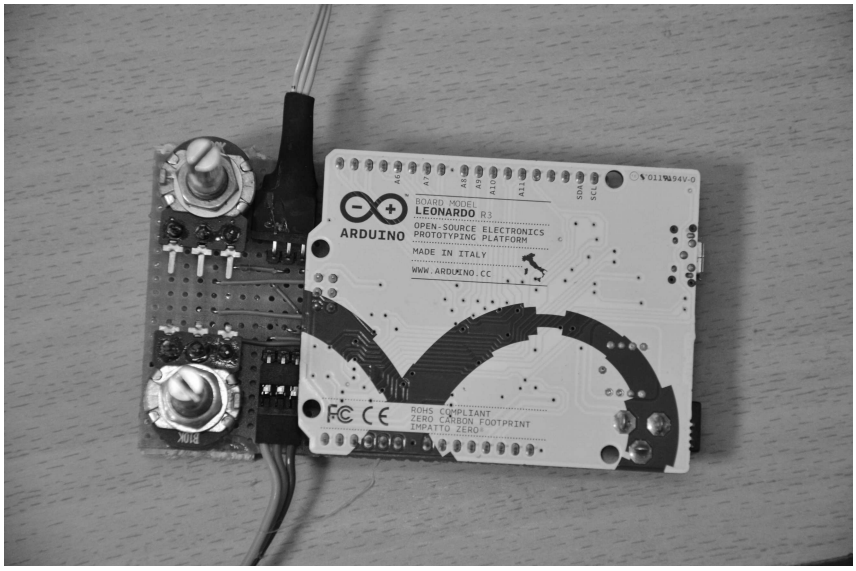
Obrázek 66.9: Připevnění delšího ramene

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 66.10: Propojovací kabel

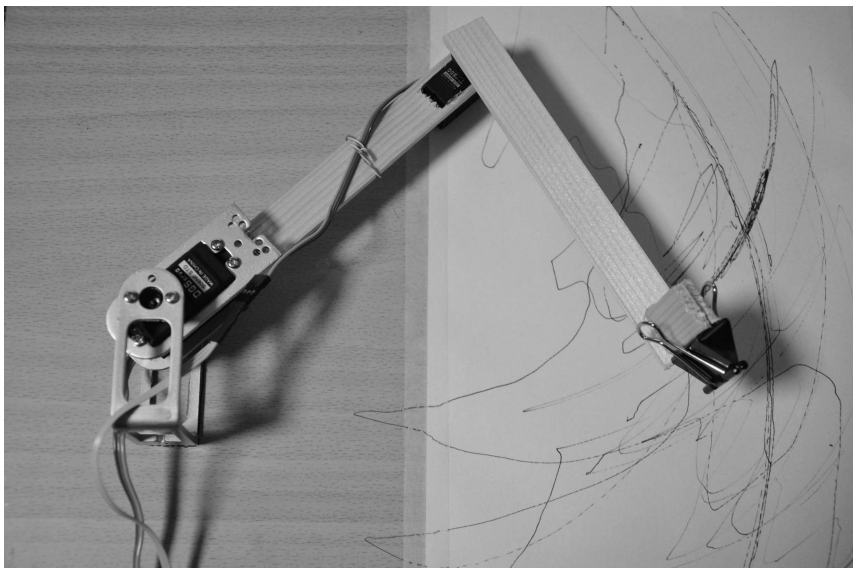
Nyní přichází na řadu zapojení. V tomto příkladu je použito následující: Velké servo je připojeno na pin A3, malé na A2. Také využijeme dva potenciometry, kterými budeme nastavovat polohu serv. Ty jsou připojeny k pinům A5 a A4. Výběr pinů však záleží čistě na nás. Na obrázku níže vidíte příklad konstrukce ovládání.



Obrázek 66.11: Ovládací deska

```
1  #include <Servo.h>
2
3  Servo sv, sm;
4  int h1, h2; //hodnoty mereni
5
6
7  void setup(){
8      sv.attach(A3);
9      sm.attach(A2);
10 }
11
12
13 void loop(){
14     h1 = map(analogRead(A5),0,1024,0,170);
15     h2 = map(analogRead(A4),0,1024,0,170);
16
17     sv.write(h1);
18     sm.write(h2);
19
20     Serial.println(h2);
21 }
```

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 66.12: 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 XVIII

WiFi shield

Kapitola 67

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.

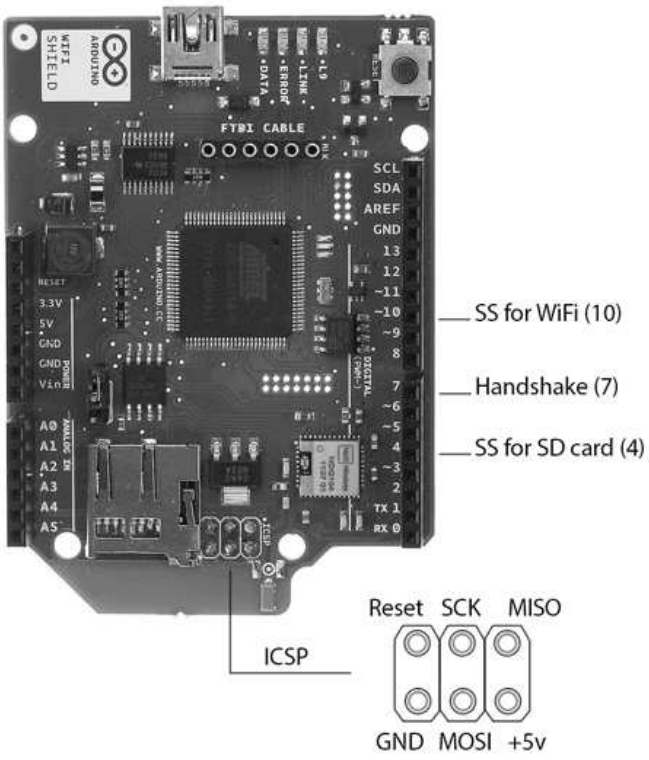
- WiFi shield se umí připojit k WiFi sítím se standardem 802.11b a 802.11g.
- Může se připojit k sítím bez hesla a také k těm se zabezpečením WEP a WPA2 Personal.
- Shield obsahuje slot na microSD kartu.
- S Arduinem shield komunikuje přes SPI rozhraní (stejně jako Ethernet Shield).
- SS pin WiFi shieldu 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.

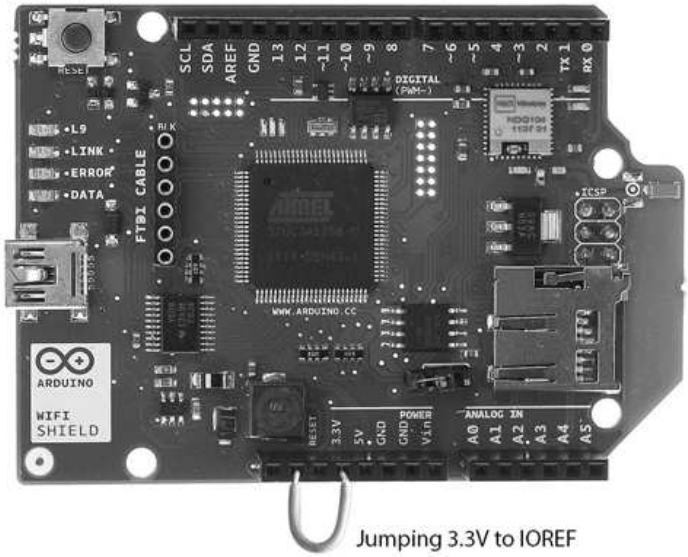
Název	Barva	Funkce
L9	žlutá	Je připojena na pin 9
LINK	zelená	Signalizuje připojení k WiFi
ERROR	červená	Signalizuje problém s komunikací
DATA	modrá	Signalizuje přenos dat

Pokud používáme shield s Arduinem starším než Arduino UNO rev3, musíme propojit 3.3V 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.



Obrázek 67.1: Piny WiFi shieldu



Obrázek 67.2: Propojení pro starší desky

Kapitola 68

Firmware shieldu

68.1 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.

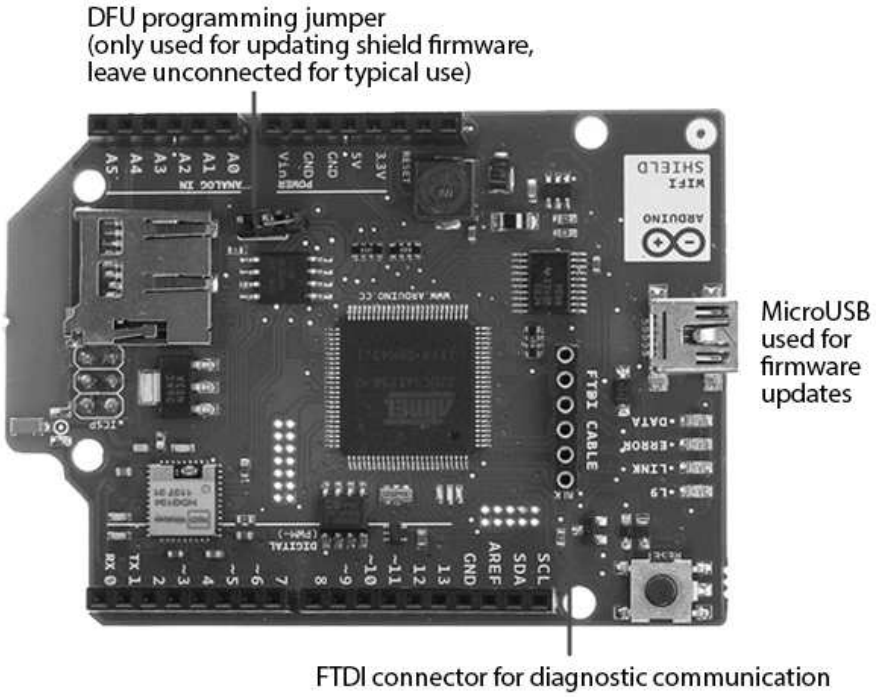
```
1  #include <SPI.h>
2  #include <WiFi.h>
3
4  void setup() {
5      Serial.begin(9600);
6      if (WiFi.status() == WL_NO_SHIELD) {
7          Serial.println("WiFi shield je nepripojen");
8          while(true); //zacykli program – ten dale nepokracuje
9      }
10     Serial.print("Verze firmware: ");
11     Serial.println(WiFi.firmwareVersion());
12 }
13
14 void loop() {}
```

Uvedený postup je použitelný pro Windows. Návod upgrade pro Mac a Linux naleznete zde.

Pokud verze firmware není 1.1.0 (nebo vyšší), je vhodné provést aktualizaci.

68.2 Aktualizace firmware

Před začátkem udpate musíme propojit dva piny, které umožní komunikaci s čipem AT32UC3 – mozkem procesoru. Jsou umístěny vedle microSD slotu. Procesor řídí chod čipu HDG104, který se stará o WiFi. Také bychom měli shield odpojit od Arduina.



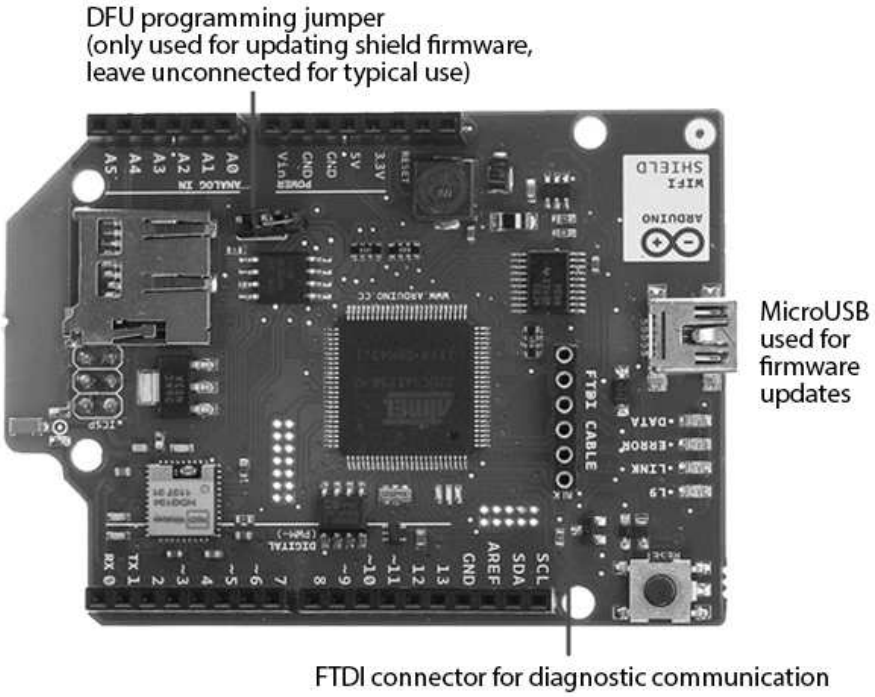
Obrázek 68.1: Konfigurační piny

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. Po stažení vhodné verze spustíme instalaci a bez jakékoliv nutnosti konfigurace jej nainstalujeme. Z Githubu stáhneme nejnovější verzi firmware. Zde nás budou zajímat soubory wifi_dnld.elf a wifiHD.elf, které stáhneme.

Poté spustíme příkazový řádek a přepneme se do složky, kde je nainstalovaný soubor FLIP (většinou C:Program Files (x86)AtmelFlip X.X.Xbin). 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 3.4.7bin
```

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.



Obrázek 68.2: Příkazový řádek

```
C:\>cd C:\Program Files (x86)\Atmel\Flip 3.4.7\bin
C:\Program Files (x86)\Atmel\Flip 3.4.7\bin>
```

Obrázek 68.3: Neznámé zařízení

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:UsersuzivatelDownloads.

```
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.

```
batchisp.exe -device AT32UC3A1256 -hardware usb -operation erase f memory flash blankcheck loadbuffer C:UsersuzivatelDownloadswifi_dnld.elf program verify start reset 0
```

Po úspěšném uploadu shield restartujeme, a poté nahrajeme i soubor wifiHD.elf.

```
batchisp.exe -device AT32UC3A1256 -hardware usb -operation erase f memory flash blankcheck loadbuffer C:UsersuzivatelDownloadswifiHD.elf program verify start reset 0
```

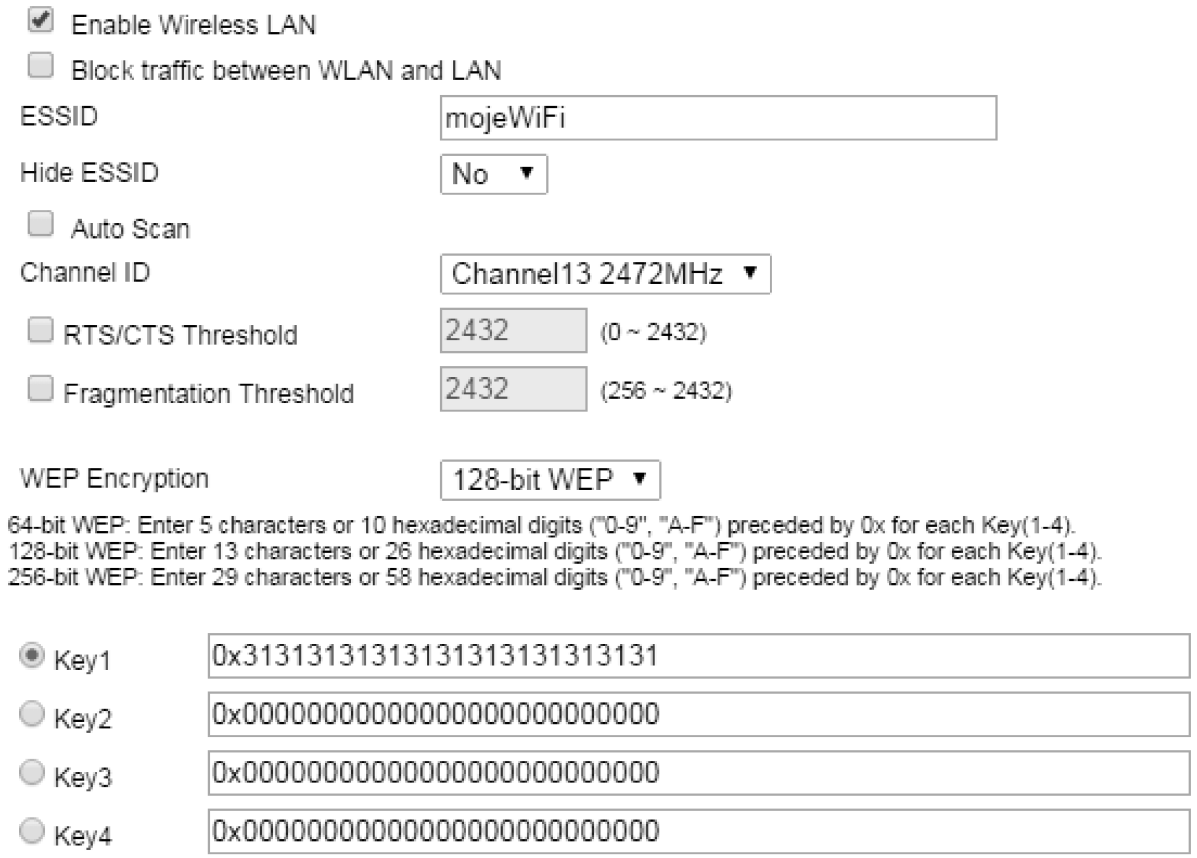
Tím je proces instalace aktuálního software u konce. Dva konektory, které jsme předtím spojili, teď rozpojíme a odpojíme shield od USB. Aktuální firmware by měl mít verzi 1.1.0 (nebo vyšší).

Kapitola 69

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 (key) a index klíče (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 69.1: Ukázka nastavení wep klíče

U sítí WPA2 Personal nám stačí SSID a heslo.

Kapitola 70

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.

70.1 Třída WiFi

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. Používejte, pokud 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];)
WiFi.RSSI()	WiFi.RSSI(); WiFi.RSSI(wifiAccessPoint);	Funkce funguje stejně jako .SSID(), pouze vrací sílu signálů sítě v dBm (decibel-miliwatt).
WiFi.encryptionType()	WiFi.encryptionType(); WiFi.encryptionType(wifiAccessPoint);	Stejné jako .SSID(), 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.

Třída, která obsahuje informace o serveru a funkce pro práci s ním.

70.2 Třída WiFiServer

Název	Zápis	Funkce
Server()	WiFiServer server(port);	Vytvoří WiFi server, který naslouchá 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.

70.3 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.
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 71

Příklady

Tímto jsme si představili funkce používané pro obsluhu WiFi shieldu. Nyní si můžeme předvést několik ukázek. Ty vycházejí z oficiálních příkladů z dokumentace.

71.1 Připojení k síti

Následující příkladu 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  }
```

71.2 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 v příkladu s Ethernet shieldem.

```
50 #include <SPI.h>
51 #include <WiFi.h>
52
53 char ssid[] = "SSID_site";
54 char pass[] = "heslo_site";
55
56 int status = WL_IDLE_STATUS;
57 WiFiServer server(80);
58
59 void setup() {
60     Serial.begin(9600);
61     pinMode(9, OUTPUT);
62
63     if (WiFi.status() == WL_NO_SHIELD) {
64         Serial.println("Shield nepripojen");
65         while(true);
66     }
67
68     while ( status != WL_CONNECTED) {
69         Serial.print("Pripojuji k SSID: ");
70         Serial.println(ssid);
71
72         status = WiFi.begin(ssid, pass);
73         delay(10000);
74     }
75     server.begin();
76     printWifiStatus();
77 }
78 void loop() {
79     //ceka na pripojeni klienta
80     WiFiClient client = server.available();
81
82     if(client){ //kdyz naleznem klienta
83         String currentLine = "";
84         while (client.connected()){
85             if (client.available()){
86                 char c = client.read();
87                 Serial.write(c);
88                 //pokud najde znak zalomeni radku (ukoncuje request od klienta)
89                 if (c == 'n'){
90                     if (currentLine.length() == 0) {
91                         client.println("HTTP/1.1 200 OK");
92                         client.println("Content-type:text/html");
93                         client.println();
94                         //hlavicka je oddelena znakem noveho radku
95
96                         //zde zacina HTML kod stranky
97                         client.print("<a href="/H">ZAPNI</a><br>");
98                         client.print("<a href="/L">VYPNI</a><br>");
```

```
                        client.println(); //dalsim prazdnym radkem konci HTML cast
                        break;
                    }
                    else{
                        currentLine = "";
                    }
                }
            else if (c != 'r') {
                currentLine += c;
            }

            //kontroluje, jestli request konci na H, nebo L
            if (currentLine.endsWith("GET /H")) {
                digitalWrite(9, HIGH); //zapne LED
            }
            if (currentLine.endsWith("GET /L")) {
                digitalWrite(9, LOW); //vypne LED
            }
            }
        }
            client.stop();
            Serial.println("client disonnected");
        }
}

void printWifiStatus() {
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");

  Serial.print("http://");
  Serial.println(ip);
}
```

Tyto základní příklady, společně se znalostmi zmíněnými v části o Ethernet shieldu, slouží jako odrazový můstek pro další tvorbu.

Část XIX

Zdroje obrázků

Arduino Mini
Arduino Nano
Arduino Micro
LilyPad Arduino
Arduino Fio
Arduino Uno
Arduino Leonardo
Arduino Yún
Arduino Mega2560
Arduino Esplora
Arduino Robot
Arduino Intel Galileo
Arduino Tre
Ethernet Shield

Arduino Pro s převodníkem

Digital Read - INPUT
Digital Read - INPUT_PULLUP
PWM

kosinusoida
sinusoida
tangentoida

zvuková stopa
sinusoida
Squarewave
Sedmisegmentový display
+-1
šestnáctisegmentový display

Arduino Leonardo
Arduino Esplora
Esplora Pinout

Sensoduino

RGB LED matrix
LED matrix

Zapojení znakového LCD

Rozložení pinů ATtiny85
Rozložení pinů ATmega168

Datasheet keypadu

Piny shieldu
Propojení pro starší desky
Konfigurační piny