

# Jak začít programovat čipy (ESP32)

Jaroslav Páral  
Jakub Streit  
Rudolf Hlaváček  
a další



# ROBOTÁRNA

[www.robotarna.cz](http://www.robotarna.cz)

za kolektiv autorů  
Miroslav Burda  
editor

[dokumentace@robotikabrno.cz](mailto:dokumentace@robotikabrno.cz)

verze 0.6

# Obsah

<b>1</b>	<b>Začínáme</b>	<b>4</b>
1.1	Cíl: první robot . . . . .	4
1.1.1	Týmy pro stavbu robotů . . . . .	5
1.1.2	Postup návrhu robota . . . . .	5
1.1.3	Co potřebujete na začátku . . . . .	6
1.1.4	První projekt . . . . .	7
1.1.5	Základní pojmy – čipy . . . . .	9
<b>2</b>	<b>Programovací jazyk C++</b>	<b>11</b>
2.1	Základy syntaxe v jazyce C . . . . .	11
2.1.1	Základní pojmy . . . . .	11
2.1.2	Výrazy a operátory . . . . .	12
2.1.3	Nejčastější příkazy C++ . . . . .	14
2.1.4	Funkce a procedury . . . . .	20
2.1.5	Knihovny . . . . .	22
2.2	Příkazy C++ pro čipy . . . . .	22
2.3	Příklady programů v C++ . . . . .	22
2.3.1	Blikání LED . . . . .	22
2.3.2	LED zapínaná tlačítkem . . . . .	23
2.3.3	Nejjednodušší PWM . . . . .	23

<b>3</b>	<b>Elektronika</b>	<b>25</b>
3.1	Základní součástky . . . . .	25
3.1.1	Rezistor . . . . .	25
3.1.2	Kondenzátor . . . . .	26
3.1.3	Dioda . . . . .	26
3.1.4	LED . . . . .	26
3.1.5	Tranzistor . . . . .	27
3.1.6	Cívka . . . . .	27
3.1.7	Driver . . . . .	27
3.1.8	Stabilizátor . . . . .	27
3.1.9	Krystal . . . . .	28
3.2	Základní veličiny v elektronice . . . . .	28
3.2.1	Proud . . . . .	28
3.2.2	Napětí . . . . .	29
3.2.3	Odpor . . . . .	29
3.2.4	Výkon . . . . .	29
3.2.5	Výpočet tepelného výkonu (ztrátového) . . . . .	29
3.2.6	Výpočet rezistoru pro LED . . . . .	30
3.3	Motorky, serva a PWM . . . . .	31
3.3.1	Motorky . . . . .	31
3.3.2	PWM . . . . .	32
3.3.3	Řízení serva . . . . .	32
3.4	ALKS, další desky a další zařízení . . . . .	32
3.4.1	ESP 32 . . . . .	32
3.4.2	ALKS . . . . .	33
3.4.3	Bluetooth . . . . .	33
3.4.4	Připojení k počítači pomocí bluetooth . . . . .	33
3.5	Desítková soustava a dvojková soustava . . . . .	34
3.5.1	Desítková soustava . . . . .	34
3.5.2	Dvojková soustava . . . . .	34
3.5.3	Převod z desítkové soustavy do dvojkové . . . . .	34
3.5.4	Převod z dvojkové soustavy do desítkové . . . . .	36
3.5.5	Bitové výrazy a práce s nimi . . . . .	36

<b>4</b>	<b>Software</b>	<b>42</b>
4.1	L <sup>A</sup> T <sub>E</sub> X . . . . .	42
4.1.1	Proč používat L <sup>A</sup> T <sub>E</sub> X . . . . .	42
4.1.2	Editor . . . . .	42
4.2	Github . . . . .	43
4.2.1	Proč a jak se používá . . . . .	43
4.2.2	Instalace githubu a stažení repozitáře . . . . .	43
4.3	Další software . . . . .	43
4.3.1	Proficad . . . . .	43
4.3.2	Linux . . . . .	44
<b>5</b>	<b>Ostatní</b>	<b>46</b>
5.1	Mechanická konstrukce robota – doporučení . . . . .	46
5.1.1	Konstrukce velkých robotů . . . . .	47
5.2	Řešení některých problémů . . . . .	47
5.2.1	Chyba při uploadu programu do desky Arduino nano: . . . . .	47
5.3	Příloha A – Hodnoty vybraných součástí . . . . .	48
5.4	Příloha B – Poznámky a vize . . . . .	49
5.4.1	Přerušení . . . . .	49
5.4.2	Baterie . . . . .	49
5.4.3	Vize – co přidat do textu . . . . .	49

# Kapitola 1

## Začínáme

### 1.1 Cíl: první robot

**Tento** text je určen pro začátečníky a mírně pokročilé v oblasti stavby autonomních robotů – převážně středoškoláky v prvním ročníku, kteří se pokoušejí postavit svého prvního **autonomního** robota, nejčastěji pro nějakou soutěž<sup>1</sup>. Protože byl sepsán na pracovišti Robotárna<sup>2</sup> (pobočka DDM Helceletka Brno), jsou některé části určené především členům jeho kroužků. Ale většina textu je použitelná všeobecně.

Každý robot se musí

- vyrobít (mechanická část – konstrukce)
- osadit elektronikou, motory a pod.
- naprogramovat

Přitom můžou nastat zhruba dvě situace:

- Nemám žádné znalosti a zkušenosti → doporučený postup je začít ve školním kroužku nebo samostatně stavět a programovat robota z *Lego Mindstorms*<sup>3</sup>. Je to daleko nejjednodušší a nejrychlejší cesta, omezení jste cenou stavebnice a jejími možnostmi.

---

<sup>1</sup>Robotiáda v Brně: <http://www.robotiada.cz/>, Robotický den v Praze <http://robotickyden.cz/>

<sup>2</sup><http://www.helceletka.cz/robotarna>

<sup>3</sup><https://www.lego.com/cs-cz/mindstorms>

JP:  
Testovací  
poznámka  
k prv-  
nímu  
slovu ve  
větě -  
autor  
Jarek  
Páral  
(JP)

- Už jsem někdy něco naprogramoval nebo zapojil nebo postavil a chci jít dál → potom je pro vás určen tento text. Jednotlivé kapitoly se věnují různým oblastem, které je postupně potřeba zvládnout (nebo jimi pověřit jiného člena týmu) s důrazem na začátečnické problémy.

### 1.1.1 Týmy pro stavbu robotů

Už bylo zmíněno, že stavba robotů zahrnuje tři propojené, ale relativně nezávislé okruhy: návrh a výrobu mechanické konstrukce, návrh a zapojení elektroniky a programování. Proto je dobré roboty stavět v týmech, kde se jednotliví členové zaměřují na tyto oblasti a navzájem se doplňují. Navíc každý tým potřebuje řadu pomocných činností (nákup součástek, vyhledávání údajů na internetu a pod.). Je dobré mít proto v každém týmu ještě pomocníka, který podporuje ostatní a umožňuje jim soustředit se na jejich hlavní úkoly.

Úplně ideální potom je, když každou funkci v týmu zastávají dva lidé, takže se mohou vzájemně zastupovat. Tým by potom měl celkem osm členů – dva mechaniky, dva elektroniky, dva programátory a dva pomocníky. To se ale v praxi téměř nikdy nepodaří. Často nastane právě opačný případ, kdy tým má pouze dva nebo tři členy, kteří se o všechny činnosti musí nějak podělit. V každém případě ale platí, že je výhoda, pokud lidé v týmu znají i věci mimo jejich „hlavní obor“, tj. když například programátor zná základy elektroniky.

### 1.1.2 Postup návrhu robota

1. stanovit, co by měl robot splnit – přibližně
2. sepsat, co by měl robot splnit – podrobně (klidně i více stran A4), z toho vyplýne
3. zjištění, jaké senzory a pohony robot potřebuje a kde budou na robotovi umístěné
4. návrh první konstrukce robota včetně umístění senzorů a pohonů
5. zprovoznění toho všeho – **hlavní cíl tohoto textu**

Začátečníci obvykle tuto posloupnost nedodrží a pak staví mechaniku pro 3 a více robotů, než zjistí, že je to dost práce navíc. A taky dost času navíc, který potom před soutěží může chybět.

### 1.1.3 Co potřebujete na začátku

#### Hardware:

- notebook nebo počítač s operačním systémem Windows 7 a novějším nebo Linux (pro starší počítače např. aktuální distribuci Xubuntu, Ubuntu, Mint)
- desku s čipem, např. DevKit ESP32<sup>4</sup>
- vývojovou desku ALKS<sup>5</sup>
- výhledově cokoli dalšího, co chcete tím čipem řídit (serva, ultrazvuk, senzory a motory všeho druhu ... )

#### Software:

- *Visual Studio Code* – viz kapitola 1.1.4
- *PlatformIO* – viz kapitola 1.1.4

Veškerý zde popisovaný a doporučený software je freeware.

#### Znalosti:

- běžná práce se soubory ve vašem operačním systému (hledání, kopírování, mazání, ... )
- základy programování C++ (viz kapitola 2.1.1) a příkazy C++ pro čipy (viz kapitola 2.2)
- velmi se hodí schopnost porozumět textu psanému v jednoduché angličtině)
- základy elektroniky – viz kapitola 3

#### Taky se hodí vědět, že:

- veškeré **zelené** a **modré** odkazy a slova jsou proklikávací (s výjimkou barevného zvýraznění syntaxe<sup>6</sup> ve zdrojových textech jazyka C++)
- na konci textu je (klikací) obsah
- těsně před obsahem je (také klikací) rejstřík

---

<sup>4</sup>varianta: jinou vývojovou desku, např. Arduino Uno, Arduino nano, ...

<sup>5</sup><https://github.com/RoboticsBrno/ArduinoLearningKitStarter>  
**NUTNÁ AKTUALIZACE** Hotová deska vás hlavně ze začátku zbavuje nutnosti vědět, co si můžete dovolit kam připojit a hlavně nutnosti vůbec nějaké připojování čehokoliv řešit.

<sup>6</sup>syntaxe – způsob zápisu daného jazyka, barevný zápis je mnohem přehlednější

- v žádném textu o elektronice a robotice nemůže být všechno, pokud zde něco nenajdete, použijte např. Google

### 1.1.4 První projekt

#### Postup

Pokud s programováním čipů začínáme, čekají nás tyto úkoly:

1. nainstalovat prostředí *Visual Studio Code*
2. do VS Code nainstalovat tzv. *PlatformIO*
3. založit nový projekt
4. napsat zdrojový kód, přeložit a dostat jej do čipu

Toto vše podrobněji probereme na dalších řádcích.

#### Nainstalovat Visual Studio Code

*Visual Studio Code* (zkráceně VS Code) je něco jako textový editor, speciálně navržený pro programátory čehokoliv. Instalujte podobně jako každý jiný program, stahujte zde: <https://code.visualstudio.com/>

#### Nainstalovat PlatformIO

PlatformIO je ten software, který umožní program v C++ přeložit tak, aby ho čip pochopil a taky ho do čipu umí nahrát. Instalace podle návodu zde: <http://docs.platformio.org/en/latest/ide/vscode.html#installation>

#### Založit nový projekt

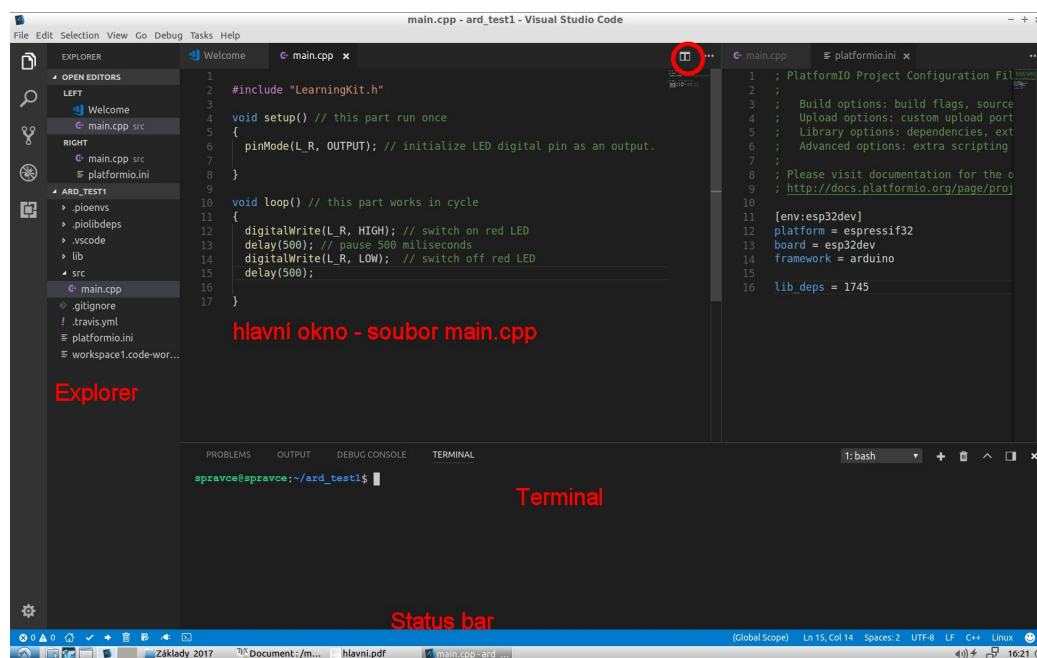
Program (ne)píšete jen do jednoho souboru, ale aby vše fungovalo, potřebujete povícero dalších souborů, které dohromady tvoří tzv. **projekt**. Tyto soubory jsou mezi sebou hodně provázané a navíc vázané na konkrétní místo uložení, takže vám po překopírování na jiný počítač pravděpodobně program nepůjde přeložit a nebo nahrát do čipu. Řešení: uložit celý projekt na flešku. Na novém počítači založit nový projekt a potřebné části zdrojového kódu kopírovat pomocí Ctrl+C, Ctrl+V.



1. Založte nový projekt z ikony „domeček“ – viz <http://docs.platformio.org/en/latest/ide/vscode.html#quick-start>.
2. Do kolonky *Board* se musí vybrat správná deska. Desek je přes 400 a jsou rozdělené do sekcí řazených podle abecedy vyznačených šedivou barvou. V sekcích jsou potom jednotlivé desky řazené taky abecedně. Pro čip ESP32 najdete sekci *Espressif32* a v ní položku *Espressif ESP32 Dev Module*. Kolonka *Framework* se potom vyplní automaticky.
3. Zbývá vybrat adresář, do kterého bude projekt uložen. Tento adresář si předem vytvořte, s adresářem vytvářeným za pochodu má VS Code kdovíproč problém. Odškrtněte zatržítko *Use default folder* a zvolte vytvořený adresář.

Pro Linux Lubuntu: projekt musí být uložen na pevném disku, ne na flešce, jinak prostě nepojede, netuším proč.

## Napsat zdrojový kód, přeložit a dostat jej do čipu



Obrázek 1.1: Rozložení oken v programu *VS Code*

Obrázek 1.1 na straně 8 ukazuje rozložení oken v rámci projektu. Hlavní okno rozdělíte na dvě části pro zobrazení dvou upravovaných souborů pomocí ikony v kroužku. Začínáme v okně **Explorer**, kde je umístěna adresářová struktura projektu<sup>7</sup>. Otevřete soubory *platformio.ini* a v adresáři *src* soubor *main.cpp*.

Pro pohodlnou práci s deskou ALKS byla napsaná tzv. knihovna *LearningKit*. Aby fungovala, musí být do souboru *platformio.ini* dopsán řádek `lib_deps = 1745` (bez mezery na začátku řádku) a do záhlaví souboru *main.cpp* doplňte `include "LearningKit.h"`. Dále dopište do souboru *main.cpp* kód, který bliká červenou LED. Vše je vidět na obrázku 1.1. Celý zdrojový kód tohoto prvního programu (obsah souboru *main.cpp*) je uveden v kapitole 2.3.

Teď budou potřeba další dvě části VS Code: terminál (okno vpravo dole) a stavový řádek (Status bar – proužek pod terminálem). Na stavovém řádku klikněte na ikonu šipky<sup>8</sup> (pátá zprava) a *PlatformIO* se pokusí váš program přeložit a nahrát do čipu. Pokud chcete program pouze přeložit, klikněte na ikonu zatržítka<sup>9</sup> hned vedle.

Při prvním pokusu nahrát program do čipu na Linuxu může mít *PlatformIO* problém, že nenajde USB spojení na desku s čipem a vyžaduje ho doinstalovat. Zpráva<sup>10</sup> se objeví se v terminálu včetně nápovědy,<sup>11</sup> jak to udělat. Nápověda je ale tak podrobná, že to středně poučený linuxový laik s pomocí internetu zvládne. Při všech dalších překladech už to nebude problém.

Další programy budou uvedeny v kapitole 2.3.

### 1.1.5 Základní pojmy – čipy

#### Datasheet

**Datasheet** je dokument, ve kterém jsou detailně popsány vlastnosti a možnosti dané elektronické součástky. Každá součástka má svůj datasheet.

---

<sup>7</sup>Pokud toto okno není vidět, zobrazíte ho v menu *View* položka *Explorer*

<sup>8</sup>totéž provede Ctrl+Alt+U

<sup>9</sup>totéž provede Ctrl+Alt+B

<sup>10</sup>Warning! Please install ‘99-platformio-udev.rules‘

<sup>11</sup><https://raw.githubusercontent.com/platformio/platformio/develop/scripts/99-platformio-udev.rules>

Datasheet pro každou součástku je možné najít na internetu<sup>12</sup> nebo na stránkách výrobce nebo prodejce součástky. Bohužel jsou všechny datasheety anglicky.

## Mikroprocesor, mikrokontrolér

**Mikroprocesor, mikrokontrolér, čip** znamenají totéž – integrovaný obvod, který se snažíme naprogramovat, aby řídil robota nebo jeho část.

**Pin** je vývod (nožička) čipu. Jednoduché čipy (např. ATtiny) mají osm pinů, složitější čipy mají 32, 40, nebo také 100 pinů.

Pin může být nastavený jako vstupní nebo jako výstupní.

Pokud je pin nastavený jako **vstupní**, umí určit, zda na něm je napětí odpovídající logické jedničce (5 V nebo 3,3 V podle typu čipu) nebo logické nule (0 V). Také umí přechytit, jaké je na něm analogové napětí.

Pokud je pin nastavený jako **výstupní**, umí se nastavit na logickou jedničku nebo logickou nulu.

## Bity a bajty

Každá číslice ve dvojkové soustavě reprezentuje jeden **bit** (nabývá hodnot 0 nebo 1).

Osm bitů dohromady tvoří **bajt**. Nejnižší bit v bajtu leží vpravo (tzv. **nultý bit**), další je nalevo od něj (první bit), až do sedmého bitu, který je nejvíce vlevo.

---

<sup>12</sup>například na stránce [www.datasheetcatalog.com](http://www.datasheetcatalog.com)

# Kapitola 2

## Programovací jazyk C++

### 2.1 Základy syntaxe v jazyce C

#### 2.1.1 Základní pojmy

Upozornění: V jazyce C++ se **rozlišují velká a malá písmena**.

**příkaz** – povel pro procesor, říká procesoru, co má dělat. Příkazy probíhají postupně, nejprve se udělá jeden a pak další, který je na řadě. Za příkazy dáváme středník (;).

**syntaxe** – „pravopis“ programovacího jazyka, říká, jak psát příkazy programovacího jazyka tak, aby nám počítač rozuměl

**preprocesor** – před samotným překladem programu do *\*.hex* souboru proběhnou tzv. direktivy preprocesoru (něco jako příkazy, podrobně .

Příklady: `#include <>` – pro vkládání hlavičkových souborů

`#include ""` – pro vkládání vlastních hlavičkových souborů

`#define KONSTANTA HODNOTA_KONSTANTY` – pro definici konstanty, všude kde se v kódu vyskytne text `KONSTANTA` bude tento text nahrazen `HODNOTA_KONSTANTY`

**proměnná** – je místo v paměti, kde jsou uložena nějaká data

**datový typ** – říká nám, v jaké podobě jsou data v proměnné uložena. V proměnné jsou pouze jedničky a nuly, Pomocí datového typu mikrokontrolér pozná, jestli ta data jsou čísla, nebo znak, v jakém rozsahu jsou čísla, atd... používané datové typy:

`uint8_t` – celá čísla od 0 do 255  
`int8_t` – celá čísla od -127 do 127  
`uint16_t` – celá čísla od 0 do 65535  
`int16_t` – celá čísla od -32767 do 32767  
`uint32_t` – celá čísla od 0 do 4294967296  
`int32_t` – celá čísla od -2147483647 do 2147483647

**operandsy** – jsou čísla nebo výrazy, které „vstupují do operace“

**operátory** – nám říká, jaké operace provedeme s operandy (co s nimi uděláme)

Příklad: `5 + 2`, přitom 5 a 2 jsou operandy a znaménko plus je operátor, který nám říká, že čísla chceme sečíst (provést operaci sečítání)

**poznámka** – cokoliv se objeví v kódu mezi znaky `/*` a `*/`, tak je poznámka a bude před překladem odstraněno z kódu.

Pokud chceme zapoznámkovat pouze jeden řádek, použijeme `//`

Příklady:

```
/* Vse co je napsano zde  
je poznamka a nebude prekladano.  
*/
```

```
// Toto je poznamka, ktera popisuje funkci kodu
```

## 2.1.2 Výrazy a operátory

Výraz je něco co nabývá nějaké hodnoty, např.: `5 + 3` je výraz, který nabývá hodnoty 8. `5 > 3` je výraz nabývající hodnoty `true` neboli pravda, `x <= 10` je výraz který je pravdivý, pokud proměnná `x` (kterou musíme mít deklarovanou) menší nebo rovna číslu 10.

### Aritmetické operátory

+ sčítání  
- odčítání  
\* násobení  
/ dělení

## Operátory inkrementace a dekrementace

pokud chceme zvýšit hodnotu proměnné o jedničku, napíšeme `++název_proměnné`;  
pokud chceme snížit hodnotu proměnné o jedničku, napíšeme `--název_proměnné`;  
Příklad:

```
uint8_t i = 5; // Vytvoření proměnné i a dosazení hodnoty 5 do ní.  
uint8_t j = 42; // Vytvoření proměnné j a dosazení hodnoty 42 do ní.  
  
++i; // Hodnota i se zvýší o jedničku na číslo 6.  
--j; // Hodnota j se sníží o jedničku na číslo 41.
```

## Logické výrazy a operátory

Přehled logických operátorů:

- `==` porovnání - rovnost
- `!=` porovnání - nerovnost
- `&&` logický součin
- `||` logický součet
- `!` negace
- `<` menší než
- `<=` menší nebo rovno
- `>` větší než
- `>=` větší nebo rovno

**Logický výraz** může nabývat pouze dvou hodnot – pravda (true) a nepravda (false). Obvykle je nepravda reprezentovaná nulou a pravda každým nenulovým číslem. Logické výrazy se používají v podmínkách (viz např. [2.1.3](#))

.

**Logické operace** se používají při vyhodnocování logických výrazů.

Příklady:

- `a == b` výraz je pravdivý, pokud se `a` rovná `b`
- `a != b` výraz je pravdivý, pokud se `a` nerovná `b` lze to napsat i konkrétněji, např.: `a != 5`, výraz je pravda pokud se `a` nerovná 5, pokud se rovná výsledkem výrazu je nepravda(false)
- `b > c` výraz je pravdivý, pokud je `b` větší než `c`
- `b >= d` výraz je pravdivý, pokud je `b` větší nebo rovný `d`

`!(a > b)` výraz v závorce je pravdivý, pokud je `a` větší než `b`, ale pak je negováno (z pravdy se stává nepravda a naopak), tj. celý výraz je nepravdivý, pokud je `a` větší než `b`, pokud je `a` menší nebo rovno, tak je výraz pravdivý.

`!(a == b)` výraz `(a == b)` je negován znaménkem `!`, to znamená, že výraz je pravdivý, pokud se výraz `a` nerovná výrazu `b`, v podstatě se to dá napsat i takto: `a != b`

Pozor: negace výrazu `(a > b)`, tj. `!(a > b)` není to samé jako výraz `(a < b)`, ale správně je to `(a <= b)` <sup>1</sup>

Logický součin `&&` se používá, pokud budeme potřebovat spojit dva nebo více výrazů dohromady, např.: `(a > b) && (c == d)`, výsledkem tohoto výrazu bude logický součin výrazů v závorkách, pro logický součin platí, že je pravda pokud oba výrazy jsou pravdivé, jinak je výsledek nepravda, tj. zde bude pravda pouze pokud bude `a` větší než `b` a zároveň bude platit, že `c` se rovná `d`. Logický součin použijeme, pokud musí všechny výrazy být pravda. Logický součet `||` je pravdivý, pokud alespoň jeden výraz je pravdivý. Např.: `(e <= f) || (g != 3)` výraz bude pravda, pokud bude platit, že `e` je menší nebo roven `f`, nebo bude platit, že `g` se nerovná 3, anebo klidně budou platit oba výrazy.

### 2.1.3 Nejčastější příkazy C++

#### Přiřazovací příkaz

Použijeme, pokud chceme, aby se do proměnné uložila nějaká data.

Syntaxe:

```
datový_typ název_proměnné;
```

syntaxe vkládání:

```
název_proměnné = hodnota_která_se_má_uložit;
```

Data do proměnné můžeme vložit rovnou při vytváření proměnných:

```
datový_typ název_proměnné = hodnota;
```

Příklad: `uint8_t b = 5;` Vytvoří se proměnná pojmenovaná `b` (která je v rozsahu od 0 do 255) a uloží se do ní číslo 5.

---

<sup>1</sup>Platí zde určité zákonitosti viz De Morganovy zákony

`int16_t B;` Vytvoří se proměnná B. B není to samé jako b, protože jazyk C rozlišuje VELKÁ a malá písmena.

`B = 1024;` Do proměnné B se uloží hodnota 1024;

## Blok příkazů

Pokud chceme někam dát více příkazů, ale můžeme tam dát pouze jeden příkaz, tak je dáme do složených závorek Příklad:

```
{  
    PŘÍKAZ1;  
    PŘÍKAZ2;  
    PŘÍKAZ3;  
}
```

## Podmiňovací příkaz (if)

Použijeme, pokud chceme, aby se program mohl rozhodnout na základě nějaké podmínky.

syntaxe:

```
if(PODMÍNKA)  
    PŘÍKAZ1;  
else  
    PŘÍKAZ2;
```

Pokud platí `PODMÍNKA` v kulaté závorce, vykoná se `PŘÍKAZ1`, pokud neplatí, vykoná se `PŘÍKAZ2`. Za `if` nebo `else` může být pouze jeden příkaz, pokud jich tam chceme dát více, použijeme blok. Větev `else` je nepovinná.

Příklad:

```
if(a > 5) // Pokud platí, že  
           //proměnná a je větší než 5, tak se  
{  
    b = a; // do proměnné b uloží hodnota, která je v proměnné a  
    c = 2; // do proměnné c se uloží číslo 2.  
} // Pokud podmínka neplatí, tak se nevykoná nic.
```



podmínka může být výraz např.:

```
a > 5 // podmínka bude platit,  
      // pokud proměnná a je větší než 5
```

podmíněné příkazy lze vnořovat - v bloku příkazů může být další if Příklad:

```
if(PODMÍNKA1)  
{  
    PŘÍKAZ1;  
    if(PODMÍNKA2)  
    {  
        PŘÍKAZ2;  
    }  
    PŘÍKAZ3;  
}  
else  
{  
    PŘÍKAZ4;  
}
```

Pokud platí `PODMÍNKA1` tak se vykoná `PŘÍKAZ1`, pak se zkontroluje `PODMÍNKA2`, pokud platí, tak se vykoná `PŘÍKAZ2`, pokud ne, program pokračuje dál a vykoná `PŘÍKAZ3`. V případě, že neplatí ani `PODMÍNKA1`, tak se blok příkazů přeskočí a vykoná se blok příkazů za `else`, tj. `PŘÍKAZ4`.

Je tu možnost také do větve `else` napsat další `if else`, vypadá to takto:

```
if(PODMÍNKA1)  
{  
    PŘÍKAZ1;  
}  
else  
{  
    if(PODMÍNKA2)  
    {  
        PŘÍKAZ2;  
    }  
    else  
    {
```

```

        PRÍKAZ3;
    }
}

```

Pokud platí PODMÍNKA1, tak se vykoná PŘÍKAZ1, pokud ne, tak se zkontroluje PODMÍNKA2, pokud platí tak se vykoná PŘÍKAZ2, pokud neplatí, tak se vykoná PŘÍKAZ3. Výše zapsaný kód sice bude fungovat, ale není přehledný, a proto z důvodu lepší orientace v kódu zvolíme tento zápis:

```

if(PODMÍNKA1)
{
    PRÍKAZ1;
}
else if(PODMÍNKA2)
{
    PRÍKAZ2;
}
else
{
    PRÍKAZ3;
}

```

### **Příkaz několikanásobného větvení (switch)**

Pokud by takto zapsaných podmínek bylo moc a vztahovali by se k jedné řídicí proměnné lze použít příkaz několikanásobného větvení neboli switch. Tak například místo zápisu:

```

if(a == 1)
{
    PŘÍKAZ1;
}
else if(a == 2)
{
    PŘÍKAZ2;
}
else if(a == 3)
{

```

```

    PŘÍKAZ3;
}
else if(a == 4)
{
    PŘÍKAZ4;
}
else
{
    PŘÍKAZ5;
}

```

Lze napsat:

```

switch(a)
{
    case 1:
        PŘÍKAZ1;
        break;
    case 2:
        PŘÍKAZ2;
        break;
    case 3:
        PŘÍKAZ3;
        break;
    case 4:
        PŘÍKAZ4;
        break;
    default:
        PŘÍKAZ5;
        break;
}

```

Čím se to celé zjednodušuje.

### **Cyklus for (cyklus řízený proměnou )**

Tento cyklus použijeme, pokud předem známe kolikrát se má opakovat určitá operace  
 Syntaxe:

```
for(řidící_proměnná; podmínka; zvýšení_nebo_snížení_hodnoty_proměnné )
{
    PŘÍKAZ1;
    PŘÍKAZ2;
}
```

Příklad:

```
for(uint8_t i=0; i<4; i++)
{
    rozsvit_ledku();
    pockej_sekundu();
    zhasni_ledku();
}
```

Co tento kód udělá? Dejme tomu, že máme už předem vytvořené funkce `rozsvit_ledku()`, atd... Potom cyklus udělá následující: vytvoří proměnnou `i` datového typu `uint8_t` (celá čísla od 0 do 255), uloží do ní číslo 0. Potom zkontroluje podmínku, zda je proměnná `i` menší než 4, pak se provedou příkazy či funkce v těle cyklu, tj. rozsvítí se LED-dioda, bude svítit jednu sekundu, a pak zhasne. Pak se provede příkaz `i++`, ten zvýší hodnotu proměnné o jednu, tj. na číslo 1. Potom se kontroluje znovu podmínka, zdali je proměnná `i` (která má teď hodnotu 1) menší než 4, atd... Celkem se tělo cyklu vykoná čtyřikrát.

Cyklus `for` lze použít i bez proměnné, podmínky a zvýšení hodnoty, potom bude probíhat donekonečna. Příklad: `for(;;)`

```
{
    rozsvit_ledku();
    pockej_sekundu();
    zhasni_ledku();
}
```

Tento cyklus bude dělat to samé, co předchozí, s tím rozdílem, že to bude dělat donekonečna a nebude se ptát na podmínku.

Použití v programu pro robota: Nekonečný cyklus, ve kterém se kontrolují podmínky, zda je na čáře, zda je před ním soupeř, zda má udělat to či ono.

## Cyklus while (cyklus s podmínkou na začátku )

Použijeme, pokud nebudeme vědět kolikrát mají proběhnout příkazy a funkce v těle cyklu. Syntaxe:

```
while(podmínka)
{
    PŘÍKAZ1;
    ...
}
```

Příklad:

```
while(!je_tlacitko_stiskle())
{
    ujed_1_cm();
}
```

Tento cyklus bude stát, dokud nezmačkneme tlačítko. Vyžaduje mít předem naprogramovanou funkci `ujed_1_cm()` a `je_tlacitko_stiskle()`.

## 2.1.4 Funkce a procedury

Pokud se nám v zdrojovém kódu opakují dokola stejné příkazy, můžeme vytvořit funkce nebo procedury, které potom voláme a které nám usnadní kód. syntaxe:

```
datový_typ_který_funkce_vrací název_funkce(datový_typ_parametru1
parametr1, datový_typ_parametru2 parametr2)
// Parametry jsou nepovinné
{
    PŘÍKAZ1; // Nepovinné příkazy
    PŘÍKAZ2;
    return HODNOTA_KTEROU_FUNKCE_VRACÍ;
    // Za return se napíše výraz, který funkce vrací.
}
```

Příklad:

```
// Funkce na sčítání dvou čísel, každé od 0 do 255.
uint16_t umocni(uint8_t zakl, uint8_t exp)
{
    uint16_t vys = 1;
    if(exp > 0)
    {
        for(uint8_t i = 0; i < exp; i++)
        {
            vys = vys * zakl;
        }
    }
    return vys;
}
```

pokud někde v programu použiji funkci `secti(10, 2)`, tak výsledkem bude, jako bych napsal `10*10`, tj. 100. Příklad:

```
c = umocni(a, b); // Do proměnné c se uloží b-tá mocnina čísla a.
```

procedura – je vlastně funkce, jen s tím rozdílem, že procedura má datový typ `void` a nemá `return` (tj. nevrací žádnou hodnotu).

Příklad:

```
void rozsvitLedku()
{
    DDRB |= (1<<PB0); /* Co dělá tento příkaz nemusíte zatím řešit,
    pak se k tomu dostaneme.
    */
}
```

Pokud někde v programu napíšu `rozsvitLedku()`, tak se rozsvítí LEDka na robotovi.

tělo funkce `main` – všechny příkazy, které jsou mezi složenými závorkami, se po zahájení programu postupně vykonají

## 2.1.5 Knihovny

### Vlastní knihovny

Z funkcí a procedur se mohou vytvářet soubory pro usnadnění práce. těmto souborům se říká **knihovny** . Soubory mají koncovku *.h* Příklad: `avr/io.h`  
`moje_knihovna.h`

Vlastní knihovnu poté mohu vložit direktivou `#include` Příklad:

```
#include "moje_knihovna.h"
```

### Použité knihovny

Pro práci s mikrokontroléry budeme potřebovat některé už vytvořené knihovny  
Příklad: `Arduino.h` – zavádí příkazy pro práci s čipy ATmega a ESP32  
`LearningKit.h` – pro pohodlnou práci s deskou ALKS

## 2.2 Příkazy C++ pro čipy

## 2.3 Příklady programů v C++

Ve všech příkladech níže je uveden vždy obsah souboru *main.cpp*. Text předpokládá, že nad příklady budete samostatně přemýšlet a učit se z nich, proto se to, co bylo řečeno u prvního příkladu, už neopakuje u druhého. Doporučuji projít soubory *LearningKit.h* a *LearningKit.cpp* (viz v [Exploratu](#) adresa `.pio/libdeps/ArduinoLearningKitStarter_ID1745/src`), protože jsou v nich zkratky typu `L_R` a jejich přiřazení pinů čipu. Další příklady jsou na <http://wall.robotikabrno.cz> a <https://www.arduino.cc/reference/en/>.

### 2.3.1 Blikání LED

Program bliká červenou LED.

```
#include "LearningKit.h"

void setup() // this part run once
{
```

```

    pinMode(L_R, OUTPUT); // initialize LED digital pin as an output.
}

void loop() // this part works in cycle
{
    digitalWrite(L_R, HIGH); // switch on red LED
    delay(500); // pause 500 milliseconds
    digitalWrite(L_R, LOW); // switch off red LED
    delay(500);
}

```

### 2.3.2 LED zapínaná tlačítkem

Žlutá LED zapínaná tlačítkem.

```

#include "LearningKit.h"

void setup()
{
    pinMode(L_Y, OUTPUT); // initialize LED digital pin as an output

    digitalWrite(L_Y, HIGH);
}

void loop()
{
    if ((digitalRead(SW1)) == LOW)
        {digitalWrite(L_Y, LOW);}
    else {digitalWrite(L_Y, HIGH);}
}

```

### 2.3.3 Nejjednodušší PWM

PWM umožňuje (ve spolupráci s drivery ) řídit motory, serva a podobně. Zde je použito na stmívání LED pomocí potenciometru.



```
#include "LearningKit.h"

void setup()
{
    ledcSetup(0, 1000, 10);
    ledcAttachPin(L_Y, 0);
}

void loop()
{
    ledcWrite(0, analogRead(POT1)/4);
}
```

tento kód funguje pro čip ESP32. Pro čipy řady ATmega, které jsou na deskách Arduino uno a Arduino nano, je potřeba použít tento kód:

Funguje stejně, ale místo příkazu ledcWrite je použít:

# Kapitola 3

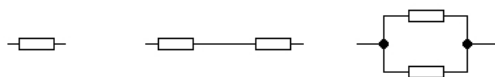
## Elektronika

### 3.1 Základní součástky

#### 3.1.1 Rezistor

**Rezistor** nebo také odpor<sup>1</sup> je součástka, která klade elektrickému proudu určitý odpor neboli ho omezuje. Toho se používá jako ochrana před zničením čipu nebo jeho části. Odpor se značí  $R$ . Jednotkou odporu je 1 Ohm, značka  $\Omega$ .

Dva rezistory (nebo jiné součástky) mohou být zapojeny buď **sériově** (tj. za sebou) nebo **paralelně** (tj. vedle sebe), viz obrázek 3.1.



Obrázek 3.1: vlevo: značka rezistoru, uprostřed: rezistory zapojené sériově, vpravo: rezistory zapojené paralelně

---

<sup>1</sup>Správně se součástce říká rezistor, odpor je její vlastnost. Ale mnoho lidí používá pro označení součástky slovo odpor.

### 3.1.2 Kondenzátor

**Kondenzátor** je součástka, která uchovává elektrický náboj. Jeho hlavní vlastností je kapacita. Jednotkou kapacity je Farad, značka F. V praxi se používají násobky jako mikrofaraď ( $\mu$  F), nanofaraď (nF) a pikofaraď<sup>2</sup> (pF). Kondenzátory se nabíjí a vybíjí různě rychle a mají různou kapacitu. Keramické kondenzátory mají nejmenší kapacitu (pF, nF) a jsou nejrychlejší, tantalové mívají kapacitu okolo pár  $\mu$ F a jsou pomalejší a nejpomalejší jsou elektrolytické s kapacitou stovek až tisíců  $\mu$ F. U tantalových a elektrolytických kondenzátorů musíme dát pozor na polaritu, tj. kam připojujeme + a kam -. Další důležitý údaj je maximální hodnota napětí, kterou kondenzátor snese.

### 3.1.3 Dioda

**Dioda** je součástka, která usměrňuje elektrický proud. To znamená, že pokud ji zapojíme do elektrického obvodu, tak zajistí, že proud bude téct pouze jedním směrem. Proto budeme diodu používat jako ochranu proti tzv. **přepólování** – chybnému zapojení baterie nebo součástky do obvodu, které obvykle vede ke zničení součástky. U samotné diody také záleží na polaritě, tj. při jejím zapojení musíme dávat pozor, kde má kladný pól a kde záporný. Na diodě vzniká úbytek napětí, se kterým musíme počítat při návrhu obvodu. Tak například pokud připojím na diodu s úbytkem napětí 0,6 V připojím 12 V, tak za diodou bude napětí 11,4 V.

Ze začátku nám bude stačit, pokud budeme používat diody [1N4148](#) a [1N4007](#).

### 3.1.4 LED

**LED**<sup>3</sup> je součástka, která není primárně určená k usměrnění proudu, ale k signalizaci, zda obvodem protéká proud. K LED se vždy musí připojit rezistor.

---

<sup>2</sup>Mikrofaraď je miliontina faradu, nanofaraď je tisíckrát menší a pikofaraď milionkrát menší než mikrofaraď.

<sup>3</sup>Light emitting diode – světlo vysílající dioda

### 3.1.5 Tranzistor

**Tranzistor** je součástka, která umožňuje pomocí malých proudů z čipu řídit větší proudy, například do reproduktoru nebo motorku.

Tranzistor má tři nožičky: **báze**, **kolektor** a **emitor**. Tranzistorů existuje obrovské množství, pro jednoduchost se budeme zabývat pouze tzv. bipolárními tranzistory. Ty existují ve dvou provedeních PNP a NPN<sup>4</sup>. Tranzistory mají prakticky dvě použití: mohou pracovat jako spínač (vypínač) nebo jako zesilovač. Budeme se zabývat jednodušším použitím, tj. jako spínače. Budeme používat tranzistory NPN. Pokud bude přes bázi do emitoru téct omezený (malý) proud, tranzistor se otevře a přes kolektor do emitoru bude téct velký proud. Tak nám stačil malý proud k řízení velkého. A toho budeme využívat. Ze začátku nám bude stačit používat tranzistory [BCC337](#), [BCC547](#) a [BD911](#).

### 3.1.6 Cívka

**Cívka** neboli **tlumivka** je součástka, jejíž hlavní vlastností je indukčnost, jednotka henry, značka H. V praxi se používají milihenry (mH) a mikrohenry ( $\mu\text{H}$ ).

### 3.1.7 Driver

Čip nemůže řídit například motor přímo, protože jedním pinem může protékat obvykle maximálně 40 mA. Většina motorů potřebuje mnohem větší proud. Proto se používají součástky zvané **drivery**, které podle pokynů z čipu řídí proud z baterií do motorů a servomotorů.

### 3.1.8 Stabilizátor

Většinu čipů je potřeba napájet přesně 5 V nebo 3,3 V. Jak toho docílit z baterií, na kterých je například 9 V nebo 12 V, zkrátka více než 5 V? Navíc napětí na bateriích kolísá podle toho, jak moc proudu zrovna odebírají motory. Pro napájení čipů je proto nutné použít stabilizátor.

**Stabilizátor** je součástka, která z kolísavého vyššího napětí vyrobí přesné napětí nižší. Nejčastěji se používají stabilizátory řady 78XX, kde XX značí, na kolik voltů součástka stabilizuje, například 7805 stabilizuje na 5 V.

---

<sup>4</sup>Pro lepší zapamatování značky: eN-Pé-eN - šipka VEN

Aby stabilizátor mohl pracovat správně, je potřeba, aby napětí, které přivedeme na jeho vstup, bylo aspoň o 2 V vyšší než které potřebujeme, tj. pokud budu chtít stabilizovat napětí na 5 V, musím stabilizátor napájet aspoň 7 V. Přesné hodnoty pro každý stabilizátor jsou v jeho datasheetu.

Stabilizátor má tři piny (vstup, zem, výstup). Zapojí se takto: kladný pól baterie (+) se napojí na vstup, záporný pól na zem (−). Mikrokontrolér se zapojí pinem VCC na výstup stabilizátoru a GND se zapojí na zem stabilizátoru. Tímto máme připojený mikrokontrolér na napájení.

Pokud máme stabilizátor před sebou tak, abychom přečetli jeho označení, např. L7805, potom první pin zleva je vstup, druhý je zem a třetí, tj. úplně vpravo je výstup. Na vstup připojíme 7 V až 12 V, prostřední pin uzemníme, a poslední pin vyvedeme na VCC mikrokontroléru. Dále je potřeba věnovat pozornost zapojení kondenzátorů. Mezi vstup a zem připojím podle datasheetu kondenzátor s kapacitou 330 nF. Mezi výstup a zem kondenzátor 100 nF.

### 3.1.9 Krystal

Pokud budeme potřebovat provozovat některé procesory na vyšší frekvenci, použijeme **krystal**. Například původní frekvence mikrokontroléru ATmega16 je nastavena na 1 MHz. S pomocí krystalu ji můžeme zvýšit až na 16 MHz. Krystal zapojíme takto: Jeden pin krystalu (je jedno, který) připojíme na pin XTAL1 a druhý na XTAL2. Dále na pin XTAL1 připojíme jednu nožičku kondenzátoru a druhou na digitální zem (11-GND). To samé u pinu XTAL2. Hodnotu kondenzátoru můžeme volit od 12 pF do 22 pF.

## 3.2 Základní veličiny v elektronice

### 3.2.1 Proud

Pokud použijeme vodní model, tak (elektrický) proud je množství vody, které proteče vodičem za jednu sekundu.<sup>5</sup> Značka:  $I$ , jednotka: 1 A = 1 ampér.

---

<sup>5</sup>Ve skutečnosti je to protečení množství elektronů nebo jiných nosičů el. nábojů s celkovým nábojem 1 Coulomb za sekundu.

### 3.2.2 Napětí

Elektrické napětí měříme vždy mezi dvěma body. Můžeme si ho představit jako rozdíl výšek dvou vodních hladin. Z výše položeného jezera (kladné napětí, +) teče voda (el. proud) do níže položeného (zem, nulový potenciál, -). Značka:  $U$ , jednotka:  $1 \text{ V} = 1 \text{ volt}$ .

### 3.2.3 Odpor

Mezi napětím  $U$  a elektrickým proudem  $I$  platí vztah:

$$U = R \cdot I,$$

Tento vztah je znám pod názvem **Ohmův zákon**.

Konstanta  $R$  se nazývá elektrický odpor, měříme ho ohmech, značka  $\Omega$ .

Z Ohmova zákona můžeme vyjádřit proud  $I$ :

$$I = U/R.$$

Ze vzorce je vidět, že pokud použijeme rezistor s větším odporem při stejném napětí, tak se protékající proud zmenší.

### 3.2.4 Výkon

Výkon je definován jako součin napětí a proudu:

$$P = U \cdot I.$$

Značka:  $P$ , jednotka:  $1 \text{ W} = 1 \text{ watt}$ .

### 3.2.5 Výpočet tepelného výkonu (ztrátového)

Na součástkách, na kterých je úbytek napětí a kterými protéká proud, vznikají tepelné ztráty.

**Příklad 1:** Na rezistoru je úbytek napětí 3,6 V a protéká jím 240 mA. Jaký je tepelný ztrátový výkon?

$$P = U \cdot I = 3,6 \text{ V} \cdot 0,24 \text{ A} = 0,864 \text{ W} = 864 \text{ mW}$$

Ztráty na rezistoru činí 864 mW, a proto musíme volit rezistor, který zvládne větší výkon, tj. minimálně 1 W.

**Příklad 2:** Diodou 1N4148 bude procházet 100 mA, při tomto proudu bude úbytek napětí na diodě 1 V. Nezničíme diodu 1N4148?

$$P = U \cdot I = 1 \text{ V} \cdot 0,1 \text{ A} = 0,1 \text{ W} = 100 \text{ mW}$$

Diodu nezničíme, protože je vyráběná na celkový výkon 500 mW.

**Příklad 3:** Na stabilizátor L7805 přivádím 12 V, stabilizátor mi vytváří 5 V stabilizovaného napětí a odebírám z něho 250 mA. Jaký tepelný výkon bude potřeba uchlazit?

$$P = U \cdot I = (12 \text{ V} - 5 \text{ V}) \cdot 0,25 \text{ A} = 1,75 \text{ W}$$

Tepelné ztráty na stabilizátoru budou 1,75 W<sup>6</sup>.

### 3.2.6 Výpočet rezistoru pro LED

LED je součástka, která není primárně určená k usměrnění proudu, ale k signalizaci. Může svítit světlem bílým, modrým, zeleným, červeným, ultra-fialovým či infračerveným<sup>7</sup>.

Pokud připojím diodu správně na napětí, tj. tak aby mohl procházet proud a ono přesto nic, tak jsem diodu spálil, protože jí tekla moc velký proud. A proto musíme vždy k diodě připojit do **série** rezistor, který omezí proud protékající přes LED. A to podle vzorce:

$$R = \frac{U}{I} = \frac{\text{Napeti\_zdroje} - \text{Ubytek\_napeti}}{\text{Pozadovany\_proud}}$$

**Příklad 1:** Vypočítejte odpor rezistoru, který zapojíme do série k LED. Připojujeme k ní napětí 5 V a provozní proud je 20 mA a maximální proud, při kterém dojde ke zničení diody je 40 mA. Úbytek na diodě je 1,2 V.

$$R = \frac{U}{I} = \frac{5 \text{ V} - 1,2 \text{ V}}{0,02 \text{ A}} = 190 \Omega$$

---

<sup>6</sup>Ve vzorci bylo použito 12 V - 5 V, je to protože na stabilizátoru ubude 7 V, abychom se dostali na požadovaných 5 V.

<sup>7</sup>Toho využijeme jako senzoru pro robota.

Použijeme rezistor s odporem  $190\ \Omega$  nebo nejbližší vyšší odpor.

**Příklad 2:** Máme sériově spojeny tři LED, s úbytky napětí  $0,6\text{ V}$ ,  $0,8\text{ V}$  a  $1,2\text{ V}$ . Připojíme je k zdroji o napětí  $12\text{ V}$ . Rezistor o jak velkém odporu musíme použít, jestliže má diodami protékat  $20\text{ mA}$ ?

$$R = \frac{U}{I} = \frac{12\text{ V} - 0,6\text{ V} - 0,8\text{ V} - 1,2\text{ V}}{0,02\text{ A}} = 470\ \Omega$$

Musíme použít rezistor o odporu  $470\ \Omega$ .

Měli bychom ještě spočítat tepelný výkon rezistoru. Můžeme použít úbytek napětí na rezistoru vynásobený procházejícím proudem:

$$P = U \cdot I = (12\text{ V} - 0,6\text{ V} - 0,8\text{ V} - 1,2\text{ V}) \cdot 0,02\text{ A} = 9,4\text{ V} \cdot 0,02\text{ A} = 0,188\text{ W} = 188\text{ mW}$$

Stačí  $250\text{ mW}$  rezistor. Nebo si můžu vypočítat úbytek napětí z  $U = R \cdot I$ , to protože znám odpor a protékající proud.

$$P = U \cdot I = R \cdot I \cdot I = R \cdot I^2 = 470\ \Omega \cdot (0,02\text{ A})^2 = 0,188\text{ W} = 188\text{ mW}$$

Vyšel nám stejný výsledek, použijeme tedy  $250\text{ mW}$  rezistor o odporu  $470\ \Omega$ .

## 3.3 Motorky, serva a PWM

### 3.3.1 Motorky

Motorek připojujeme vždy přes [tranzistor](#). Pokud chceme řídit motorek z čipu stylem start – stop, postačí přes odpor např.  $1\text{ k}\Omega$  spojit výstupní pin čipu s bází tranzistoru a na emitor a kolektor připojit baterii a motorek zapojený do série.

Dále je potřeba bází tranzistoru propojit pomocí např.  $10\text{ k}\Omega$  se zemí. Jinak totiž při vypnutém signálu z čipu báze „visí v luftě“ a chová se jako anténa – indukují se na ní různé signály a většinou se díky tomu motorek samovolně slabě otáčí.

Nakonec je potřeba mít společnou zem pro čip i pro motorek – pokud to není splněno, obvykle motorek nejede.



Pro větší motorky potřebujeme lepší tranzistor nebo dva tranzistory v tzv. Darlingtonově zapojení (viz web). Další možností jsou speciální integrované obvody pro řízení motorů, tzv. [drivery](#).

Pokud chceme řídit motory programově pomocí PWM (viz další kapitola), musíme navíc mezi emitor a kolektor tranzistoru vložit (obyčejnou) diodu polovanou závěrně vůči baterii. Při vypnutí tranzistoru vznikají totiž na cívkách motorku napěťové špičky, které deformují tvar PWM signálu. Další proudy se indukují, když se motorek po vypnutí proudu setrvačností otáčí dál.

### 3.3.2 PWM

#### 3.3.3 Řízení serva

Jak se řídí pohyb serva? Pro tento účel je ideální právě generování PWM signálu.

Servo se řídí logickým signálem (jedničkou) po dobu od 1 ms do 2 ms (často i od 0,5 ms do 2,5 ms), a celková perioda je 20 ms. Podle toho, jak dlouho signál trvá, tak se servo natočí. Tj. pokud budeme chtít servo maximálně natočit na jednu stranu, nastavíme pin, který slouží jako řídicí signál pro servo, na logickou jedničku po dobu 1 ms a pak 19 ms logickou nulu a pak zase logickou jedničku, logickou nulu, atd...

Pokud budeme chtít servo posunout do druhé krajní polohy, necháme logickou jedničku po dobu 2 ms a logickou nulu po dobu 18 ms. Pokud budeme chtít střední polohu, tak jedničku nastavíme na 1,5 ms a nulu na 18,5 ms. Jestliže budeme potřebovat jiný úhel natočení, nastavíme logickou jedničku na odpovídající dobu.

**Pozor!** Servo nesnáší přepólování napětí, když se přepóluje, tak shoří (když se přepóluje signál, tak to tolik nevádí).

## 3.4 ALKS, další desky a další zařízení

### 3.4.1 ESP 32

Deska ESP 32 je vývojová deska osazená čipem ESP-WROOM-32, který má řadu výborných vlastností.<sup>8</sup>

---

<sup>8</sup>[urlhttp://navody.arduino-shop.cz/navody-k-produktum/vyvojova-deska-esp32.html](http://navody.arduino-shop.cz/navody-k-produktum/vyvojova-deska-esp32.html)

Deska se napájí z USB (5 V) a je na ní napěťový převodník na 3,3 V. Přitom USB může dodávat oficiálně 100 mA, v reálu ale běžně dodává 500 mA až 1 A. USB porty jsou také vcelku odolné proti zkratu.

### 3.4.2 ALKS

Deska ALKS<sup>9</sup> byla navržena přímo na Robotárně Brno právě proto, že hotová deska vás hlavně ze začátku zbavuje nutnosti vědět, co si můžete dovolit kam připojit a jestli to bude fungovat.

Na ALKS se dají nasadit desky ESP 32, Arduino uno a Arduino nano, které jí také poskytují napájení.

Stránky věnované desce ALKS jsou zde:

<https://github.com/RoboticsBrno/ArduinoLearningKitStarter>

Zapojení desky ALKS je zde: <https://github.com/RoboticsBrno/ArduinoLearningKitStarter/blob/master/docs/ArduinoLearningKitStarter.pdf>

Zapojení pinů desky ALKS zde: <https://github.com/RoboticsBrno/ArduinoLearningKitStarter/blob/master/docs/pinout.pdf>

Při připojování čehokoliv dalšího k této nebo jiné desce si nechte před zapojením napájení všechno zkontrolovat. Hlavně, pokud to žere víc proudu (serva a motory) nebo pokud to vyžaduje vyšší napětí.

### 3.4.3 Bluetooth

Moduly **bluetooth** slouží ke komunikaci mezi dvěma čipy nebo počítačem a čipem (nebo jiným zařízením). Mohou být napájeny 5 V nebo 3,3 V.

Propojujeme vždy pin Rx na jednom čipu s pinem Tx na druhém čipu.

### 3.4.4 Připojení k počítači pomocí bluetooth

Nový bluetooth (zub) se musí napoprvé vyhledat a aktivovat v počítači. Pokaždé se musí připojit a zkontrolovat – když je komunikace v pořádku (aktivována, ale nemusí se přenášet data), svítí LED na zubu. Když je v pořádku modul v počítači, tak bliká.

Dále musí být spojena země zubu a čipu.

---

<sup>9</sup>Arduion Learning Kit Starter

## 3.5 Desítková soustava a dvojková soustava

V počítači jsou všechna data uložena pouze v podobě jedniček a nul. A proto se nám bude hodit, když se budeme aspoň trochu orientovat v převodech mezi dvojkovou a desítkovou soustavou.

### 3.5.1 Desítková soustava

V **desítkové soustavě** je základem číslo deset.

Příklad: číslo 156 si můžeme rozložit jako  $1 \cdot 10^2 + 5 \cdot 10^1 + 6 \cdot 10^0$ , přitom  $10^0=1$ ,  $10^1=10$ ,  $10^2=10 \cdot 10=100$ ,  $10^3=10 \cdot 10 \cdot 10=1000$  atd ..., takže naše číslo potom vypadá takto:  $1 \cdot 100 + 5 \cdot 10 + 6 \cdot 1 = 156$ .

### 3.5.2 Dvojková soustava

Ve **dvojkové soustavě** je základem číslo dva.

Příklad: číslo 11001010 rozložíme na

$1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$ ,  
přitom

$2^0=1$ ,  $2^1=2$ ,  $2^2=2 \cdot 2=4$ ,  $2^3=2 \cdot 2 \cdot 2=8$ ,  $2^4=2 \cdot 2 \cdot 2 \cdot 2=16$ ,  $2^5=32$ ,  
atd..., proto naše číslo má hodnotu:

$1 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 202$ .

### 3.5.3 Převod z desítkové soustavy do dvojkové

Jak na to, když chceme převádět číslo z desítkové soustavy do dvojkové?

Například převedeme číslo 97.

Nejprve je potřeba, abychom znali násobky čísla dva:

2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, atd...

Napišeme jedničku.

1

Najdeme nejbližší nižší násobek dvou k našemu číslu, v našem případě je nejbližší 64. Odečteme si od 97 číslo 64,  $97-64=33$ . Odečítali jsme číslo 64, další v pořadí je číslo 32, je 32 obsaženo v 33? Ano, proto napíšeme další jedničku.

11

Dále  $33-32=1$ , další číslo je 16, je 16 obsaženo v 1? Ne, proto napíšeme nulu.

110

Další číslo je 8, také není obsaženo v 1, takže napíšeme nulu.

1100

Čísla 4 a 2 také nejsou obsažena v čísle 1, proto napíšeme nuly.

110000

A nakonec, je číslo jedna obsaženo v čísle jedna? Ano, napíšeme jedničku. Náš binární zápis čísla 97 vypadá takto: 1100001. Čísla v binární soustavě by měla mít počet cifer rovno 8, 16, 32, podle toho s kolika bitovými čísly počítáme. V našem případě musíme doplnit jednu nulu. Pokud bychom počítali s 16 bitovými čísly doplníme o 8 nul více. Nuly doplňujeme zleva. Zapamatujeme si to snadno v desítkové soustavě se číslo 128 také nezmění pokud před něj dáme nuly, 000128. Nuly navíc nebudeme psát v desítkové soustavě, ale v dvojkové nám to pomůže se lépe orientovat. Takže to bude vypadat: 01100001 No a to ještě není všechno. Aby mělo číslo správný tvar, musí mít před číslem uvedeno 0b(nula a malé B), tím říkáme v jazyce C, že číslo je v (binární) dvojkové soustavě.

Pozor: Čísla 101 a 0b101 nejsou to samé! Je to jednoduché, pokud 0b neuvedeme bude překladač číslo brát jakože je v desítkové soustavě. Tj. 101 => číslo sto jedna, 0b101 => číslo pět.

### **Další příklad:**

Převédeme číslo 237.

Nejbližší nižší číslo je 128.

1

$237-128=109$  je číslo 64 obsaženo v čísle 109? ANO, další jednička.

11

$109-64=45$  je číslo 32 obsaženo v 45? ANO, jednička.

111

$45-32=13$  je 16 obsaženo v čísle 13? NE, nula.

1110

je číslo 8 obsaženo v 13? ANO, jednička.

11101

$13-8=5$  je 4 obsaženo v 5? ANO, jednička.

111011

$5-4=1$  je obsažena v 1? NE, nula.

1110110

je 1 obsažena v 1? ANO, jednička.

11101101 Binární zápis čísla 237 vypadá takto: 11101101.

### 3.5.4 Převod z dvojkové soustavy do desítkové

Vezmeme si například binární číslo 1011. Očíslujeme si pozice zprava doleva od nuly, tj.:

1 0 1 1  
3 2 1 0

Pohybujeme se v dvojkové soustavě, tj. základ bude 2. Vezmeme nultou číslici a vynásobíme s ní číslo  $2^0$ . Dostáváme  $1 \cdot 2^0 = 1$ . Dále první číslici vynásobíme číslem  $2^1$ , tj.  $1 \cdot 2^1 = 2$ . Dále druhou číslici vynásobíme číslem  $2^2$ , tj.  $0 \cdot 2^2 = 0$ . A to samé provedeme s číslicí na třetí pozici:  $1 \cdot 2^3 = 8$ . A nyní čísla sečteme  $8 + 0 + 2 + 1 = 11$ , takže binární číslo 1011 převedené do desítkové soustavy je 11.

Příklad 2: Převed'te binární číslo 11101001 do desítkové soustavy.

$$1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5 + 1 \cdot 2^6 + 1 \cdot 2^7$$

To je:

$$1 + 0 + 0 + 8 + 0 + 32 + 64 + 128 = 233 \quad \text{Binární číslo 11101001 převedeno do desítkové soustavy je 233.}$$

Příklad 3: Převed'te binární číslo 00010101.

$1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 + 0 \cdot 2^6 + 0 \cdot 2^7$ , samozřejmě členy  $2^x$ , které násobíme nulou budou nulové a tudíž je můžeme beztréstně vynechat, tj.

$$1 \cdot 2^0 + 1 \cdot 2^2 + 1 \cdot 2^4 = 1 + 4 + 16 = 21 \quad \text{Binární číslo 00010101 má v desítkové soustavě hodnotu 21.}$$

### 3.5.5 Bitové výrazy a práce s nimi

Bitové operace se od logických operací (viz [2.1.2](#)) liší tím, že se neaplikují na číslo jako celek, ale bit po bitu, viz níže.

My budeme potřebovat následující bitové operace: bitový součet, bitový součin, bitovou negaci, bitový posun doleva.

## Bitové operátory

| bitový součet  
& bitový součin  
~ bitová negace  
^ bitový exkluzivní součet  
<< bitový posuv doleva  
>> bitový posuv doprava

## Bitový součet

Platí, že  $0|0=0$ ,  $0|1=1$ ,  $1|0=1$ ,  $1|1=1$ , sčítají se vždy pouze bity, které leží pod sebou. Zkrátka pokud je aspoň jeden bit jedna, výsledek je také jedna. Příklad: máme dvě binární čísla 00000100 a 01100001 a chceme je bitově sečíst.

Když sečteme nulté bity zadaných čísel, dostaneme  $0|1=1$ . Pak provedeme bitový součet i pro první, druhé a další bity. Zapišeme:  $(0b00000100 | 0b01100001)$ ; Nebo pokud budeme mít čísla v proměnných:

```
uint8_t a~ = 0b00000100;
uint8_t b = 0b01100001;
uint8_t c;
c = (a|b);
```

Do proměnné c se zapíše výsledná hodnota.

```
00000100 - první číslo
01100001 - druhé číslo
-----
01100101 - výsledek po bitovém logickém součtu
```

Výsledek: 01100101.

## Bitový součin

Platí, že  $0 \& 0 = 0$ ,  $0 \& 1 = 0$ ,  $1 \& 0 = 0$ ,  $1 \& 1 = 1$ , tedy platí, že výsledek je jedna, pokud oba bity jsou jedna. Příklad: bitový součin čísel 00110011 a 01000001 Zapišeme:  $(0b00110011 \& 0b01000001)$ ; Nebo pokud budeme mít čísla v proměnných:

```
uint8_t c = 0b00110011;
uint8_t d = 0b01000001;
uint8_t e;
e = (c&d);
```

Výsledná hodnota se zapíše do proměnné e.

```
00110011 - první číslo
01000001 - druhé číslo
-----
00000001 - výsledek bitového logického součinu
```

## Bitová negace

Tam, kde byla jednička, bude nula, a kde byla nula, bude jednička. Příklad: znegujte číslo 00110101 Zapišeme:  $\sim(0b00110101)$ ; Nebo pokud budeme mít číslo v proměnné:

```
uint8_t a = 0b00110101;
uint8_t b;
b = ~a;
```

Do proměnné b se zapíše výsledná hodnota.

```
00110101 - číslo, které bude logicky znegováno
-----
11001010 - výsledek
```

## Bitový exklusivní součet

Výsledkem exklusivního součinu je pravda, pokud se hodnoty liší, tj. nejsou stejné. Takže platí  $0 \wedge 0 = 0$ ,  $0 \wedge 1 = 1$ ,  $1 \wedge 0 = 1$ ,  $1 \wedge 1 = 0$

Zapišeme:  $(0b00110011 \wedge 0b01000001)$ ; Nebo pokud budeme mít čísla v proměnných:

```
uint8_t c = 0b00110011;
uint8_t d = 0b01000001;
uint8_t e;
e = (c^d);
```

Výsledná hodnota se zapíše do proměnné e.

```
00110011 - první číslo
01000001 - druhé číslo
-----
01110010 - výsledek bitového exklusivního součtu
```

## Bitový posun doleva

Vezmeme celé osmibitové číslo a posuneme ho o určitý počet míst doleva. Přitom čísla vlevo zmizí a na na uprázdněné místo vpravo přijdou vždy nuly. Zapišeme:  $(0b01000101 \ll 3)$ ; Nebo pokud budeme mít číslo v proměnné:

```
uint8_t c = 0b01000101;
uint8_t d;
d = (c << 3);
```

Do proměnné d se přiřadí výsledek operace. Pokud například už nebudeme potřebovat původní hodnotu proměnné c, lze to napsat i takto:

```
uint8_t c = 0b01000101;
c = (c << 3);
```

Hodnota v proměnné c se nejprve posune o tři pozice a pak se výsledná pozice zapíše na stejnou pozici v paměti mikrokontroléru.

Příklad: proveďte bitový posun doleva o tři s číslem 00000101



01000101

-----

10001010 - posun o jedno doleva

00010100 - znovu o jedno doleva (celkem o dvě místa)

00101000 - celkový posun o tři bity doleva

### Bitový posun doprava

Bitový posun doprava je podobný jako operace bitový posun doleva, liší se jen ve směru posunu, tj. posouvá se na opačnou stranu. Příklad 1: Proveďte bitový posun doprava o tři pozice s binárním číslem 00000101. Zapišeme: (0b00000101 >> 3); Výsledkem bude toto:

00000101

-----

00000010 - posun o jednu pozici doprava

00000001 - posun o druhou pozici

00000000 - posun o třetí pozici

Příklad 2: Proveďte bitový posun o dvě pozice doprava a potom o tři doleva s binárním číslem 11110111. Zapišeme: ((0b11110111 >> 2) << 3); Výsledkem bude toto:

11110111

-----

00111101 - posun o dvě pozice doprava

11101000 - posun o tři pozice doleva, toto je výsledek.

Příklad 3: Ale co když zadám číslo do proměnné v desítkové soustavě, například 189 a provedu bitový posun o dvě pozice doprava? Co dostanu? Dostanu snad 1? Ne, i když jsme číslo zadali v desítkové soustavě, v mikrokontroléru je stejně uloženo jako jedničky a nuly, takže číslo musíme nejprve převést do dvojkové soustavy.  $189 = 0b10111101$  A teď posun o dvě pozice doprava: 00101111 a to je číslo 47. Takže 47 je výsledek bitového posunu čísla 189 o dvě pozice doprava.

Zapišeme takto:

```
uint8_t a = 189;
```

```
uint8_t b;
```

```
b = (a >> 2);
```

No a výsledek (tj. číslo 47 ) se uloží do proměnné **b**.

# Kapitola 4

## Software

### 4.1 L<sup>A</sup>T<sub>E</sub>X

#### 4.1.1 Proč používat L<sup>A</sup>T<sub>E</sub>X

Tento text je psán v sázecím systému L<sup>A</sup>T<sub>E</sub>X. Jeho silná stránka je především matematická sazba (bohužel nevyužijeme) a snadné zpracování obsahu, rejstříků, seznamů obrázků a tabulek a podobně, což dramaticky urychluje přípravu dokumentu. Čas, který vložíte do nastavení a učení se systému, se vrátí v rychlosti práce → jedná se o řešení vhodné pro delší dokumenty, např. pro soutěž SOČ nebo dlouhodobou maturitní práci.

Návody pro L<sup>A</sup>T<sub>E</sub>X lze najít v [http://mirrors.nic.cz/tex-archive/info/czech/\delimiter"026B30Dlatex-pro-pragmatiky/latex-pro-pragmatiky.pdf](http://mirrors.nic.cz/tex-archive/info/czech/\delimiter) a na internetu. Hodně vám také může pomoci zdrojový text této dokumentace.

Příkazy L<sup>A</sup>T<sub>E</sub>Xu podobně jako C++ a systémy typu Linux **rozlišují velká a malá písmena**.

#### 4.1.2 Editory

Ve WinXP používám PSpad, v linuxu TeXstudio. Oba editory umí zavolat překlad do pdf pomocí klávesové zkratky, zobrazit výsledný pdf a barevné zvýraznění syntaxe. PSpad se dá rozjet i pod linuxem pomocí prostředí Wine, ale nepodařilo se mi rozchodit volání překladu.

## 4.2 Github

### 4.2.1 Proč a jak se používá

Pokud více lidí pracuje průběžně na stejném projektu, musejí nějak sdílet výsledky svojí práce. K tomu se používají tzv. repozitáře a Github<sup>1</sup> je v současnosti jeden z nejpoužívanějších webů pro tvorbu a správu repozitářů.

Postup práce je následující: na webu github.com si vytvořím repozitář (viz 4.2.2) . Stáhnu k sobě na počítač aktuální verzi repozitáře, upravím, co potřebuji a upravené soubory nahraju zpět na server.

Repozitář, ve kterém je i tato dokumentace, je na adrese <https://github.com/RoboticsBrno/RobotikaBrno-guides/tree/RoboticsManual>.

### 4.2.2 Instalace githubu a stažení repozitáře

1. Na github.com vytvořím účet a přihlásím se. Vpravo můžu založit nový repozitář (new repository) nebo se můžu přepnout do už existujících repozitářů.
2. Na svém počítači si nainstaluji github. V linuxu napíšu do terminálu: `sudo apt-get install git`.
3. Vytvořím si lokální adresář a stáhnu do něj repozitář. V linuxu napíšu do terminálu: `git clone <cestu k repozitáři>` . Cestu zjistím tak, že se na webu přepnu do repozitáře a kliknu na tlačítko „Clone or download“. Ose bjeví

## 4.3 Další software

### 4.3.1 Proficad

Proficad<sup>2</sup> je software určený původně pro snadné a rychlé kreslení elektronických schémát a v této oblasti je vynikající. Lze jej použít i jako jednoduchý vektorový editor obrázků.

---

<sup>1</sup>[www.github.com](http://www.github.com)

<sup>2</sup>Instalační soubor seženete v kroužku nebo na webu.

SPŠ Sokolská zakoupila plnou multilicenci pro Proficad, takže studenti i učitelé jej mohou používat bez omezení.

Ovládání programu je velice intuitivní a nápovědu prakticky nepotřebujete – s jedinou výjimkou, a tou je nastavení rastru. Po instalaci je rastr zobrazení automaticky nastaven na 2 mm. To znamená, že součástky můžete umisťovat například 10 mm nebo 12 mm od kraje, ale nic mezi tím. Většinou se to hodí – součástky máte na schématu pěkně zarovnané – ale někdy je prostě potřeba rastr například vypnout neboli nastavit na nulu. Nastavení rastru je schované zde: *soubor/nastavení/dokument/obsah/rastr*.

### 4.3.2 Linux

Tato kapitolka není úvodem do Linuxu (materiálů na toto téma je na webu spousta). Jsou zde poznámky, které hodí, když uživatel přechází z Windows na Linux.

#### Některé rozdíly Linux – Windows

- Když si chce uživatel nainstalovat Linux, vybírá ne operační systém, ale tzv. distribuci = operační systém + spousta předinstalovaných programů. Kritérií pro výběr je spousta, pár tipů:
  - distribuce *Mint* má podobné rozložení grafických ovládacích prvků jak windows a je nenáročná na hardware
  - distribuce *Ubuntu* je standardní volba s velkou výbavou programů. Její odlehčená verze pro starší počítače je *Lubuntu*.
- V Linuxu se celkově mnohem víc (a účinněji) používá příkazová řádka neboli program **terminál**. Spouští se klávesou F4.
- Linux **rozlišuje velká a malá písmena**. V terminálu, v názvech souborů, všude.
- Adresářová struktura Linuxu je jednotná pro všechny disky v počítači. Akce typu: „*přepni se na disk e:*“ se v terminálu provede jako „*přepni se do adresáře /media/<uživatel>/ ...* “. Nebo se přepnete pomocí jiného programu, např. Správce souborů.
- Některé programy fungují pod Linux i Windows, u jiných se při přechodu z Windows na linux musí na webu najít odpovídající náhrada, opět pár tipů:

- Ve WinXP se pro prohlížení pdf souborů osvědčil Foxit Reader verze 2.3, v linuxu Evince.
- Irfan view jede i v Linuxu<sup>3</sup>.

### Co se hodí vědět

- Heslo v Linuxu se při zadávání v terminálu nevypisuje ani tečkami (ale počítač o něm ví!).
- Znak zavináč se napíše **AltGr+v**.
- Drivery do tiskáren mají příponu **ppd**. Před jejich použitím se musí nainstalovat program **cups**.
- Snímek obrazovky uložíte stiskem klávesy **PrintScreen** do adresáře **/home/<prihlaseny uzivatel>**.
- Znak **#** znamená řádek na terminálu (zastupuje vypsání aktuálního adresáře).

---

<sup>3</sup><http://www.boekhoff.info/install-irfan-view-on-linux/>

# Kapitola 5

## Ostatní

### 5.1 Mechanická konstrukce robota – doporučení

Roboty rozlišujeme podle způsobu ovládání a podle velikosti.

#### Ovládání robotů

Každý robot může být buď řízený nebo autonomní. **Autonomní** znamená, že je naprogramovaný a během soutěže nebo prezentace se pohybuje samostatně.

Roboty můžeme řídit po kabelu nebo bezdrátově, například přes [bluetooth](#).

#### Velikost robotů

Na škole a v DDM stavíme a programujeme hlavně roboty dvojího typu : „střední“ a „velké“.

Jako střední používáme roboty *yunimin* a *3pi* - jsou v zásadě hotové a „pouze“ se programují, jsou určeny pro soutěže typu sledování čáry (Robotický den v Praze, Istrobot), minisumo a další. Pro střední roboty neexistuje soutěž, kde by se ovládali po kabelu, musí proto být autonomní.

Velké roboty (typicky pro soutěž Eurobot a Robotický den) si stavíme sami. Mohou být dálkově řízené i autonomní. Začátečnickům se silně doporučuje začít s dálkově řízeným robotem a nebo se přidat k týmu zkušenějších konstruktérů.

### 5.1.1 Konstrukce velkých robotů

Celého robota nejprve navrhne ve vhodném CAD programu: Solidworks<sup>1</sup>, onshape<sup>2</sup>, . Pokud jako základní materiál zvolíme překližku nebo například plexisklo (kombinované s díly z lega), můžeme díly vymodelované v CADu nechat vyřezat na laseru na pracovišti Fablab.<sup>3</sup> To dramaticky urychluje práci. Pro konstrukci prototypů se také hodí měkčené PVC.

Při stavbě velkých robotů se držíme osvědčeného schématu:

Podvozek tvoří základní deska z překližky nebo rám z hliníkových profilů typu „L“ tvaru obdélníku, osmiúhelníku nebo kruhu. Na podvozku jsou připevněné dva motory z akušroubováků nebo z akuvrtaček včetně převodovek, které se koupí hotové. Jiné typy motorů jsou buď pomalé nebo slabé. V žádném případě se nepokoušejte koupit pouze motor (např. protože je levnější) a vyrábět si převody sami.

Dále jsou zde dvě kola, každé připojené ke svému motoru a jedna nebo více podpěr podvozku, obvykle z kartáčku na zuby. Motory s koly lze umístit doprostřed nebo dozadu, podle toho, co má robot na hřišti dělat.

Na podvozek se umísťuje konstrukce z hliníkových tyčí, profilů nebo z merkuru, s pomocí které robot plní svoje úkoly. Měně tuhé materiály se neosvědčily.

## 5.2 Řešení některých problémů

### 5.2.1 Chyba při uploadu programu do desky Arduino nano:

Chyba:

```
Please specify 'upload_port' environment or use global  
'-upload-port' option
```

Řešení: pomohlo spustit příkaz `sudo service udev restart` a znovu spustit VS code a odpojit a připojit desku.

---

<sup>1</sup><http://www.solidworks.com>

<sup>2</sup>[doplňit](https://onshape.com)

<sup>3</sup><https://www.fablabbrno.cz>



## 5.3 Příloha A – Hodnoty vybraných součástek

### **Dioda 1N4148**

Maximální napětí: 100 V

Maximální proud: 200 mA

Maximální výkon: 500 mW

Úbytek napětí: 1,8 V

### **Dioda 1N4007**

Maximální napětí: 1000 V

Maximální proud: 1 A

Maximální výkon: 3 W

Úbytek napětí: 1,1 V

### **tranzistor BC337**

Max. napětí mezi kol. a emit.  $V_{CEO}$ : 50 V

Max. napětí mezi bází a emit.  $V_{CBO}$ : 5 V

Max. proud tekoucí kolektorem  $I_C$ : 800 mA

Maximální výkon  $P_C$ : 625 mW

Zesílení  $h_{fe}$ : 100 až 600

### **tranzistor BC547**

Max. napětí mezi kol. a emit.  $V_{CEO}$ : 45 V

Max. napětí mezi bází a emit.  $V_{CBO}$ : 6 V

Max. proud tekoucí kolektorem  $I_C$ : 100 mA

Maximální výkon  $P_C$ : 500 mW

Zesílení  $h_{fe}$ : 110 až 800

### **tranzistor BD911**

Max. napětí mezi kol. a emit.  $V_{CEO}$ : 100 V

Max. napětí mezi bází a emit.  $V_{CBO}$ : 5 V

Max. proud tekoucí kolektorem  $I_C$ : 15 A

Maximální výkon  $P_C$ : 90 W

Zesílení  $h_{fe}$ : 5 až 250

Důležité je si všimnout, že minimální zesílení<sup>4</sup> je 5. To znamená, že pokud budeme chtít tranzistorem spínat proud 10 A, tak budeme muset nechat bázi téct řídicí proud 2 A, to znamená, že ho nemůžeme naplno využít, pokud ho budeme řídit mikrokontrolérem, tj. musíme ho spínat jiným tranzistorem. Anebo přijdeme na myšlenku, že pokud budeme chtít řídit velké proudy, budeme potřebovat tranzistory typu MOS-FET, např.: IRF520, IRL3803.

## 5.4 Příloha B – Poznámky a vize

### 5.4.1 Přerušení

Co je přerušení? Procesor může zvládnout pouze jednu operaci na jeden tik krystalu, postupuje od jednoho příkazu k druhému a nemůže jen tak všeho nechat a věnovat se něčemu jinému, občas je to ale potřeba. Při přerušení procesor všeho nechá a bude se věnovat přerušení, potom co skončí se bude věnovat dál programu tam, kde přestal.

### 5.4.2 Baterie

Dobíjecí: napětí 1,2 V

Jednorázové: napětí 1,5 V

### 5.4.3 Vize – co přidat do textu

Github

Lorris

Arduino IDE

Laser ve Lablabu

Osciloskop

Baterie LiPol a jejich nabíjení

Převodník napěťových úrovní

Pájení

Sběrnice - USART/UART, IIC, SPI, další?

---

<sup>4</sup>Platí pokud bude kolektorem protékat 10 A.