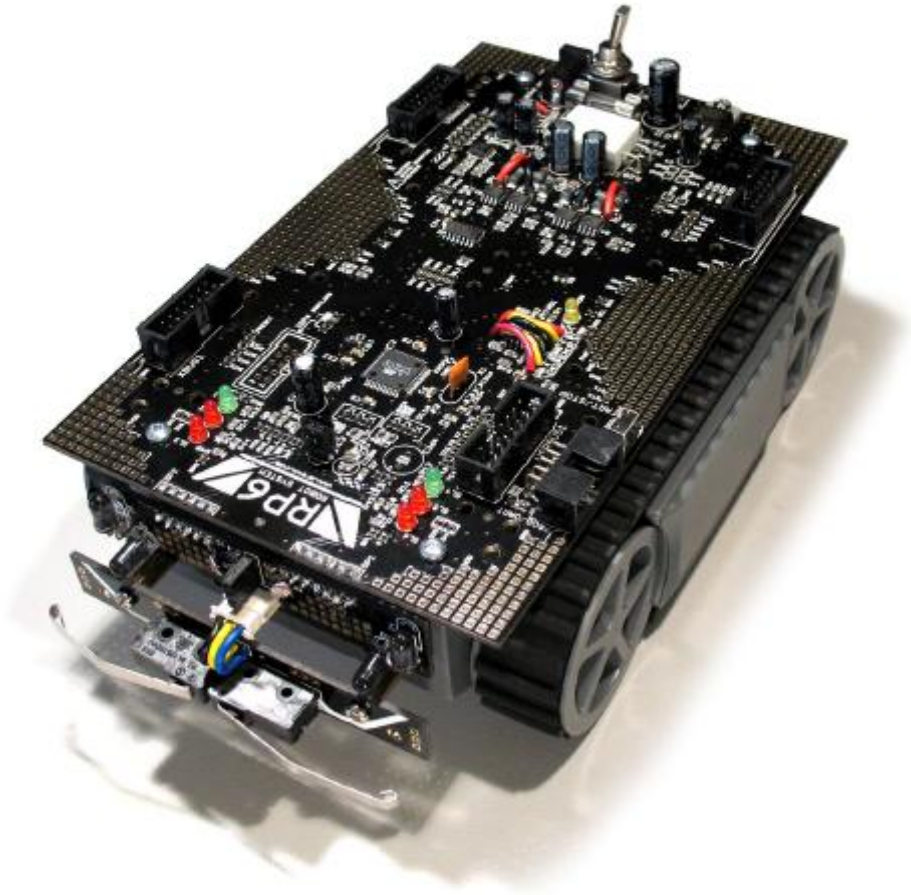
RP6 ROBOT SYSTEM

RP6 ROBOT BASE



AREXX Engineering

RP6-ZÁKLAD

www.arexx.com

Robotický systém
RP6

Příručka

- Čeština -

Version RP6-BASE-CZ-20071029

PŘEDBĚŽNÁ VERZE

DŮLEŽITÁ INFORMACE!
Prosím, čtěte pozorně!

Před zahájením provozu RP6 nebo jiných rozšiřujících přístrojů musíte kompletně přečíst příručku základního modulu i příručky rozšiřujících modulů! Dokumentace obsahuje informace o tom, jak systém správně funguje a jak se vyhnout nebezpečným situacím! Příručky dále poskytují další důležité detaily, které běžný uživatel nemusí znát.
Důsledkem nedodržení upozornění uvedených v této příručce bude ztráta záruky! Firma AREXX Engineering nemůže odpovídat za škody způsobené zanedbáním pokynů z této příručky!

Zvláštní pozornost věnujte kapitole "Bezpečnostní pokyny"!

Nepřipojujte rozhraní USB k osobnímu počítači PC před prostudováním kapitoly 3 – "Nastavení Hardware a Software" a kompletní instalací software!

Symboly

V příručce se používají následující symboly:

Symbol “Výstraha!” se používá k označení důležitých detailů. Nedodržení těchto pokynů může poškodit nebo zničit robot nebo dalších dílů a může ohrozit vaše zdraví!

Symbol “Informace” se používá k označení užitečných tipů a triků nebo základních informací. V tomto případě lze informace označit jako “užitečné, ale ne nezbytné”.

Obsah

1. Úvod

RP6 je levný autonomní mobilní robotický systém, navržený pro začátečníky, kteří mají zkušenosti s elektronikou a vývojem software, jako úvod do fascinujícího světa robotiky.

Robot se dodává kompletně sestavený. To znamená, že je vhodný pro všechny uživatele bez praxe s pájením a mechanickým zpracováním, kteří se chtějí soustředit na vývoj software. Můžete však také implementovat vlastní obvody a přidávat další části robotu! Ve skutečnosti je RP6 otevřený pro řadu rozšiřování a může se používat jako platforma pro širokou škálu zajímavých elektronických experimentů!

RP6 je následník velmi úspěšného "C-Control Robby RP5", který byl představen roku 2003 společností Conrad Electronic SE. Zkratka “RP5” se může interpretovat jako "Robotický Projekt 5". Nový robot a předchozí systém nemají příliš mnoho společného s výjimkou shodné mechanické části. Mikrokontrolér C-Control od firmy Conrad Electronic byl zaměněn a proto se nadále nemůže nový robot programovat v interpretu jazyka Basic. Místo toho je aplikován mnohem výkonnější mikroprocesor ATMEGA32 od výrobce Atmel, který se programuje pomocí jazyka C. Do budoucna plánujeme nabídnout rozšiřující modul pro přizpůsobení robotu k nejnovějším modulům řady C-Control (tj. CC-PRO MEGA 128). Tento modul bude umožňovat programování v mnohem jednodušším jazyku Basic a nabízí velké množství rozšiřujících rozhraní a větší paměť.

Nová konstrukce obsahuje rozhraní USB a nové rozšíření systému se zdokonalenou montáží, odometrické snímače s velkým rozlišením (rozlišení je 150x vyšší ve srovnání s předchozím systémem), přesný stabilizátor napětí (který byl u staršího systému nabízený jen jako rozšiřující modul), nárazníky sestavené ze dvou mikrospínačů s dlouhými rameny a mnoho dalších zdokonalení. Výhodnou součástí systému je i experimentální rozšiřující modul pro sestavení vlastních elektronických obvodů. Ve srovnání s předchozím systémem je mnohem výhodnější poměr ceny a výkonu.

Základní mechanické koncepce byla odvozena ze systému RP5. Konstrukci je však optimalizovaná na nižší provozní hluk a nyní poskytuje další otvory pro mechanické rozšiřování systému.

Robot RP6 byl navržen tak, aby byl kompatibilní s našimi předchozími roboty ASURO a YETI, které používají menší mikroprocesor ATMEGA8 a identické vývojové nástroje (WinAVR, avrgcc). Systémy ASURO a YETI se dodávaly jako konstrukční stavebnice, které si musel sestavit uživatel. S ohledem na to, že byl RP6 navržen pro mnohem náročnější uživatele, umožňuje větší možnosti rozšiřování, obsahuje větší mikroprocesor a více snímačů.

Momentálně je k dispozici několik rozšiřujících modulů, které se mohou použít na rozšíření možností robotu. Například to bude rozšíření systému na předchozí řízení C-Control, rozšiřující modul nabízející zvláštní MEGA32 a samozřejmě experimentální rozšiřující deska pro sestavení vlastních elektronických obvodů, která se dá přikoupit samostatně. Na robot je možné umístit několik modulů.

V blízké době se připravuje několik dalších zajímavých modulů a samozřejmě je možné vyvíjet vlastní rozšiřující obvody!

Přejeme vám mnoho zábavy a úspěchů s robotickým systémem RP6!

1.1. Technická podpora

Na následující adrese můžete prostřednictvím internetu kontaktovat náš podpůrný tým (Před vyžádáním technické podpory si pečlivě prostudujte návod, abychom mohli co nejlépe odpovědět na vaše dotazy! Přečtěte si pozorně také dodatek A – Řešení problémů):

- prostřednictvím našeho fóra: http://www.arexx.com/forum/

- e-mailem: info@arexx.nl

Na začátku příručky najdete poštovní adresu firmy. Všechny aktualizace software, nové verze příručky a další informace budou publikovány na domovské stránce:

http://www.arexx.com/

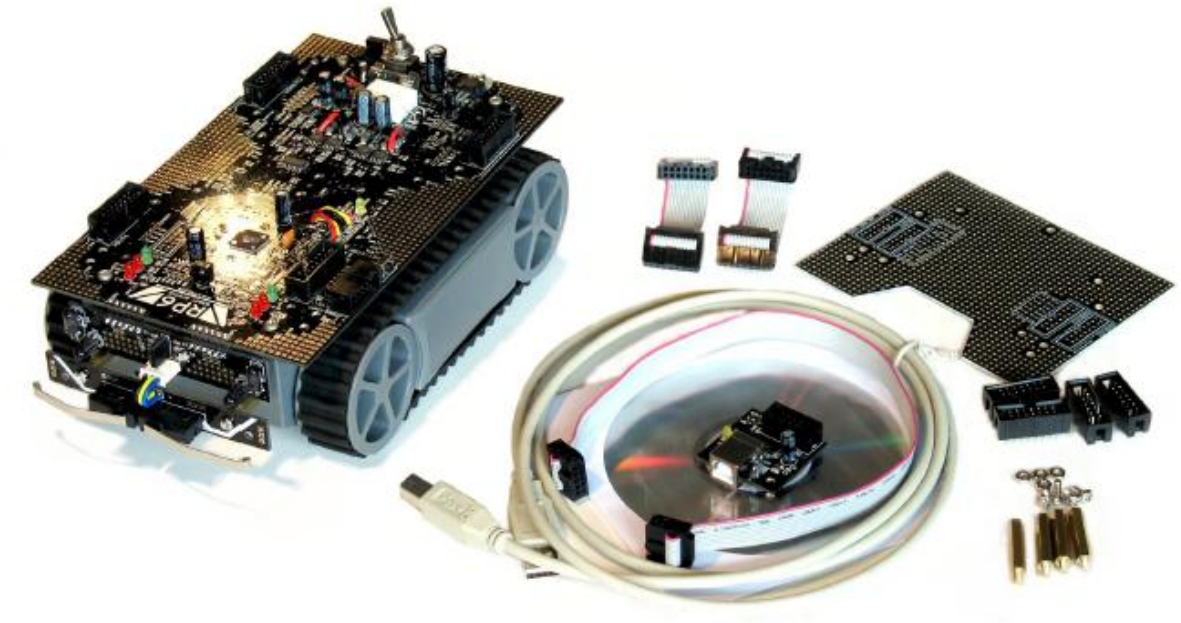
a stránce věnované robotu:

http://www.arexx.com/rp6

1.2. Obsah balení

V krabici s robotem RP6 byste měli najít následující položky:

- **Kompletně sestavený robot**
- Rozhraní RP6 USB
- USB kabel A->B
- 10 žilový plochý kabel
- CD-ROM RP6
- Příručka pro rychlý začátek
- **Experimentální deska RP6**
- 4 ks. šroub M3 x 25 mm
- 4 ks. matice M3
- 4 ks. podložka M3
- 4 ks. 14 vývodový konektor
- 2 ks.14 žilový plochý kabel



1.3. Vlastnosti a technické údaje

Tato část podává přehled vlastností robotu a zavádí některá základní klíčová slova, abyste se seznámili s terminologií používanou v této příručce. Většina těchto klíčových slov bude vysvětlena v následujících kapitolách.

Vlastnosti, jednotlivé části a technické údaje ROBOTICKÉHO SYSTÉMU RP6:

- **Výkonný 8-bitový mikroprocesor Atmel ATmega32**
  - ◊ Rychlost 8 MIPS (= 8 milionů instrukcí za sekundu) při hodinové frekvenci 8 MHz
  - ◊ Paměť: 32 kB Flash ROM, 2 kB SRAM, 1 kB EEPROM
  - ◊ Volně programovatelný v jazyku C (používá WinAVR/avr-gcc)!
  - ◊ … mnoho dalších předností! Další podrobnosti budou uvedeny v kapitole 2
- **Flexibilní rozšiřující systém, založený na sběrnici I^2C**
  - ◊ Potřebuje pouze dva signály (TWI -> "Dvou vodičové rozhraní")
  - ◊ Přenosová rychlost až 400 kbit/s
  - ◊ Architektura Master->Slave
  - ◊ Na sběrnici se může připojit až 127 zařízení typu Slave
  - ◊ Velmi populární sběrnicový systém. Na trhu je nabízeno mnoho standardních integrovaných obvodů, snímačů a dalších modulů, které stačí jen přímo připojit na sběrnici
- **Je možná symetrická montáž rozšiřujících modulů na přední a zadní část robotu**
  - ◊ Teoreticky je možné na robot umístit řada rozšiřujících modulů, ale možnosti napájecího zdroje a přetížení podvozku umožňuje montáž maximálně 6 až 8 modulů (3 až 4 moduly na přední a zadní straně robotu).
  - ◊ Základní deska poskytuje 22 volných montážních otvorů s průměrem 3,2 mm a podvozek nabízí dalších 16 otvorů. Celkem je to 38 montážních otvorů – dále je na podvozku volný prostor pro individuální otvory.

- **Balení obsahuje experimentální desku plošných spojů** (viz fotografie obsahu balení)

- **USB rozhraní** pro programování mikroprocesoru přímo z PC.
  - ◊ Kabelové připojení zajistí maximální přenosovou rychlost. Nahrávání aktualizace programu obvykle probíhá rychlostí 500 kBaud a celý paměťový prostor (30 kB, 2 kB jsou vyhrazeny pro bootloader) se naplní během několika sekund.
  - ◊ Rozhraní se může používat k programování všech dostupných rozšiřujících modulů systému RP6 s mikroprocesorem AVR.
  - ◊ Rozhraní se může použít také pro komunikaci mezi robotem a rozšiřujícími moduly. Připojení můžete například využít k ladění, přenosu textových zpráv a dalších dat do PC.
  - ◊ Ovladač rozhraní umožňuje vytvořit virtuální COM port (VCP) pro všechny populární operační systémy včetně Windows 2K/XP/Vista a Linux. VCP se může používat v běžných terminálových programech a aplikačních programech.

- ◊ Program RP6Loader pro Windows a Linux umožňuje pohodlné nahrávání aktualizovaného programu. Obsahuje také malý terminál pro komunikaci s robotem prostřednictvím textových zpráv.

- **Výkonná jednotka pásového pohonu** v kombinaci s novou převodovkou s minimální hlučností (ve srovnání s předchozím systémem CCRP5…)
  - ◊ Dva výkonné stejnosměrné motory 7,2 V.
  - ◊ Maximální rychlost cca 25 cm/s – závisí na stavu nabití a kvalitě baterií, celkové hmotnosti a dalších podmínkách!
  - ◊ Samo mazná, zapouzdřená ložiska pro všechny 4 hřídele kol.
  - ◊ Dva gumové pásy.
  - ◊ Robot je schopen překonávat malé překážky (s výškou do cca 2 cm), například hrany koberců, práh nebo šikmou plochu se sklonem až do 30% (s namontovanými nárazníkovými spínači). Při odstranění nárazníku a omezením počtu modulů na 2 dokáže robot šplhat do sklonu až 40%.
- **Dva výkonné budiče motorů s tranzistory MOSFET** (H-Bridges)
  - ◊ Rychlost a směr otáčení se může řídit přímo mikroprocesorovým systémem.
  - ◊ **Dva proudové snímače** provádí měření proudu jednotlivých motorů až do velikosti 1,8 A. To umožňuje rychlé rozpoznání zablokování nebo přetížení motorů.
- **Dva enkodéry s vysokým rozlišením** pro řízení rychlosti a trajektorie.
  - ◊ Rozlišení 625CPR ("přírustků na otáčku"), které znamená, že systém napočítá 625 segmentů během jediné otáčky kola! Rozlišení je 150x vyšší než u předchozího systému CCRP5, který měl pouze 4 CPR.
  - ◊ Přesné a rychlé měření a regulace rychlosti!
  - ◊ Velké rozlišení umožňuje měření vzdálenosti cca 0,25 mm na jeden segment snímače.
- **Anti kolizní systém (ACS),** který může detekovat překážky pomocí integrovaného IR přijímače a dvou IR diod umístěných na levé a přední straně robotu.
  - ◊ Detekuje překážky přímo před robotem, vlevo i vpravo od robota.
  - ◊ Nastavení citlivosti a výkonu vysílače umožňuje spolehlivou detekci předmětů se špatným odrazem.
- **Infračervený komunikační systém (IRCOMM)**
  - ◊ Dokáže přijímat signály standardního univerzálního infračerveného dálkového ovladače televize nebo videa. Robot můžete ovládat dálkovým ovladačem se systémem RC5! Protokol se může změnit v software, ale poskytována je pouze implementace standardního protokolu RC5.
  - ◊ Dálkové ovládání se může používat ke komunikaci s více roboty (pomocí přímého dosahu nebo odrazem od stropu a stěn) nebo pro vysílání telemetrických dat.
- **Dva světelné snímače** – tj. pro měření intenzity světla a vyhledávání světelného zdroje.
- **Dva nárazníkové snímače** pro detekci kolize.
- **6 stavových LED** – pro zobrazení stavů snímačů a programu.
  - ◊ Pokud je nezbytné, mohou se pro další funkce použít čtyři LED porty.

- **Dva volné kanály analogově/číslicového převodníku (ADC)** pro externí senzorické systémy (alternativně se mohou použít jako standardní I/O vývody).
- **Přesný stabilizátor napětí 5 V.**
    - ◊ Maximální napájecí proud: 1,5 A.
    - ◊ Rozsáhlá měděná plocha pro odvod tepla do desky plošných spojů.
    - ◊ Trvalý odběr proudu nesmí překročit 1A. Vyšší proud vyžaduje zvláštní chlazení! Doporučujeme, aby byla maximální hodnota trvalého odběru proudu pod 800 mA.
- Výměnná pojistka 2,5 A.
- **Nízký klidový proud** menší než 5 mA (typicky 4 mA a přibližně 17 až 40 mA při používání, který samozřejmě závisí na zatížení a aktivitě systému (LED, snímače atd.). Tyto hodnoty zahrnují pouze spotřebu elektronických obvodů a neberou v úvahu motory a rozšiřující moduly!
- **Napájecí zdroj tvořený baterií 6 nabíjecích článků NiMH** (nejsou obsaženy v balení!).
    - ◊ Doporučené články Panasonic nebo Sanyo (NiMH 1,2 V, 2500 mAh, HR-3U, velikost AA HR6) nebo Energizer (NiMH 1,2V, 2500 mAh, NH15-AA).
    - ◊ Provozní doba přibližně 3 až 6 hodin, závisí na způsobu používání a kvalitě/kapacitě baterie (pokud se příliš často nepoužívají motory, může robot fungovat mnohem déle. Tento údaj o provozní době je uvedena pro vlastní robotický systém bez rozšiřujících modulů).
- **Připojení externí nabíječky baterie** – hlavní spínač napájení se přepíná mezi dvěma polohami “Nabíjení/Vyp” a “Provoz/Zap”.
    - ◊ Pomocí několika pájecích propojek na DPS je možné robot připojit k externímu napájecímu zdroji nebo přídavné baterii.
    - ◊ K nabíjení 6 článků NiMH baterie se může použít libovolný nabíječ. Různé externí nabíječky, které se drasticky liší výkonem a provozními možnostmi, dovedou nabít baterii během 3 až 14 hodin. K nabíjení robotu je potřeba nabíječka s kulatým konektorem o průměru 5,5 mm.
- Hlavní deska poskytuje **6 malých rozšiřujících oblastí** (a dále 2 velmi drobné políčka na malých DPS snímačů v přední části robotu) pro další senzorické obvody tj. k implementaci dalších IR snímačů pro zlepšení detekce překážek. Rozšiřující oblasti se mohou také použít k montáži mechanických součástí.
- **Umožňuje vytvořit velké množství rozšíření!**

Dále dodáváme docela velké množství ukázkových programů v jazyku C a rozsáhlou knihovnu funkcí pro pohodlný vývoj software.

Webové stránky robotu budou brzy nabízet další programy a aktualizaci software určeného pro robotický systém a rozšiřující moduly. Samozřejmě uvítáme nabídku vašich vlastních programů, které zařadíme na internet ke sdílení s ostatními uživateli RP6. Knihovna RP6Library a soubory ukázkových programů jsou šířeny na základě licence Open Source Licence GPL!

1.4. Co RP6 dokáže?

No tak – vyjměte obsah krabice!

Software určuje skutečné chování robotu RP6 - co to přesně bude, to záleží jen na vás a vaší kreativitě naučit robota správně fungovat. Přitažlivost základů robotiky spočívá ve fascinujícím procesu implementace nových nápadů či optimalizaci a zdokonalování existujících věci! Samozřejmě můžete začít tím, že jednoduše spustíte a postupně úpravíte připravené ukázkové programy, které předvádí standardní funkce, ale neomezujte se pouze na to!

Následující seznam zmiňuje pouze několik příkladů dovedností RP6, které můžete dále rozšiřovat. Existují stovky možností (viz příklady na následující stránce).

Základní provedení robotu RP6 může …:

- … autonomní pohyb v prostoru (to znamená nezávislý, bez dálkového ovládání),
- … vyhýbat překážkám,
- … sledovat světelné zdroje a měřit intenzitu osvětlení,
- … detekovat kolize, blokovat motory, sledovat stav baterie a správně reagovat na podněty,
- … měřit a regulovat rychlost otáčení motorů – prakticky nezávisle na stavu baterie, hmotnosti atd. (umožňují to enkodéry s vysokým rozlišením),
- … přesun na určenou, otáčení o zadaný úhel a měření ujeté vzdálenosti (podrobnosti viz kapitola 2),
- … projíždění geometrických obrazců tj. kruhů, polygonů a dalších,
- … výměnu dat s dalšími roboty nebo zařízeními. Povely mohou být přijímány ze standardního TV/video/HiFi dálkového ovládání a robot budete jednoduše ovládat podobně jako dálkově ovládané autíčko.
- … přenos senzorických a dalších dat do PC prostřednictvím rozhraní USB,
- … snadné rozšiřování pomocí flexibilního sběrnicového systému!
- … modifikaci podle vlastních návrhů. Stačí prostudovat schéma a osazení DPS na CD! Při realizaci modifikací dávejte pozor na úplné zvládnutí problematiky! Obvykle je vhodnější projekt realizovat na rozšiřující desce – zvláště pokud nemáte dostatečnou praxi s pájením a sestavováním běžných elektronických obvodů.

1.5. Záměry a plány aplikace

Robot RP6 byl konstruován tak, aby dobře umožňoval rozšiřování. Pokud RP6 vybavíte dalšími senzorickými obvody, můžete robot “naučit” některé z následujících dovedností (některé z následujících úloh jsou docela komplikované a seznam je uspořádán podle složitosti):

- Rozšíření robotu o další ovladače zvyšuje výkon CPU, rozšiřuje paměť nebo jednoduše přidává další I/O porty a ADC, jak bude probráno v ukázkových programech pro snadné rozšíření pomocí I^2C portu a ADC.
- Výstup senzorických dat a textu na LCD displej.
- Reakce na hluk a generované akustické signály.
- Měření vzdálenosti k překážce pomocí ultrazvukových snímačů, infračervených snímačů nebo jiných podobných zařízení, které zajišťuje lepší předvídání kolize.
- Sledování černých čar na podlaze.
- Sledování a následování dalších robotů nebo objektů.
- Ovládání robotu z PC pomocí infračervených signálů (k tomu je třeba speciální hardware, robot bohužel nedokáže spolupracovat se standardním rozhraním IRDA). Alternativně můžete začít přímo používat bezdrátové VF moduly.
- Ovládání RP6 pomocí PDA nebo Smartphone (v takovém případě doporučujeme zabudovat tato zařízení přímo do robotu místo používání jako dálkového ovladače, jsou však možné obě řešení!).
- Shromažďování předmětů (např. čajových svíček, kuliček, drobných kovových předmětů …).
- Připevnění malého robotického ramene k uchopení předmětů.
- Navigace pomocí elektronického kompasu nebo infračervených paprsků (realizované malou věžičkou s řadou IR LED a polohování do známých směrů) pro určení polohy robotu a vyhledání zadané polohy.
- Nabízí řadu robotické výbavy včetně kopání do balónu, obslužných mechanismů a nějakých zvláštních snímačů, které umožňují zařazení do závodních týmů pro soutěže v robotickém fotbalu!
- … mnoho dalšího, co vás může napadnout!

Nejdříve byste však měli přečíst příručku a seznámit se s robotikou a programováním. Předchozí seznam nápadů je pouze zlomek možností a základ motivace.

A pokud se vám programování nepodaří na poprvé, hned to nevzdávejte a nevyhazujte z okna: každý začátek je těžký!

2. RP6 podrobně

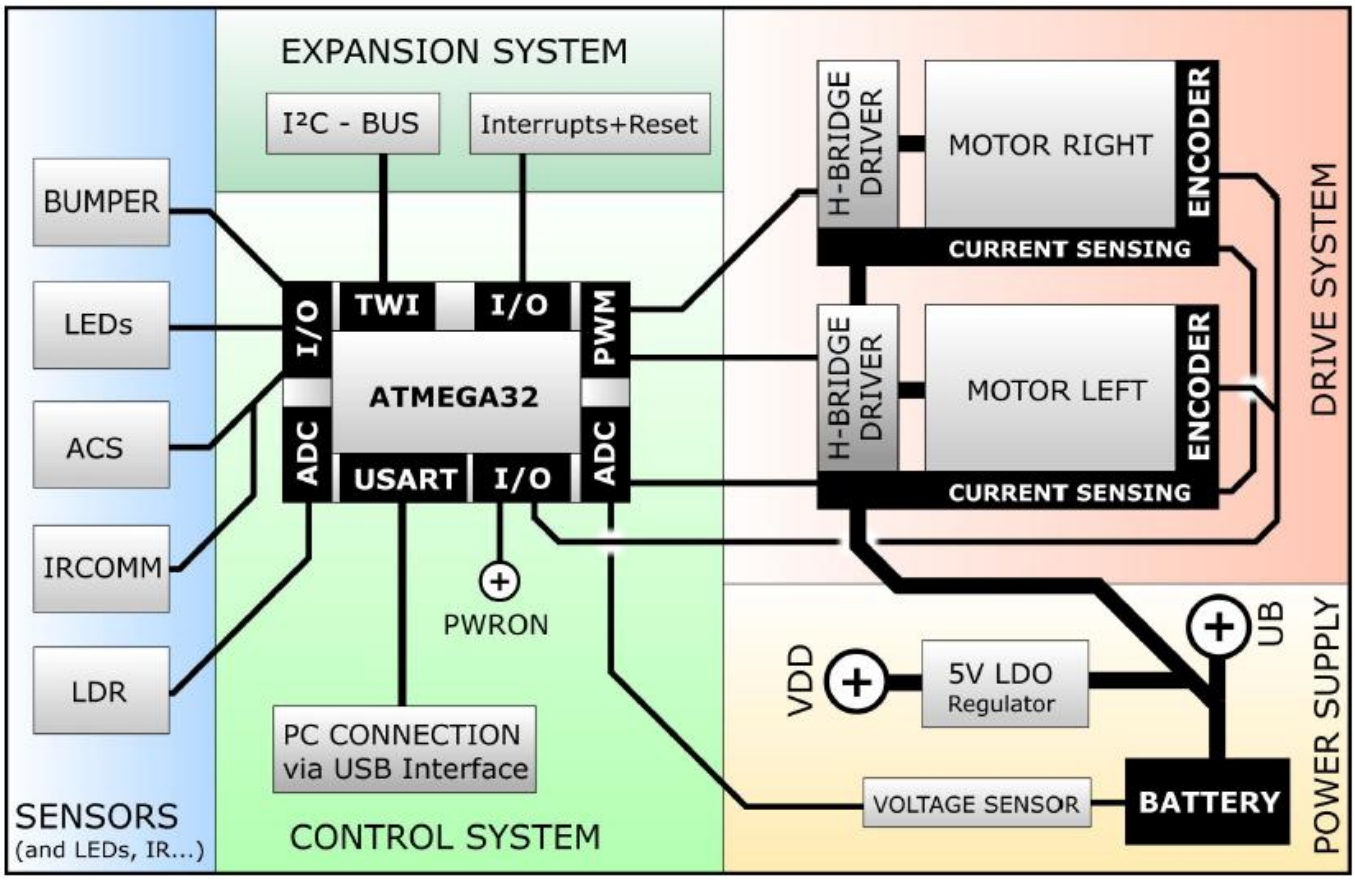
Tato kapitola popisuje nejdůležitější části hardware ROBOTICKÉHO SYSTÉMU RP6. Probereme zde elektroniku, mikroprocesor a propojení software a hardware. Pokud již ovládáte technologii mikropočítače a elektroniky, pravděpodobně tuto kapitolu pouze letmo prohlédnete. Začátečník by však měl tuto kapitolu pečlivě prostudovat, aby získal základní poznatky o RP6.

Pokud nechcete čekat a raději otestujete robot, přejděte na kapitolu 3, ale k této kapitole se později vraťte, protože obsahuje řadu užitečných vysvětlení podrobností programování robotu. A vy přece chcete znát, co se řídí pomocí software a jak vše funguje, že ano?

Nebudeme se nořit do detailů, ale několik témat v této kapitole může být složitější na pochopení – autor se pokoušel vše vysvětlit co nejlépe.

Pokud chcete podrobně studovat některou problematiku, můžete vyhledat další informace na webových stránkách http://www.wikipedia.org/, která poskytují dobrou výchozí pozici pro většinu témat.

Obrázky často řeknou více než slova a proto začneme s přehledným blokovým schématem RP6. Obrázek ukazuje drasticky zjednodušené schéma elektronické části robotu a vzájemné propojení:



Robot můžeme rozdělit na pět hlavních funkčních jednotek:

- Řídicí systém
- Napájecí zdroj
- Snímače, IR komunikace a displeje (senzorika) – vše komunikuje s okolním světem a měří fyzikální veličiny
- Systém pohonu
- Rozšiřující systém

2.1. Řídicí systém

Jak vidíte na blokovém schématu, centrální jednotkou robotu je 8-bitový mikrokontrolér ATMEL ATmega32.

Mikrokontrolér je kompletní jednočipový mikropočítač. Tento mikroprocesor se liší od velkých počítačů (jako je třeba PC) tím, že poskytuje méně periferií. Malý mikroprocesor samozřejmě nemůže obsahovat mechaniku normálního hard disku a paměť RAM o velikosti několika GB. Mikrokontrolér nepotřebuje příliš velkou paměť. MEGA32 nabízí "pouze" 32 kB (32768 byte) Flash ROM – který můžeme srovnat s normální "mechanikou hard disku" nebo nověji s flashdrive. Flash ROM se používá k uložení všech programových dat. Velikost paměti s libovolným přístupem (RAM) je omezena na 2 kB (2048 byte) a přitom bude dostatečná pro vaše potřeby. Pro představu srovnání s řadičem staršího robotu CCRP5 s pouhými 240 byte RAM, která byla celá vyhrazena pro interpret jazyka Basic.

Ale proboha, jak může mikroprocesor fungovat s tak malou kapacitou paměti? Je to jednoduché: procesor nikdy nezpracovává velké množství dat jako operační systém Linux nebo Windows a nedokáže realizovat složité grafické rozhraní nebo podobné úkoly. Zde poběží pouze jeden program a bude to vás vlastní software!

Toto omezení není žádná nevýhoda, ale jedna ze základních předností mikroprocesorových systémů ve srovnání s velkými počítači (dále můžeme zmínit spotřebu energie, velikost a cenu)! Mikroprocesor je navržen pro zpracování úloh ve známých časových intervalech (často se označuje jako zpracování "v reálném čase"). Obvykle mikroprocesor nesdílí napájení s řadou dalších mikroprocesorů jako je tomu v běžném PC a programátor se nemusí zabývat určením časového rozložení speciálního funkčního modulu.

Řídící jednotka RP6 běží na frekvenci 8 MHz, která dovoluje zpracování programu rychlostí 8 milionu instrukcí za sekundu. Mikroprocesor sice umožňuje časování frekvencí až 16 MHz, ale nižší frekvence se používá kvůli snížení spotřeby systému. Stoj zůstává dostatečně rychlý pro zpracování všech standardních úloh! Rychlost můžeme opět porovnat se starším předchůdcem CCRP5 s hodinami 4 MHz, které umožňovaly zpracování pouze 1000 (interpretovaných) instrukcí jazyka Basic během jedné sekundy. Z tohoto důvodu bylo ACS řízení staršího robotu koncipováno s dalším slave řadičem – už nikdy nebudeme potřebovat tento podružný mikroprocesor! Více procesorů má navíc větší spotřebu energie a rozsáhlejší rozhraní. Na RP6 je možné přidat rozšiřující řídící modul M32, který obsahuje další MEGA32 časovaný maximálně frekvencí 16 MHz.

Mikrokontrolér komunikuje s okolním světem přes 32 I/O vývodů ("vstup/výstupní piny"), uspořádaných do "portů" po 8 I/O vývodech. Tímto způsobem poskytuje MEGA32 4 "porty": PORTA až PORTD. Mikroprocesor je schopen číst logické stavy těchto portů a získanou informaci zpracovat programem. Mikroprocesor bude samozřejmě používat také výstup logických signálů na portech pro ovládání malých zátěží do maximálního proudu 20 mA (například LED).

Mikroprocesor dále poskytuje řadu integrovaných hardwarových modulů určených pro speciální úlohy. Implementace těchto úloh v software může být velmi komplikované nebo nemožné. Jednou z takových speciálních funkcí je časovač. K dispozici jsou tři časovače na čítání hodinových period. Časovače jsou naprosto nezávislé na běhu programu. Ve skutečnosti může mikroprocesor zpracovávat jinou práci, dokud nenastane požadovaný stav čítače.

RP6 používá jeden časovač na generování PWM signálů (PWM = "pulsně šířková modulace") pro regulaci rychlosti motorů a tak časovač může přijímat příslušné vstupní parametry, které zvládnou tuto úlohu na pozadí. Generování PWM signálu podrobně probereme v kapitole “Systém pohonu”.

Další moduly MEGA32 například jsou:

- Sériové rozhraní (UART) pro komunikaci RP6 s PC přes sběrnici USB. Pokud není zapojena sběrnice USB, může se pomocí tohoto rozhraní připojit další mikroprocesor s USART.
- Modul "TWI" (= "dvou vodičové rozhraní") poskytuje sběrnici I^2C pro rozšiřující moduly.
- Analogově-číslicový převodník (ADC) poskytuje 8 vstupních kanálů pro měření napětí s 10-bitovým rozlišením. RP6 používá ADC ke sledování napětí baterie, snímačů proudu motorů a intenzitu světla se dvěma fotorezistory.
- Tři vstupy externího přerušení pro generování signálů, které budou přerušovat chod programu v řídící jednotce a vynutí skok do speciální "obsluhy přerušení". Mikroprocesor zpracuje obsluhu přerušení a okamžitě se vrátí do normálního programu. Tuto programovou vychytávku budeme používat pro snímače orometrie. Tento snímač podrobně probereme později.

Integrované hardwarové moduly nemají vlastní individuální vývody, ale mohou se použít alternativně místo standardních I/O vývodů. Běžně se tyto speciální funkce volně mapují na I/O vývody, ale RP6 je téměř všechny vývody standardně nakonfigurovány (protože jsou trvale připojené k ostatním elektronickým obvodům) a modifikace bude komplikovaná.

MEGA32 nabízí řadu dalších možností, které nemohou být podrobně popsané v této příručce. Více informací získáte v katalogových listech jednotlivých výrobců (které můžete najít na RP6 CD-ROM).

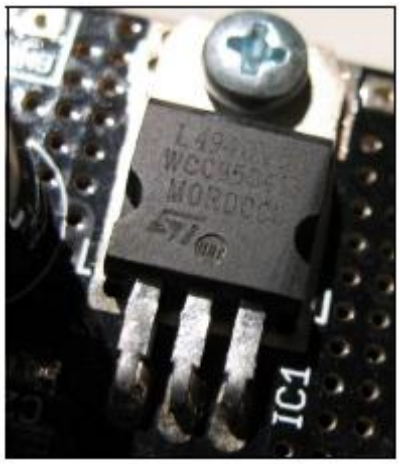
2.1.1. Bootloader

Ve speciální části paměti mikroprocesoru je umístěn tzv. bootloader. Tento krátký program zajišťuje nahrávání uživatelských programů do paměti počítače přes sériové rozhraní. Bootloader komunikuje s programem RP6Loader, který běží na nadřízeném PC. Při takovém programování není potřeba další hardware. USB rozhraní se může použít ke komunikaci s mikroprocesorem pomocí textových zpráv a dále pro programování mikroprocesoru. Používání bootloaderu má však jednu nevýhodu: zabírá 2 kB paměti FLASH a pro vlastní program zbývá 30 kB volné paměti. Toto omezení nijak neomezuje, protože je dostatek místa i pro velmi složité programy (v porovnání s robotem ASURO, kde je k dispozici 7 kB volné paměti)!

2.2. Napájecí zdroj

Robot samozřejmě potřebuje energii. RP6 získává tuto energii v podobě baterie složené ze šesti akumulátorů. Provozní čas bude velmi záviset na kapacitě baterie, a protože elektronické systémy budou spotřebovávat relativně malé množství energie bude hlavní spotřeba energie v motorech, která závisí na zatížení. Dostatečně dlouhou provozní dobu můžete zajistit oblíbenými bateriemi s kapacitou vyšší než 2500 mAh. Dostatečné budou i baterie s kapacitou 2000 mAh. Velmi kvalitní baterie umožní provozní dobu 3 až 6 hodin, podle zatížení motorů a kvality nabití. Budete potřebovat 6 kusů baterií, které společně dávají napětí 6 x 1,2 V = 7,2 V. V blokovém schématu je toto napětí označeno jako "UB" (= "U-baterie", U je standardní písmeno používané v elektrotechnických vzorcích pro napětí). "UB" je definováno jako jmenovité napětí, které se může časem měnit. Úplně nabitá NIMH baterie může dodávat až 8,5 V! Při vybíjení baterie se napětí snižuje a může se prudce změnit podle zátěže a kvality článků. Kritický faktor kvality článků je vnitřní odpor.

Proměnné napětí samozřejmě není vhodné pro senzorická měření. Mnohem důležitější je však omezený rozsah provozního napětí polovodičových obvodů. Například mikroprocesor se poškodí při napájecím napětí větším než 5 V. Proto se musí napájecí napětí snížit a stabilizovat na přesně definovanou hodnotu.



Napájení elektroniky je vyřešeno pomocí integrovaného stabilizátoru napětí, který je schopen dodávat proud až 1,5 A (viz obrázek). Při odběru 1,5 A se tento stabilizátor zahřívá, proto je umístěn na velké měděné ploše DPS Toto chlazení však omezuje odběr proudu na maximálně 1A po dobu několika sekund. Pro větší odběry proudu se musí instalovat přídavný chladič. Trvalý odběr proudu je omezen přibližně na 800 mA. Větší zátěž navíc rychle vybije baterii.

Při normální zátěži elektronikou bez rozšiřujících modulů nebude robot odebírat více než 40 mA, který se sníží vyřazením vysílače IRCOMM. Tato hodnota proudu není žádný problém pro stabilizátor a tak je možné připojit řadu experimentálních desek. Rozšíření elektroniky obvykle zvýší odběr proudu o maximálně 50 mA, pokud neobsahuje zátěže jako jsou motory nebo LED.

2.3. Senzorika

Většina snímačů byla zmíněna v předchozích kapitolách, ale nyní se na ně víc zaměříme.

V přehledném schématu najdete snímače v krajní modré oblasti “Senzory”. Některé snímače zasahují do dalších modulů. I když mezi ně patří odometrické enkodéry, snímače proudu a napětí baterie budou také probrány v této kapitole!

2.3.1. Snímač napětí baterie

Základem tohoto snímače je jednoduchý dělič napětí složený ze dvou rezistorů. Můžeme předpokládat, že maximální napětí baterie bude 10 V. Šest NiMH článků bude mít zaručeně menší hodnotu. Referenční napětí ADC, které se při měření porovnává, je nastaveno na 5 V. Vstup nikdy nesmí překročit napájecí napětí mikroprocesoru 5 V. Z tohoto důvodu se musí sledované napětí dělit dvěma. Dělič napětí musí zajistit, aby sledované napětí odpovídalo napěťovému rozsahu převodníku.

AD převodník měří napětí s rozlišením na 10 bitů (10 V se převede na hodnotu v rozsahu 0 až 1023 jednotek), výsledkem je rozlišení napětí 10V/1024 = 9.765625mV. Změřená hodnota 512 jednotek odpovídá hodnotě 5 V a 1023 přibližně 10V. Tuto hodnotu nemůže 6 běžných NiMH baterií překročit!

Měření není příliš přesné, protože nepoužíváme přesné rezistory. Přesnost se pohybuje v jednotkách procent. Referenční napětí není přesné a může kolísat podle zatížení napájecího zdroje. Tato nepřesnost nás neznepokojuje, protože potřebujeme pouze rozpoznat hranici vybití baterie. Pokud potřebujete přesně určit napětí, musíte pomocí altimetru změřit přesnou hodnotu napětí a pak upravit hodnoty v software.

Pokud akceptujete tolerance, můžete napětí určit přímo z hodnot AD převodníku: 720 jednotek zhruba odpovídá 7,2 V; 700 na 7,0 V a 650 na 6,5 V. Konstantní hodnota 560 se může chápat jako prázdná baterie.

2.3.2. Světelné snímače (LDR)

Malá destička plošných spojů snímače na přední straně robotu obsahuje dva tak zvané LDR (= "rezistor citlivý na světlo"), které míří na levou respektive pravou stranu. Mezi dvěma senzory je černá přepážka, která brání dopadu světla na “špatnou” stranu systému světelného snímače. Jelikož má snímač napěťový výstup, tvoří oba světelné snímače společně s pevnými rezistory napěťové děliče, určující intenzitu osvětlení. V tomto případě se 5 V dělí na hodnotu určenou proměnným rezistorem. Dělicí poměr se mění podle intenzity dopadajícího světla a poskytuje napětí závislé na světle, které se přivádí na jeden z kanálů ADC!

Rozdíl napětí mezi oběma snímači se může použít k určení, na které straně robotu je umístěn jasnější zdroj světla: vlevo, vpravo nebo uprostřed. Vhodný program může sledovat jasnou svítilnu ve tmavé místnosti nebo navádět robota do nejvíce osvětlené části podlahy.

Samozřejmě se můžete pokusit o opak: robot můžete naprogramovat tak, aby se skrýval před světlem.

Systém světelného snímání můžete zdokonalit montáží jednoho nebo dvou dalších LDR na boční strany robotu. Výchozí používání pouze dvou snímačů nemusí dobře rozlišit světlo na přední a zadní straně. Dva kanály AD převodníku jsou stále volné…

2.3.3. Anti kolizní systém (ACS)

Z pohledu software je nejsložitější snímač ACS - “Anti kolizní systém”! ACS tvoří integrovaný obvod infračerveného (IR) přijímače (viz obrázek) a dvě IR LED umístěné na levé a pravé straně přední senzorické DPS. Mikroprocesor přímo ovládá IR LED. Obslužná funkce se může změnit a upravit přesně podle vašich potřeb! Předchozí model robotu měl k tomuto účelu speciální řadič a uživatel nemohl modifikovat software tohoto zařízení.

IR LED vysílají krátké infračervené impulsy modulované na kmitočtu 36 kHz, které může detekovat IR přijímač. Jakmile se IR impulsy odrazí od předmětu zpět a zachytí je IR přijímač, může mikroprocesor reagovat na tuto situaci a spustit únikový manévr. Aby se potlačila příliš velká citlivost, zpozdí ACS rutina detekci událostí dokud systém nepřijme definovaný počet impulsů během krátké časové periody. Dále ACS synchronizuje detekci pomocí rutiny pro příjem kódování RC5 a robot nebude reagovat na signály z televizního dálkového ovladače. Jiné kódy však mohou se systémem ACS interferovat a robot se může pokusit vyhnout neexistující překážce!

Díky tomu, že má ACS systém umístěnu jednu IR LED na levé a druhou na pravé straně, může snadno určit, zda se překážka nachází vlevo, vpravo nebo přímo před robotem.

Systém umožňuje změnu intenzity impulsů obou IR LED ve třech úrovních. Ale při nejvyšší hodnotě proudu nemůže ACS spolehlivě detekovat všechny překážky. To velmi závisí na odrazových vlastnostech povrchu překážek!

Černý předmět bude samozřejmě IR světlo odrážet méně než bílá překážka a předmět s reflexními hranami může nasměrovat IR světlo přímo do několika zvláštních směrů. Z těchto důvodů dosah ACS drasticky závisí na povrchu překážek! Tato závislost musí být považována za základní nevýhodu všech infračervených senzorických systémů (obzvlášť v této cenové kategorii).

Robot přesto může bezvadně rozpoznávat a obcházet překážky. Pokud selže ACS detekce, zůstávají v činnosti nárazníky s dotykovými snímači. A když selžou i dotykové snímače, může robot pomocí snímače proudu nebo enkodéru zjistit zablokování motoru!

Pokud nebudete spokojeni s tímto senzorickým systémem, můžete na robot namontovat například nějaké ultrazvukové snímače.

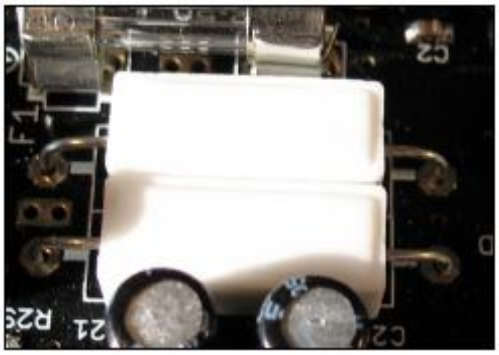
2.3.4. Nárazníky

Malá destička osazená dvěma mikrospínači s dlouhými páčkami je umístěná na přední části robotu. Tato destička chrání IR LED snímače před mechanickým poškozením, pokud robot nešťastně narazí na překážku. Pomocí mikrospínačů může mikropočítač detekovat kolize, couvnout nebo zatočit a pak znovu jet dopředu.

Spínače jsou připojeny na porty již používané pro LED. Proto nezabírají volné porty mikroprocesoru. Toto dvojité využití způsobí, že se LED rozsvítí, jakmile se sepne některý spínač! Spínače se však stisknou jen občas a aktivace LED nebude rušit.

Destička nárazníků se může odmontovat a příležitostně nahradit například kopacím/záchytávacím zařízením pro balóny.

2.3.5. Snímače proudu motoru



Každý ze dvou snímačů proudu motoru obsahuje výkonový rezistor. Ohmův zákon U = R•I říká, že úbytek napětí na rezistoru je přímo úměrný proudu, který přes něj protéká!

Aby nebyl úbytek napětí příliš velký, musí se zvolit velmi malá hodnota odporu. Zde jsme použili 0,1 ohmu.

Při tak nízké hodnotě je úbytek napětí velmi malý (0,1 V při proudu 1 A) a před přivedením na vstup AD převodníku se musí zesílit. Zesílení se realizuje pomocí operačního zesilovače. RP6 používá samostatný operační zesilovač pro každý individuální proudový snímač. Měřicí rozsah proudu je přibližně 1,8 A. Výsledkem tohoto proudu je úbytek napětí na výkonovém rezistoru 0,18 V a na výstupu operačního zesilovače napětí asi 4 V. To je maximální výstupní napětí operačního zesilovače napájeného ze zdroje napětí 5 V.

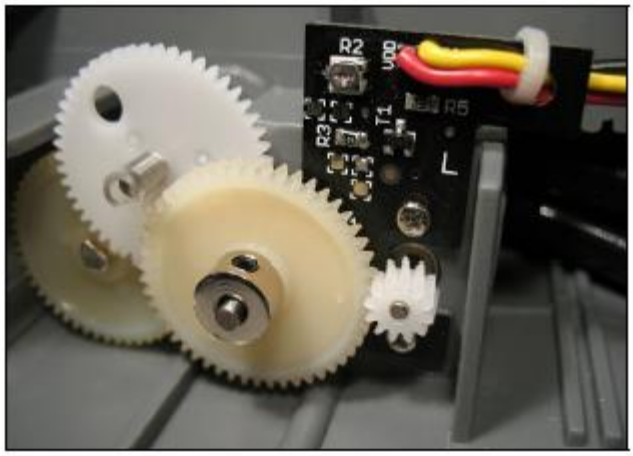
Použité typy výkonových rezistorů mají toleranci hodnoty 10 %, rezistory u operačních zesilovačů 5 %. Všechny součástky jsou nepřesné, a pokud neprovedete kalibraci, můžete zjistit odchylku měřené hodnoty až 270 mA! My však potřebujeme pouze zjistit úroveň proudu, která odpovídá podmínkám kritického zatížení motoru. Robot bude schopen detekovat blokování/přetížení motoru popřípadě poruchu motoru nebo odometrických snímačů! Stejnosměrné motory odebírají při větší zátěži (momentu) větší proud. Při zablokovaných motorech se rapidně zvýší protékající proud. Tento stav se rozpozná v software a spustí se nouzové odpojení. Pokud by se tak nestalo, budou se motory velmi zahřívat (a také přetěžovat) a časem dojde k jejich poškození.

Pokud selžou enkodéry – jakýmkoliv způsobem – může systém tento stav také spolehlivě rozpoznat. Měření rychlosti samozřejmě spadne na nulu. Ale pokud se motor pohání plným výkonem a proudové snímače detekují pouze malý proudy (které značí, že motor není zablokován) můžete vyvodit závěr, že je poškozen motor, enkodér nebo obě zařízení. Tento stav může například nastat, když se v programu zapomenou aktivovat snímače…

2.3.6. Enkodéry

Enkodéry fungují naprosto jinak než dříve probrané snímače. Tvoří je reflexní optické snímače a kódovací kola připevněná na jednom převodovém kolu v každé převodovce. Tato sestava se používá k určení rychlosti otáčení motorů. Obě kola enkodérů mají 36 segmentů (jak ukazuje obrázek, jedná se o 18 černých a 18 bílých políček). Jakmile se převodovky otáčí, pohybují se tyto segmenty před reflexním snímačem. Bílé segmenty odráží IR světlo, zatímco černé budou odrážet jen zanedbatelné množství světla. Stejně jako u ostatních snímačů produkují enkodéry analogový signál, který ale bude interpretován číslicově. Nejprve se signál zesílí a následně se Schmitovým klopným obvodem tvaruje na pravoúhlý signál. Náběžná i sestupná hrana signálu (změny z 5 V na 0 V a z 0 V na 5 V) spustí přerušení tato událost se započítá programem. Tímto způsobem se může měřit vzdálenost a společně s časovačem se může vypočítat rychlost.

Určení rychlosti je hlavní aplikace enkodérů. Zpětná vazba z enkodérů je jediný spolehlivý způsob regulace rychlosti motoru. V neřízeném systému by mohla rychlost motoru záviset na napětí baterie, zatíženi a parametrech motoru. Vysoké rozlišení enkodérů umožňuje spolehlivou regulaci i při docela malých rychlostech.



V každém z obou převodových složení má střední převodový systém 50 zubů a malé vnitřní převodové kolo 12 zubů (viz obrázek). Kódovací kolečka jsou umístěna na převodovém kole, které je umístěno za pastorkem motoru, tak lze vypočítat:

$$\frac{50}{12} \bullet \frac{50}{12} = 17\frac{13}{36}; 17\frac{13}{36}36 = 625$$

Přestože je zde 36 segmentů musí být výsledek pro celou otáčku kola celé číslo bez zlomkové části. Enkodéry generují 625 hran na jednu otáčku a každá hrana představuje jeden segment.

Rozměr kola včetně gumového pásu je kolem 50 mm a teoreticky získáme, při obvodu cca 157 mm, krok 0,2512 mm na každou započítanou jednotku enkodéru. Sledování dráhy však může být deformováno díky tlaku nebo nedostatečně poddajným povrchem. Proto můžeme uvažovat maximálně 0,25 mm na každou započítanou jednotku. Často bude lepší aplikovat 0,24 mm nebo 0,23 mm. Kalibrační hodnoty se mohou určit pohonem na přesně definovanou vzdálenost, jak je popsáno v dodatku. Měření není příliš přesné díky prokluzu nebo podobným vlivům. Při přímé jízdě vpřed bude mít enkodér minimální chybu přesnosti, ale při zatáčení robotu budou růst odchylky výsledku. Největší odchylku způsobí speciálně otáčení robotu na místě.

Odchylky mohou být zjištěny a opraveny testováním, zkoušením a chybami. Je to nevýhoda všech pásových pohonů – v našem robotu i mnohem dražších systémech. V porovnání s roboty se standardní diferenciální pohonnou jednotkou se dvěma koly a přídavným podpůrným kolečkem umožňují pásové systémy lepší chování při jízdě v různorodém prostředí. Pásový pohon bude snadno překonávat malé překážky, šikmé plochy a hrbolaté podlahy. Na každém povrchu jsou extrémně užitečné enkodéry, protože umožňují regulaci rychlosti při všemožném zatížení, kompletně nezávislé na kvalitě povrchu, zátěži motoru a napětí baterie.

Pokud budeme uvažovat hodnotu 0,25 mm na segment, pak při rychlosti 50 segmentů za sekundu dostaneme rychlost 1,25 cm/s. Tato rychlost je minimální, která se může spolehlivě regulovat (při implementaci nejnovější verze standardního software). Přesná hodnota se může u jednotlivých robotů lišit. Rychlost 1200 segmentů za sekundu odpovídá maximální dosažitelné rychlosti 30 cm/s (rozlišení 0,25 mm, kdežto 0,23 mm koresponduje s 27,6 cm/s). Maximální rychlost závisí na stavu nabití baterie a při běžných bateriích nelze 30 cm/s dosáhnout na delší dobu. Z tohoto důvodu je knihovní funkce omezena na 1000 segmentů/sekundu, která udržuje konstantní maximální rychlost během delší doby vybíjení baterie. Dále se při nižších rychlostech prodlužuje životnost převodovek a motorů!

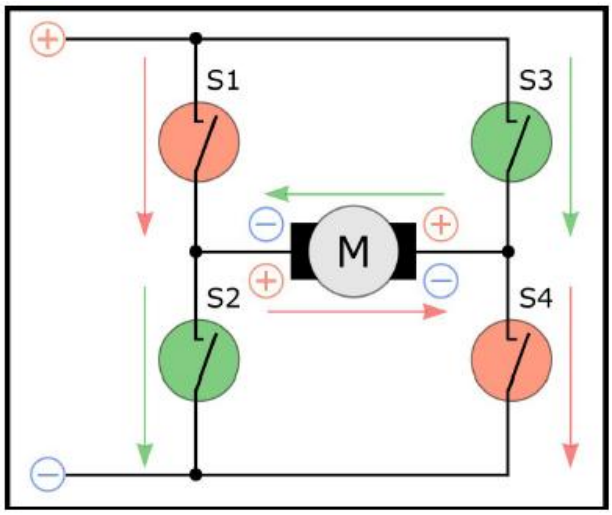
Jakmile robot napočítá 4000 segmentů, urazí se vzdálenost přibližně jeden metr. Jak již bylo vysvětleno, tato specifikace je platná pro rozlišení přesně 0,25 mm – bez správné kalibrace musíme uvažovat větší či menší odchylky. Pokud vás nezajímá přesný výpočet vzdálenosti, nemusíte kalibrovat enkodéry a jednoduše uvažovat hodnotu 0,25 mm nebo ještě lépe 0,24 mm!

Dobré navigační systémy nespoléhají při řízení vzdálenosti a úhlovém natočení zcela na enkodérech, ale používá externí pevné značky, jako jsou infračervené majáky a přesné elektronické kompasy. Používání externích systémů je obvykle dobrý nápad pro častou korekci odchylek orometrie.

2.4. Pohonný systém

Pohonný systém RP6 se skládá ze dvou stejnosměrných motorů s přídavnou převodovkou pro pohon pásových kol (viz předchozí obrázek). Motory mohou spotřebovat docela velké množství energie a mikroprocesor nemůže přímo dodávat tak velké proudy.

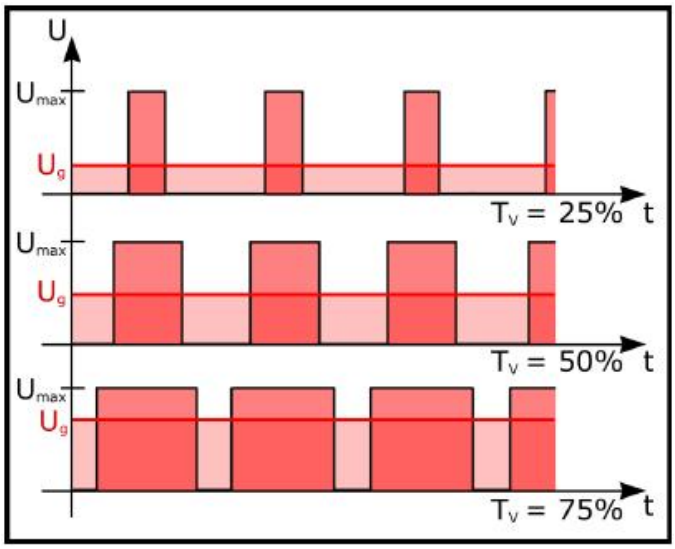
Z tohoto důvodu potřebujeme výkonný budič motoru. Pro řízení motorů v robotu RP6 používáme dva tak zvané H-můstky. Schéma na pravé straně ukazuje základní princip budiče. Můžete zde také vidět, že spínače a motor společně tvoří písmeno “H”.



Nyní uvažujeme všechny spínače jako otevřené, Pokud sepneme spínače S1 a S4 (červené) bude se na motor přivádět napětí a ten se začne otáčet, řekněme doprava. Pokud nyní znovu otevřeme spínače S1 a S4 a následně sepneme spínače S2 a S3 (zelené), přivede se napětí s opačnou polaritou a motor se začne otáčet opačným směrem (doleva). Samozřejmě musíme dávat pozor na to, aby se současně neseply spínače S1 a S2 nebo S3 a S4. Každá tato kombinace může mít za výsledek zkrat obvodu a může poškodit spínače.

Konstrukce RP6 samozřejmě nebude používat mechanické spínače, ale tranzistory MOSFET, které vedou, pokud je na jejich hradlo přivedeno vhodné napětí. MOSFET může spínat velkou rychlostí v řádu několika kHz.

Nyní jsme nalezli způsob jak měnit směr otáčení motoru, A jak můžeme realizovat zrychlování nebo zpomalování motoru? Stejnosměrný motor se bude otáčet rychleji, když se přivede vyšší napětí a rychlost motoru můžeme regulovat zvyšováním nebo snižováním napětí. Podívejme se znovu podrobně na H-můstek.



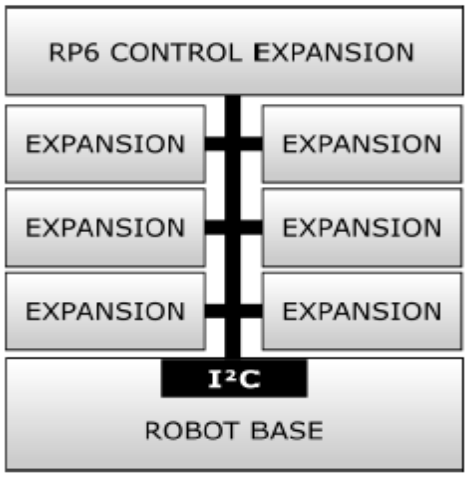
Obrázek ukazuje, co můžeme dělat, Generujeme pravoúhlý průběh s pevným kmitočtem a zavedeme pulsně šířkovou modulaci, která mění střídu. "Střída" znamená poměr mezi vysokou a nízkou periodou signálu.

Nyní bude motor dostávat menší střední hodnotu stejnosměrného napětí, která odpovídá střídě.

Na grafu je toto chování vyznačeno červenou čarou (Ug) a červenou plochou pod čarou. Pokud je například na řídící obvod motoru přivedeno napětí baterie 7 V a motor se reguluje PWM signálem se střídou 50 %, bude průměrná hodnota stejnosměrného napětí 3,5 V. Tento výklad přesně neodpovídá podmínkách reálného elektronického obvodu, ale je dobrou vizualizací principu.

Robot RP6 výhodně používá převodovku s velkým redukčním poměrem (~ 1:72). Díky tomu dostává robot skutečně silný pohon, který umožňuje, ve srovnáním s malými roboty typu ASURO, převoz větších zátěží. Při zvyšování hmotnosti se však musí počítat s větším zatížením napájecího zdroje, jehož důsledkem se zkrátí doba vybíjení baterie…

V porovnání s dálkově ovládaným závodním autem můžeme se domnívat, že je robot RP6 pomalé vozítko – což je naprostá pravda – ale robot byl záměrně konstruován pro pomalou jízdu. Robot je postaven pro ovládání mikroprocesorem a pokud programátor vytvoří chyby v software, může být nevýhodné, kdyby robot narazil do zdi rychlostí třeba 10 m/s. Díky volnější rychlosti se robot nedostane do problémů, protože pomalejší pohyb poskytne dostatek času pro reakci snímačů na překážky. Robot je navíc výkonnější a získá přesnější regulaci rychlosti. Menší rychlost umožňuje pomalý pohyb robotu RP6 konstantní rychlostí.



2.5. Rozšiřující systém

Jednou z nejužitečnějších vlastností RP6 je rozšiřující systém, který umožňuje snadné přidávání dalších komponent na základ robotu. Základní platforma RP6 obsahuje dostatek snímačů. Stávající počet snímačů přesahuje běžné vybavení srovnatelných robotů v dané cenové kategorii, ale robot se po přidání několika senzorických modulů stane mnohem zajímavější. Systém ACS pak bude například pouze detekovat existenci překážek před robotem. Použití ultrazvukových snímačů nebo dokonalejších IR snímačů můžete být schopni určit vzdálenost a zahájit sofistikované manévrování při obcházení překážek.

Vzdálené senzorické obvody může řídit další mikroprocesor, který je užitečný pro zpracování dalších úloh, např. RP6 CONTROL M32 poskytuje další mikroprocesor ATmega32.

Rozšiřovací systém samozřejmě umožňuje připojení několika rozšiřujících modulů (viz obrázek), které používají minimální počet signálových vodičů a přitom poskytuje dostatečně velkou komunikační rychlost.

2.5.1. Sběrnice I^2C

Tyto požadavky splňuje sběrnice I^2C. Název tohoto standardu vnitřní sběrnice integrovaných obvodů je odvozen z I kvadrát C. Někdy se místo "I^2C" píše "I2C", protože symbol druhé mocniny "^{2}" není možné vložit do názvu proměnných a podobně. Sběrnice potřebuje pouze dva signálové vodiče a může se na ni připojit 127 dílčích zařízení komunikujících rychlostí 400 kbit/s.

Velmi populární sběrnici I^2C, navržená firmou Philips Semiconductors v průběhu osmdesátých a devadesátých let minulého století, je aplikována ve velkém počtu elektronických přístrojů, například video rekordérech, televizních přijímačích, ale také v průmyslových systémech. Řada moderních PC a notebooků používá variantu této sběrnice, která se nazývá SMBus, pro regulaci ventilace a teploty vnitřních zařízení. Sběrnicový systém I^2C používá také velký počet robotů. Z tohoto důvodu je sběrnicí I^2C vybavena řada senzorických modulů jako jsou ultrazvukové snímače, elektronické kompasy, teplotní čidla a podobná zařízení dostupná na trhu.

Sběrnice I^2C má master/slave orientaci. Jeden nebo více zařízení typu master řídí komunikaci s až 127 zařízeními typu slave. I když tato sběrnice dokáže zpracovat multi masterovou komunikaci, budeme popisovat sběrnicovou komunikaci s jediným zařízením typu master. Topologie multi-master je jen složitější variantou.

Dvě nezbytné datové linky se nazývají SDA a SCL. SDA se může číst jako "sériová data" a SCL se nazývá "sériové hodiny" – které již vysvětlují používání datového a hodinového signálového vodiče. SDA se používá jako obousměrný signál a proto jsou schopna přenášet data zařízení typu master i slave. SCL je zcela ovládán zařízením typu master.

Datové bity se vždy přenášejí synchronně s hodinovým signálem odvozeným v zařízení typu master. Úroveň signálu SDA se může měnit pouze, pokud je signál SCL v low (s výjimkou podmínky START a STOP, viz dále). Přenosová rychlost se může, kdykoliv během přenosu dat, měnit mezi 0 a 400 kbit/s.

Předchozí obrázky ukazují obvyklé přenosové protokoly. Na prvním je přenos z masteru do zařízení slave. Bílá políčka odkazují na přenos dat z master do slave a tmavá políčka představují odezvu od zařízení typu slave.

Každý přenos začíná inicializační podmínkou START a musí být ukončen podmínkou STOP. Podmínka START se vytvoří pokaždé, když se při vysoké úrovni SCL přitáhne linka SDA z vysoké do nízké úrovně. Opačná podoba signálových úrovní se aplikuje při podmínce STOP: když se při vysoké úrovni SCL vytáhne linka SDA z nízké do vysoké úrovně dostaneme podmínku STOP.

Bezprostředně po podmínce START vyšleme 7 bitů dlouhou slave adresu, která adresuje zařízení, následovanou bitem, který definuje, zda se budou zapisovat nebo číst data. Zařízení typu slave odpoví vysláním ACK ("Acknowledge = potvrzení"). Následovat může libovolný počet datových byte a každý jednotlivě přijatý byte bude potvrzen od slave (pomocí signálu ACK). Komunikace se ukončí podmínkou STOP.

Tento popis je pouze velmi stručné vysvětlení sběrnice I^2C. Hloubaví čtenáři si mohou vyhledat další informace ve specifikaci sběrnice I^2C od firmy Philips. Mnoho informací obsahuje také dokumentace mikroprocesoru ATmega32.

Ukázkové programy názorně ukazují, jak se používá hardware sběrnice. Knihovna RP6 již nabízí funkce pro ovládání sběrnice I^2C. Nebudeme zabíhat do detailů protokolu, ale je užitečné porozumět základní funkci komunikace po sběrnici.

2.5.2. Rozšiřující konektory

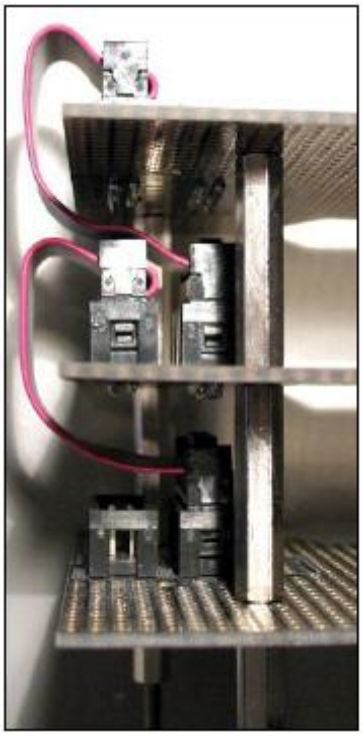
Hlavní deska poskytuje čtyři rozšiřující konektory. Dva jsou označeny “XBUS1” respektive “XBUS2”. “XBUS” je zkratka “eXpansion BUS = rozšiřující sběrnice”. “XBUS1” a “XBUS2” jsou kompletně propojené a na hlavní desce jsou uspořádány symetricky. Z tohoto důvodu budete moci umístit rozšiřující moduly na přední i zadní stranu robotu. Každý rozšiřující modul poskytuje na jedné straně modulu dva konektory XBUS. K vzájemnému propojení modulů a hlavní desky slouží 14 žilový plochý kabel. K propojení nabízí každý rozšiřující modul dva shodné propojovací konektory. Vnější konektor se použije k propojení směrem dolů, kdežto vnitřní konektor slouží k propojení směrem nahoru. Tento způsob umožňuje (teoreticky) skládat na sebe řadu modulů (viz obrázek, který ukazuje tři rozšiřující moduly s jednotlivými obvody sestavenými na univerzální desce RP6).

Konektory XBUS poskytují napájecí napětí, dříve popsanou sběrnici I^2C, reset hlavního mikroprocesoru a přerušovací signály.

Napájecí zdroj tvoří dvě napětí: nejprve je to stabilizovaných 5 V ze stabilizátoru, ale také napětí přímo z baterie. Napětí se bude během zatížení měnit – obvykle od 5,5 V (vybité baterie) až do přibližně 8,5 V (nové nabité baterie – tato hodnota se u jednotlivých výrobců liší). Hodnota napětí se však může překročit tyto limity podle zátěže, typu a stavu nabití baterie.

Signál master reset je důležitý pro resetování všech mikroprocesorových obvodů, když se stiskne tlačítko Start/Stop nebo při programování. Programování pomocí bootloaderu v mikroprocesoru spustí uživatelský program low impulsem (high-low-high) na lince SDA. Tímto způsobem se spustí všechny programy na mikroprocesorech (AVR) současně po stisknutí a uvolnění tlačítka Start/Stop nebo start programu bootloaderem … (bootloader negeneruje pouze low impuls pro start, ale také úplné všeobecné volání na sběrnici I^2C s datovým byte = 0).

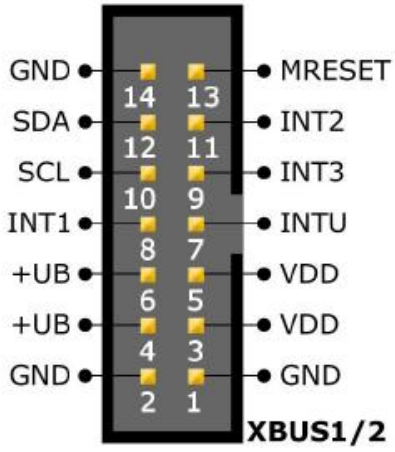
Několik modulů může využívat linky přerušení pro signalizaci příchodu nových dat nebo dokončení práce a očekávání nových povelů od hlavního procesoru. Tyto linky nejsou určeny pro vynucené opakované dotazování některých speciálních rozšiřovacích modulů na nová data. Tato metoda je samozřejmě možná, ale alternativní návrh s dalšími linkami přerušení budou obvykle omezovat provoz sběrnice a zatěžovat CPU. Přestože je počet linek přerušení omezený na 3 signály a jednu volnou linku vyhrazenou pro uživatelské signály, můžeme přiřadit jednu linku několika modulům (například všem ultrazvukovým snímačům) a oslovovat všechny moduly na základě jediné signalizace přerušení.



Další dva rozšiřující konektory na hlavní desce označené “USRBUS1” a “USRBUS2” nejsou vzájemně propojené. Všechny linky jsou přivedeny na pájecí plošky všech rozšiřujících modulů a na tyto plošky můžete aplikovat svoje vlastní signály.

“USRBUS” je zkratka “uživatelské-sběrnice”. Tento 14 vývodový rozšiřující konektor můžete použít pro cokoliv chcete – pro vlastní sběrnicový systém, další napájecí vodiče (ale musíte být opatrní, protože spoje dovolují maximální proud pouze 500 mA) nebo pro cokoliv jiného. Uvedeme příklad: jste schopni propojit dva rozšiřující moduly bez možnosti připojení k dalšímu modulu. To může být užitečné pro řadu složitých obvodů nebo snímačů, které se namohou umístit na jediný rozšiřující modul. Tato metoda bude čistější pro vlastní zapojení.

Samozřejmě nemůžete přidávat libovolný počet rozšiřovacích modulů – pokud nechcete přetížit vozidlo, můžete na přední nebo zadní stranu robotu navršit 6 modulů. Příliš velký počet modulů způsobí také problémy s přetěžování bateriového zdroje. Běžným pravidlem je připojení maximálně 8 modulů na RP6: 4 na přední stranu a 4 na zadní stranu.



Obrázek ukazuje zapojení obou rozšiřujících konektorů. Na hlavní desce je špička 1 vždy umístěna blízko nápisu XBUS1 respektive XBUS2. Alternativně je špička označena “1” na značce umístění konektoru.

+UB je napětí baterie, VDD je rozvod +5V, GND označuje “minus” neboli “zem” (GND = uzemnění), MRESET označuje signál Master Reset, INTx jsou linky přerušení, SCL jsou hodiny a SDA datová linka sběrnice I^2C.

Jediné, co musíte udělat, je zapájení konektoru USRBUS.

<u>Důležité upozornění</u>: nepřetěžujte napájecí vodiče VDD a +UB! Tyto vodiče mohou dodávat <u>maximální proud 1 A</u> (aplikovaný na obě špičky DOHROMADY. To znamená spojení špiček 4+6 (+UB) a 3+5 (VDD) na konektorech!

3. Nastavení hardware a software

Dříve než začnete nastavovat robot RP6 nebo příslušenství, přečtěte si pozorně následující bezpečnostní pokyny. Zvláště to platí, pokud budou RP6 později používat děti!

Tuto kapitolu čtěte obzvlášť pozorně!

3.1. Bezpečnostní pokyny

Díky otevřené architektuře RP6, existuje na konstrukci několik ostrých hran. Proto by robot neměly používat děti mladší 8 let! Hlídejte, prosím děti, když se v místnosti pohybuje RP6 a informujte děti o možném nebezpečí!

Neprovozujte robot v místech, kde se volně pohybují zvířata, například křečci, protože by je mohl RP6 poranit. Naopak velká zvířata jako psi a kočky mohou poškodit robot…

Pásový systém pohonu má nějaké nebezpečné části mezi pásy a koly, **kam může pás vtáhnout prsty.** Tyto oblasti jsou z velké části zakryté koly, přesto však dávejte pozor. Hlavně nestrkejte prsty mezi točící se kolo a pásy. Motory jsou skutečně výkonné a mohou vás snadno zranit. Prsty nestrkejte ani mezi pásy a desku plošných spojů!

POZOR: v případě, že používáte standardní software, mohou motory automaticky zvyšovat výkon! Podle způsobu naprogramování, mohou motory začít fungovat kdykoliv a nečekaně reagovat pohybem!

Robot nikdy neprovozujte bez dozoru!

3.1.1. Elektrostatické výboje a zkraty

Povrch hlavní desky plošných spojů, rozhraní USB a všech rozšiřujících modulů není nijak chráněn a odhaluje velké množství nechráněných součástek a vodivých cest. Nezpůsobte, prosím, zkrat tím, že na povrch robotu položíte kovové předměty nebo nástroje!

Napájecí napětí se mění ve velkém rozsahu, je však pro člověka bezpečné. Řadu součástek může poškodit elektrostatický výboj (ESD) a proto se jich nedotýkejte, pokud to není nezbytné! Speciálně v kombinaci se syntetickými textiliemi a suchým vzduchem se může vytvořit elektrostatický výboj při pohybu člověka. Také robot může získat náboj pohybem po některém povrchu podlahy. Při dotyku na kovové části se může náboj vybít přes tělo a vytvořit malé jiskry. Při manipulaci s robotem mohou tyto výboje poškodit nebo zničit elektronické součástky. Poškození vlivem ESD se vyhnete tím, že před manipulací s elektronickými obvody vybijete náboj z těla dotykem na velké uzemněné předměty (například kovová skříň PC, vodovodní potrubí nebo ústřední topení). Dotyk s uzemněným předmětem vybije elektrostatický náboj z těla. Neřízené vybití robotu při dotyku s uzemněnou překážkou nepoškodí robot, ale může porušit program nebo způsobit neočekávané chování.

Všechny elektrické linky vedoucí do systému musí být zapojeny, před přivedením napájecího napětí.

Neočekávané zapojení nebo odpojení konektorů, kabelů nebo modulu do spuštěného robotu může poškodit nebo zničit součástky elektronického systému a další díly.

3.1.2. Prostředí robotu

Neprovozujte robot na horní desce stolu nebo plochách s velkým převýšením, které mohou způsobit pád robotu na zem. Seznamte se, prosím, s možnostmi šplhání pásového vozidla! Robot může snadno přejet přes malé překážky a odtlačit lehké předměty. Z provozní oblasti robota odstraňte všechny předměty, které obsahují tekutiny tj. pohárky, sklenice a vázy.

Šasi robotu bude chránit mechanické díly před řadou účinků okolního prostředí, ale není vodotěsné a prachotěsné. Elektronika není chráněna vůbec. Robot byste měli provozovat pouze v čistém a suchém domácím prostředí. Nečistota, malé mechanické drobky a vlhkost mohou poškodit nebo zničit mechanické a elektronické části robotu. Provozní teplota je omezena na rozsah od 0°C do 40°C.

Provoz vnitřních stejnosměrných motorů generuje drobné jiskření. Robot se nesmí používat v prostředí s nebezpečím požáru nebo výbuchu (tekuté, plynné nebo prašné).

Pokud se robot nepoužívá delší dobu, neměl by se skladovat na místě s vyšší vlhkostí. Také, prosím, vyjměte baterie, aby nedošlo k poškození vytékajícím elektrolytem.

3.1.3. Napájecí napětí

Robot byl konstruován na napájení ze zdroje s napětím 7,2 V, který tvoří 6 nabíjecích NiMH článků. Maximální napětí zdroje je 10 V a nesmí být nikdy překročeno. Používejte pouze nabíjecí články s platnou bezpečnostní certifikací pro nabíjení.

Jako náhradu můžete robot provozovat se šesti kvalitními alkalickými bateriemi. Normální baterie se však velmi rychle vybijí, důsledkem je dražší provoz a zatěžování životního prostředí. Pokud je to možné, používejte vždy jen nabíjecí články. Nabíjecí články dodávají větší maximální proud a mohou se snadno nabíjet uvnitř robotu!

Dodržujte, prosím, bezpečnostní a provozní podmínky pro baterie uvedené v dodatku!

Modifikace robotu by měli provádět pouze zkušení uživatelé, kteří úplně ovládají problematiku. Nevhodná modifikace může poškodit robot nebo zranit obsluhu (například přehřívání součástek může způsobit požár bytu…).

3.2. Nastavení software

Následuje nastavení software. Pro všechny následující kapitoly je nezbytná správná instalace software.

Při instalaci musíte mít přístupová práva administrátora, přihlaste se proto jako administrátor počítačového systému.

Doporučujeme nejprve přečíst celou kapitolu a pak procházet jednotlivé pokyny krok za krokem.

Musíme předpokládat, že máte základní znalosti pro práci s počítači, které používají operační systémy Windows nebo Linux a standardní softwarové balíčky jako je souborový manažer, prohlížeč webových stránek, komprimační programy (WinZip, WinRAR, unzip atd.) a adekvátní Linux-Shell! Pokud neovládáte práci s počítačem, měli byste se před používáním RP6, seznámit se základními znalostmi tohoto oboru. Tato příručka nemůže poskytnout úvodní kurz používání osobního počítače a na toto problematiku se text příručky nezaměřuje. Tato příručka bude popisovat robotický systém RP6, programování RP6 a specializovaný systémový software.

3.2.1. CD-ROM RP6

Pravděpodobně jste již CD-ROM RP6 vložili do mechaniky CD-ROM ve vašem PC – pokud ne, vložte CD nyní. V systému by se měla spustit akce automatického spuštění a v okně prohledávače by se měla objevit nabídka. Pokud ne, můžete ve webovém prohlížeči např. Firefox otevřít soubor "start.htm" umístěný v hlavním adresáři CD. Pokud PC nedisponuje moderním prohlížečem, můžete najít instalační balíček Firefox ve složce CD:

```
<CD-ROM-Drive>:\Software\Firefox
```

Měli byste používat poslední verzi Firefox 1.x nebo Internet Explorer 6.

Zvolte váš národní jazyk a CD menu vám nabídne řadu užitečných informací a software. Mimo tuto příručku (kterou můžete stáhnout z našich domácích stránek) si můžete prohlédnout například katalogových součástek použitých v robotu. Nabídka označená “software” poskytuje přístup ke všem softwarovým nástrojům, ovladač USB a ukázkové programy včetně zdrojových textů pro RP6.

Podle bezpečnostního nastavení vašeho webového prohlížeče můžete spustit instalační balíčky přímo z CD. Pokud to nastavení prohlížeče neumožňuje, nepokoušejte se instalaci dokončit. Zkopírujte soubory na pevný disk a instalaci proveďte z pevného disku PC. Podrobnosti tohoto postupu najdete na softwarové stránce CD nabídky. Alternativně můžete prozkoumat kořenový adresář CD souborovým manažerem a spustit instalaci přímo z CD. Názvy adresářů můžete vybrat podle názvu příslušného softwarového balíčku a operačního systému.

3.2.2. WinAVR pro Windows

Nejdříve nainstalujete WinAVR. WinAVR je však – jak již napovídá název – dostupný jen pro Windows.

Uživatelé Linuxu mohou tuto část přeskočit.

WinAVR (vyslovuje se “whenever”) je balíček užitečných a nezbytných nástrojů pro vývoj software pro mikroprocesory AVR v programovacím jazyce C. Více informací o samotném GCC pro cílovou platformu AVR (který se nazývá "AVR-GCC") bude následovat později. Prostředí WinAVR také poskytuje komfortní editor zdrojových textů, nazvaný "Programmers Notepad 2", který je vhodný i pro vývoj software pro RP6. WinAVR má vnitřní projektové uspořádání a programový balíček je volně dostupný na internetu. Nové verze a další informace můžete najít na oficiálních webových stránkách projektu:

`http://winavr.sourceforge.net/`

Teprve nedávno zahájila firma ATMEL oficiální podporu projektu a AVRGCC se nyní může integrovat do jejich integrovaného vývojového prostředí AVRStudio. Editor Programmers Notepad 2 je pro vaše vlastní projekty mnohem výhodnější, proto zde nebudeme popisovat prostředí AVRStudio. Přesto můžete při vývoji programů pro RP6 používat i AVRStudio. Instalaci WinAVR můžete najít na CD:

`<CD-ROM-Drive>:\Software\AVR-GCC\Windows\WinAVR\`

Instalace WinAVR je velmi jednoduchá a samo vysvětlující – obvykle nemusíte měnit nastavení – vždy jen kliknete pokračovat. Pokud budete mít problémy se spuštění nejnovější verze WinAVR, jsou na CD k dispozici také starší verze tohoto programu. Pokud se objeví nějaké problémy se standardní verzí programu, je zde také složka pro Win x64.

3.2.3. AVR-GCC, avr-libc a avr-binutils pro Linux

Uživatelé Windows mohou tuto část přeskočit.

Instalace avr-gcc v prostředí Linux může být o něco komplikovanější. Několik distribucí již poskytuje potřebnou podporu, ale programové balíčky často obsahují nepodporované verze bez nezbytných složek.

Pravděpodobně budete muset kompilovat a instalovat nejnovější verze.

Nemůžeme zmiňovat detaily všech známých rozdílných variant distribucí operačního systému Linux jako jsou SuSE, Ubuntu, RedHat/Fedora, Debian, Gentoo, Slackware, Mandriva atd. a jejich odlišnosti. Seznámíme vás pouze s obecnou instalací.

Platí to také pro všechny ostatní Linux témata uvedená v této kapitole!

Pro specifické nastavení systému nemusí být následující seznámení automaticky dostačující. Často budete muset hledat pomoc pomocí hesla "<LinuxDistribution> avr gcc" a modifikací fráze v tomto řetězci. Je to také dobrý postup pro všechny další potíže, které se mohou objevit v operačním systému Linux. Pokud se při instalaci avr-gcc vyskytnou nějaké potíže, můžete se pokusit najít řešení návštěvou našeho fóra nebo řady dalších diskusí věnovaných systému Linux.

Nejdříve musíte odinstalovat předchozí verze avr-gcc – zpravidla již nepodporované – totéž platí pro nástroje avr-binutils a avr-libc. Odinstalování spustíte z nástroje správy souborů, vyhledáním “avr” a odstraněním programového balíčku ze systému. Pokud nástroj najde příslušné objekty, spustí “avr-gcc”.

Snadno můžete zkontrolovat, zda je avr-gcc instalován či ne. Pokud existuje, můžete zjistit umístění programu pomocí následujícího příkazového řádku:

```
> which avr-gcc
```

Pokud systém reaguje názvem adresáře, bude pravděpodobně ve vašem systému již existovat nějaká verze avr-gcc. V takovém případě zkontrolujte verzi:

```
> avr-gcc --version
```

Pokud je číslo verze nižší než 3.4.6, pak ji definitivně odinstalujte. Pokud je verze mezi 3.4.6 a 4.1.0 můžete ji vyzkoušet kompilací programů (viz následující kapitola). Pokud kompilace selže, odinstalujte starší verze a instalujte avr-gcc verzi z CD. Následující kapitola se opírá o nejnovější verzi 4.1.1 (uvolněnou v březnu 2007), která obshuje nějaké důležité záplaty aby se shodovala s WinAVR.

Pozor: před zahájením kompilace a instalací zkontrolujte možnosti standardního vývojového balíku pro Linux tj. GCC, make, binutils, libc atd. Použijte manažer distribučního balíku. Každá distribuce systému Linux by měla na instalačním CD poskytovat potřebnou podporu. Alternativně můžete poslední verzi získat přes internet.

Dávejte pozor, aby byl nainstalován program “texinfo”. Pokud program chybí, musíte ho vložit před instalací – jinak instalační proces selže.

Pokud dokončíte přípravy, můžete zahájit aktuální instalaci.

Můžete si vybrat ze dvou možností: buď ruční kompilaci a instalaci všech balíčků, nebo můžete použít jednoduchý skript automatické instalace.

Doporučujeme, aby jste se nejprve pokusili spustit skrip a ruční istalaci použít pouze když se objeví problémy.

Pozor: Zkontrolujte, prosím, zda je na disku dostatek místa. Budete potřebovat více než 400 MB volného prostoru. Více než 300 MB těchto dat je potřeba pouze dočasně pro kompilaci a můžete je později odstranit.

Řada instalací požaduje přesné umístění adresářů a doporučujeme přihlášení kořenového adresáře “su” nebo alternativně spustit kritické úlohy s parametrem “sudo” (jak je to obvyklé u distribuce Ubuntu) nebo příslušným příkazem. Instalační skript mkdir v adresářích /usr/local/ a make file vytvoří při instalaci správnou strukturu adresářů.

Dávejte pozor na SPRÁVNÉ psaní následujících příkazů. Všechny symboly jsou důležité a některé příkazy se mohou zablokovat –tyto řádky přepisujte správně, aby neobsahovaly chyby (samozřejmě můžete nahradit řetězec <CD-ROM-drive> názvem vaší mechaniky CD-ROM).

Důležité instalační soubory pro avr-gcc, avr-libc a binutils můžete najít v adresáři:

```
<CD-ROM-Drive>:\Software\avr-gcc\Linux
```

Začněte kopírováním všech instalačních souborů do adresáře na pevném disku – to platí pro obě metody instalace. V tomto případě použijeme domácí adresář (standardní zkratka domácího adresáře je znak vlnovka: “~”).

```
> mkdir ~/RP6
> cd <CD-ROM-Laufwerk>/Software/avr-gcc/Linux
> cp * ~/RP6
```

Po dokončení instalace se mohou tyto soubory odstranit, aby se ušetřilo místo na disku.

3.2.3.1. Skript automatické instalace

Spustitelný skript, který používá chmod, můžete vytvořit následovně:

```
> cd ~/RP6
> chmod -x avrgcc_build_and_install.sh
> ./avrgcc_build_and_install.sh
```

Můžete reagovat odpovědí “y”, když budete chtít instalovat s touto konfigurací.

POZOR: Zpracování kompilace a instalace bude někdy záviset na výkonu počítačového systému (tj. asi 15 minut při dvojitém jádru s taktem 2 GHz, notebook – pomalejší systémy mohou potřebovat mnohem větší dobu).

Skript také vytvoří několik adresářů – ty jsou označeny v adresáři .diff-files.

Při dokončení zpracování můžete vidět následující zprávy:

```
(./avrgcc_build_and_install.sh)
(./avrgcc_build_and_install.sh) dokončení instalace nástrojů avr GNU
(./avrgcc_build_and_install.sh) přidání /usr/local/avr/bin do adresáře
avr GNU nástrojů
(./avrgcc_build_and_install.sh)  můžete  chtít  spustit  následující
úsporu místa na disku:
(./avrgcc_build_and_install.sh)
(./avrgcc_build_and_install.sh)    rm    -rf    /usr/local/avr/source
/usr/local/avr/build
```

Pak můžete zpracovat dopuručený příkaz:

```
rm -rf /usr/local/avr/source /usr/local/avr/build
```

Tento příkaz zruší dočasné soubory, které už nebudete nikdy potřebovat.

Nyní můžete provést následující krok a nastavit adresář pro proměnné prostředí avr-tools.

Pokud skript skončí nějakou chybovou zprávou, přečtěte si chybové zprávy pozorně (a posouvejte po obrazovce) – mohou chybět nějaké programy, které se nenainstalovaly v předchozím kroku (tj. dříve zmíněný program texinfo).

Před zpracováním chybové zprávy, vám můžeme poradit zrušení vygenerovaných souborů ve standardním instalačním adresáři “/usr/local/avr”. Dále doporučujeme zrušit celý adresář.

Pokud nechtěně uděláte chybu, uložte všechny výstupy příkazových řádků do souboru a pošlete popis příslušných chyb a testový soubor podpůrnému týmu. Pošlete, prosím, všechny dostupné informace. Jedině tak můžete získat vyčerpávající pomoc.

3.2.3.2. Ruční postup instalace

Pokud preferujete ruční instalaci nebo selže skript automatické instalace, můžete postupovat podle následujících kroků.

Popis byl odvozen z následujícího materiálu:

http://www.nongnu.org/avr-libc/user-manual/install_tools.html

V dokumentaci AVR Libc na CD můžete najít také PDF dokument:

<CD-ROM-Drive>:\Software\Documentation\avr-libc-user-manual-1.4.5.pdf

Začněte PDF souborem na straně 240 (respektive 232 podle systému číslování dokumentu).

Tento popis je poze souhrn dokumentu, ale také instalujeme několik důležitých adresářů – pokud tyto adresáře nevytvoříte, nemusí některé komponenty správně fungovat (například velmi užiteřčné binární konstanty).

Nejprve musíme vytvořit adresář, ve kterém probíhá instalace všech nástrojů. Adresář by se měl nazývat: /usr/local/avr.

Následující příkazy ukončí ENTER jako KOŘENOVÝ ADRESÁŘ:

```
> mkdir /usr/local/avr
> mkdir /usr/local/avr/bin
```

Pokud tento adresář nepotřebujete, jednoduše definujte proměnnou s názvem $PREFIX tohoto adresáře:

```
> PREFIX=/usr/local/avr
> export PREFIX
```

Nyní do adresáře konečně přidáme proměnné:

```
> PATH=$PATH:$PREFIX/bin
> export PATH
```

Binutils pro AVR

Zpracujeme extrahované zdrojové kódy Binutils a vytvoříme několik adresářů. Všechny soubory můžete kopírovat do základního adresáře ~/RP6:

```
> cd ~/RP6
> bunzip2 -c binutils-2.17.tar.bz2 | tar xf > cd binutils-2.17
> patch -p0 < ../binutils-patch-aa.diff
> patch -p0 < ../binutils-patch-atmega256x.diff
> patch -p0 < ../binutils-patch-coff-avr.diff
> patch -p0 < ../binutils-patch-newdevices.diff
> patch -p0 < ../binutils-patch-avr-size.diff
> mkdir obj-avr
> cd obj-avr
```

Nyní zpracujte konfigurační skript:

```
> ../configure --prefix=$PREFIX --target=avr --disable-nls
```

Tento skript analyzuje, zda je dostupný systém a generuje potřebné make file. Na konci skriptu se vždy provede kompilace a instalace:

```
> make
> make install
```

Podle výkonu PC to zabere několik minut – to platí také pro následující kroky – zvlášť pro GCC!

GCC for AVR

Používá podobný postup jako Binutils, GCC musí být kompilována a instalováno do správných adresářů:

```
> cd ~/RP6
> bunzip2 -c gcc-4.1.1.tar.bz2 | tar xf > cd gcc-4.1.1
> patch -p0 < ../gcc-patch-0b-constants.diff
> patch -p0 < ../gcc-patch-attribute_alias.diff
> patch -p0 < ../gcc-patch-bug25672.diff
> patch -p0 < ../gcc-patch-dwarf.diff
> patch -p0 < ../gcc-patch-libiberty-Makefile.in.diff
> patch -p0 < ../gcc-patch-newdevices.diff
> patch -p0 < ../gcc-patch-zz-atmega256x.diff
> mkdir obj-avr
> cd obj-avr
> ../configure --prefix=$PREFIX --target=avr --enable-languages=c,c++
\
  --disable-nls --disable-libssp –with-dwarf2
> make
> make install
```

Použitím “\” můžete stisknout Enter a pokračovat v psaní příkazového řádku – tento znak umožňuje rozdělit řádek a zapsat extrémně dlouhý příkazový řádek přehledně do několika řádků. Tento znak můžete samozřejmě vynechat a příkaz zapsat jako jediný velmi dlouhý řádek.

AVR Libc

Nakonec AVR libc:

```
> cd ~/RP6
> bunzip2 -c avr-libc-1.4.5.tar.bz2 | tar xf >
cd avr-libc-1.4.5
> ./configure --prefix=$PREFIX --build=`./config.guess` --host=avr
> make
> make install
```

Pozor: V –build=`./config.guess` musíte dát pozor na “zvýraznění” (<-- tenké škrtnutí nad písmeny. Nesmíte použít normální apostrof, který nebude fungovat.

3.2.3.3. Nastavení adresáře

Nyní dávejte pozor aby byl adresář /usr/local/avr/bin v adresářové proměnné! Jinak nebudete schopni spustit avr-gcc z terminálu a vytvořit makefiles. Adresář avr-gcc musíte přidat do souboru /etc/profile nebo /etc/environment nebo podobných souborů (tyto proměnné se liší podle distribuce). K existujícímu řetězci můžete přidat další adresář, oddělený znakem “:”. Řádek v souboru může obsahovat více adresářů:

```
PATH="/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/usr/local/avr/bin"
```

Nyní můžete odzkoušet funkčnost instalace vložením příkazu “avr-gcc -–version” na terminál, jak bylo uvedeno v předchozí části. Pokud dostanete správnou odezvu, byla instalace dokončena.

3.2.4. Java 6

RP6Loader (podrobnosti výše) byl navržen pro prostředí Java a může se používat ve Windows i Linux (teoreticky může dobře fungovat i pod dalšími operačními systémy jako je OS X, ale AREXX Engineering je bohužel nepodporuje). V případě RP6Loader musíte instalovat poslední verzi prostředí Java Runtime Environment (JRE). Možná je na počítači již nainstalované, ale nejde o nejnovější verzi 1.6 (= Java 6)! Pokud nemáte instalováno nejnovější JRE nebo JDK, instalujte SUN Microsystems JRE 1.6 z podpůrného CD nebo alternativně z webových stránek http://www.java.com nebo http://java.sun.com.

3.2.4.1. Windows

V prostředí Windows je JRE 1.6 umístěn v adresáři:

`<CD-ROM-Drive>:\Software\Java\JRE6\Windows\`

Ve Windows je instalace skutečně jednoduchá – pouze spustíte Setup a sledujete pokyny – hotovo! Následující část můžete přeskočit.

3.2.4.2. Linux

Instalace Java je v prostředí Linux většinou stejně snadná jako ve Windows, ale některé distribuce mohou vyžadovat ruční zpracování.

JRE6 můžete najít jako RPM (SuSE, RedHat atd.) a je ve tvaru samo-extraktujícího archivu ".bin" v adresáři:

`<CD-ROM-Drive>:\Software\Java\JRE6\`

Naše rada je vyhledat balíček Java pomocí zvláštního manažeru distribuce (vyhledávání pro "java", "sun", "jre" or "java6" ...) a tento balíček použít místo instalace dodávané na CD. Dávejte pozor, aby jste nainstalovali poslední verzi Java 6 (= JRE 1.6) nebo vyšší.

U Ubuntu nebo Debian nemusí někdy RPM Archiv fungovat – v takovém případě můžete použít souborový manažer distribuce. Další distribuce jako RedHat/Fedora, SuSE mohou používat RPM bez využití souborového manažeru.

Pokud není instalace úplná, můžete se ještě pokusit extrahovat soubor JRE ze samo rozbalovacího archivu (.bin) v adresáři pevného disku (tj. /usr/lib/Java6) a pak ručně nastavit cestu k JRE (příkazy PATH a JAVA_HOME atd.).

Postupujte podle pokynů instalace Sun, kterou můžete najít ve zmíněném adresáři na webových stránkách Java.

Vždy zkontrolujte funkčnost instalace příkazem "java -version". Měla by následovat odezva:

```
java version "1.6.0"
Java(TM) SE Runtime Environment (build 1.6.0-b105)
Java HotSpot(TM) Client VM (build 1.6.0-b105, mixed mode, sharing)
```

Pokud je odezva jiná, může být instalace bez podpory nebo může v systému ještě běžet další Java VM.

3.2.5. RP6Loader

RP6Loader jsme vytvořili pro zjednodušení nahrávání nových programů do RP6 a všech rozšiřujících modulů (pokud tyto moduly obsahují mikroprocesor s kompatibilním bootloaderem). Navíc jsme implementovali několik užitečných funkcí, například jednoduchý terminálový program pro sériovou komunikaci.

RP6Loader se nemusí instalovat – místo toho se program jednoduše zkopíruje do nového adresáře na disku. RP6Loader je umístěn v ZIP archivu na RP6 CD-ROM:

```
<CD-ROM-Drive>:\Software\RP6Loader\RP6Loader.zip
```

Rozbalte soubor někam na disk – tj. do nového adresáře C:\RP6\RP6Loader (nebo podobného). Tento adresář obsahuje spustitelný program RP6Loader.exe.

Ve skutečnosti je RP6Loader umístěn v Java Archive (JAR) RP6Loader_lib.jar. Alternativně můžete spustit RP6Loader z okna příkazového řádku.

Windows:

```
java -Djava.library.path=".\lib" -jar RP6Loader_lib.jar
```

Linux:

```
java -Djava.library.path="./lib" -jar RP6Loader_lib.jar
```

Dlouhá alternativa –D vyžaduje povolení JVM pro umístění všech nezbytných knihoven. Obvykle nebodeme tuto alternativu potřebovat a jen spustíte soubor .exe. Linux používá skript “RP6Loader.sh”, který potřebujete pro nastavení spouštění vydáním chmod -x ./RP6Loader.sh. To vám umožní spuštění “./RP6Loader.sh” z terminálu nebo pracovního prostředí.

Doporučujeme vytvořit odkaz RP6Loader na pracovní ploše nebo nabíce Start. Ve Windows se to udělá kliknutím pravého tlačítka myši na RP6Loader.exe a výběrem “Přenést do” --> “Desktop (vytvoření odkazu)”.

3.2.6. Knihovna RP6, knihovna RP6 CONTROL a ukázkové programy

Knihovna RP6 a příslušné ukázkové programy jsou umístěny v ZIP archivu na přiloženém CD:

```
<CD-ROM-Drive>:\Software\RP6Examples\RP6Examples.zip
```

Rozbalte tento archiv do adresáře na pevném disku, Doporučujeme použít adresář v datové části. Alternativně můžete použít adresář “My Documents” a vytvořit složku “RP6\Examples\” nebo použít domácí adresář Linux.

Ukázkové programy podrobně probereme v této příručce později.

Archiv obsahuje také příklady pro rozšiřující modul RP6 CONTROL M32 včetně příslušných knihovních souborů.

3.3. Připojení rozhraní USB – Windows

Uživatelé Linux mohou totu část přeskočit.

Existuje několik způsobů, jak instalovat ovladače USB rozhraní. Nejjednodušší způsob je instalace ovladače PŘED prvním připojení zařízení. CD nabízí různé instalační programy ovladače.

Pro 32 a 64 bitové Windows XP, Vista, Server 2003 a 2000:

```
<CD-ROM-Laufwerk>:\Software\USB_DRIVER\Win2k_XP_Vista\CDM_Setup.exe
```

Nastavení programu bohužel není tak pohodlné jako ve Win98SE/Me – v tomto případě budete muset ručně instalovat starší verzi ovladače až po připojení zařízení do PC (viz dále).

Spusťte klidně instalační program CDM – program zobrazí krátký informační dialog, který potvrzuje, že byl ovladač úspěšně nainstalován. To je vše.

Po instalaci můžete připojit USB rozhraní do PC, ALE NEPŘIPOJUJTE JE ZATÍM DO ROBOTU! Zapojte je pomocí USB kabelu do PC. Destičky se dotýkejte pouze ze stran nebo za USB konektor, respektive plastový kryt programovacího konektoru (viz bezpečnostní pokyny o statických výbojech). Dávejte pozor na nežádoucí dotek součástek na destičce, pájecích plošek nebo spojovacích dílů. To jsou obecná pravidla pro zacházení se všemi elektronickými přístroji bez krytu.

Již nainstalovaný ovladač se automaticky přiřadí k zařízení a není potřeba další akce. V operačním systému Windows XP/2k se objeví několik zpráv – poslední zpráva by měla vypadat takto: “Hardware byl úspěšně instalován a je připraven k používání”!

Pokud připojíte USB rozhraní před instalací ovladače (nebo když používáte Win98/Me) – nebuďte smutní. Windows se vás zeptá na ovladač, který můžete najít a rozbalit z dodaného CD. Windows obvykle zobrazí instalační dialog ovladače. Budete požádáni o zadání cesty k ovladači. V systému Windows 2k/XP musíte nejdříve vybrat “ruční instalaci”. Nevolte “vyhledání na webu” nebo podobné možnosti, protože ovladač je umístěn na CD ve dříve specifikovaném adresáři.

Jednoduše vyberte adresář s ovladačem pro vaši verzi Windows a možná dalšími soubory, které nejsou přímo požadovány systémem (všechny soubory jsou umístěny ve stejných adresářích, které budou popsány v následující části)…

Obvykle Windows XP nebo novější verze budou nyní pokračovat s upozorněním, ve kterém Microsoft varuje, že ovladač nemusí být autorizován nebo ověřen – jedná se o irelevantní výstrahu a můžete ji bez rizika potvrdit. V takovém případě se ovladač FTDI autorizuje a systém přestane zobrazovat upozornění.

Pro 32 a 64 bitové systémy Windows XP, Vista, Server 2003 a 2000:

`<CD-ROM-Laufwerk>:\Software\USB_DRIVER\Win2k_XP_Vista\FTDI_CDM2.02.04\`

Pro starší Windows 98SE/Me:

`<CD-ROM-Laufwerk>:\Software\USB_DRIVER\Win98SE_ME\FTDI_D2XX\`

Několik starších verzí Windows tj. Win98SE vyžadují po instalaci ovladače restart. **POZOR**: U Win98/Me můžete instalovat jednu ze dvou verzí ovladače buď D2XX nebo VCP (virtuální COM port). Pro starší systémy neexistuje ovladač s oběma funkcemi. Obvykle není dostupný virtuální COM port, protože RP6Loader pro standardní verzi Windows používá ovladač D2XX (ten může být podle potřeby změněn – pro pomoc můžete kontaktovat podpůrný tým).

3.3.1. Kontrola správné funkce připojeného zařízení

Při kontrole správného připojení zařízení ve Windows XP, Vista, 2003 a 2000 můžete použít buď RP6Loader nebo Windows správce zařízení:
Klikněte pravým tlačítkem na Tento počítač --> Vlastnosti --> Hardware --> Správce zařízení nebo alternativně: Start --> Nastavení --> Ovládací panely --> Možnosti a údržba --> Hardware --> Správce zařízení. Zkontrolujte zobrazenou větev “Připojení (COM a LPT)” pro "USB-Sériový Port (COMX)" – kde X představuje číslo portu nebo zkontrolujte “USB - Ovladač” pro “USB Sériový převodník” a dávejte pozor, aby nebyl do počítače zapojen standardní USB sériový adaptér.

3.3.2. Odinstalování ovladače

Pokud budete někdy potřebovat odstranit ovladač (ne, nedělejte to právě teď – je to jen pro informaci): když proběhla instalace pomocí instalačního programu CDM můžete odinstalování provést nástrojem Start --> Nastavení --> Ovládací panely --> Software. Seznam by měl obsahovat položku “Ovladač FTDI USB sériového převodníku”. Pouze vyberete tuto položku a kliknete na odstranit/odinstalovat.

Pokud jste ovladač instalovali ručně, můžete spustit program "FTUNIN.exe" v adresáři USB ovladače na CD. Pozor: Tento ovladač mohou používat některé adaptéry USB-->RS232, které používají čipovou sadu FTDI.

3.4. Připojení rozhraní USB – Linux

Uživatelé Windows mohou tuto část přeskočit.

Linux Kernel 2.4.20 nebo vyšší již zahrnují ovladač rozhraní USB s obvodem FT232R (nejnovější typ kompatibilní s předchůdcem FT232BM). Zařízení se rozpozná automaticky a nemusíte dělat nic dalšího. Jen pokud se objeví potíže, můžete získat ovladač Linux (a podporu včetně nových verzí ovladačů a dokumentace) přímo od FTDI:

http://www.ftdichip.com/

Po připojení zařízení do počítače se systémem Linux můžete zkontrolovat správné příjmutí USB sériového portu, vložením příkazu:

cat /proc/tty/driver/usbserial

To je vše.

Jen pro informaci: Windows verze RP6Loaderu používá ovladače D2XX a v seznamu portů bude zobrazovat úplný název USB (tj. “USB0 | RP6 USB Interface | serialNumber”). Naproti tomu Linux verze programu bude zobrazovat název virtuálního COM portu /dev/ttyUSB0, /dev/ttyUSB1 nebo podobně. Mohou se zobrazit i další standardní označení COM portu (“dev/ttyS0” atd.).V takovém případě budete muset zkusit, který port je správný. Linux bohužel neumožňuje snadnou instalaci ovladače pro obě funkce a z tohoto důvodu preferujeme používání ovladače virtuálního COM portu, který je obvykle obsažen ve standardním jádru Linux. Instalace ovladače D2XX může vyžadovat nějaké ruční zásahy.

3.5. Dokončení instalace software

To bylo vše, co se musí udělat pro nastavení software a USB rozhraní.

Na závěr můžete zkopírovat většinu důležitých souborů z CD na pevný disk (speciálně celý adresář “Dokumentace” a “Příklady”, pokud jste to již neprovedli). Díky tomu nemusíte pokaždé hledat konkrétní soubor na CD. Adresáře na CD jsou pojmenovány podle obsahu softwarových balíčků, takže se v nich velice dobře orientujete.

Pokud CD ztratíte, můžete všechny důležité soubory stáhnout z našich webových stránek. Najdete zde také nejnovější verze dat, které mohou obsahovat důležité odstranění chyb nebo nové vlastnosti.

3.6. Vložení baterií

Je na čase věnovat se vlastnímu robotu. V první řadě musí mít robot 6 baterií.

Doporučujme používat velmi kvalitní NiMH baterie (od známých výrobců tj. Sanyo, Panasonic, atd.) se specifikovanou reálnou provozní kapacitou větší než 2000 mAh (optimální kapacita baterií je 2500 mAh). Nepoužívejte, prosím, běžné alkalické baterie, jejich provoz je velmi drahý a také způsobují nežádoucí znečistění životního prostředí.

Doporučujeme používat **předem nabité baterie.** Vždy dávejte pozor, aby byly baterie nabité na stejnou úroveň (všechny baterie jsou buď nabité, nebo vybité) a používejte relativně nové baterie. Baterie se mohou opotřebovat dlouhým skladováním, počtem nabíjecích cyklů, způsobem vybíjení a teplotou. Nejlepší je používat nové baterie, protože staré baterie vlivem dlouhodobého skladování ztrácí původní vlastnosti. Velmi důležité je také používat velmi shodné články. U jednotlivých článků se může lišit kapacita, stáří, úroveň nabití…

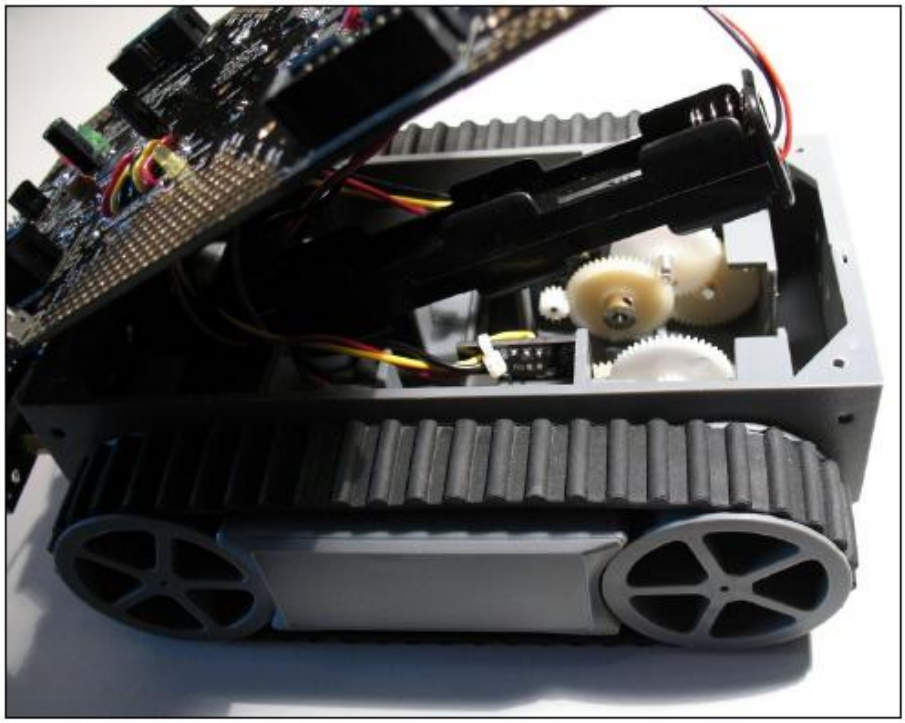
Pokud upřednostňujete externí nabíječku (velmi doporučujeme, ale není obsažena v balení), můžete baterie instalovat pouze JEDNOU. Doporučujeme používat nabíječku řízenou mikroprocesorem, která zajistí optimální nabití baterií. Pro svou vlastní bezpečnost používejte pouze certifikované a odzkoušené nabíjecí zařízení!

Nepoužívejte externí nabíjecí zařízení se zvláštním adaptérem konektoru, které potřebuje dlouhodobé nabíjení. Raději vyjměte baterie ze systému, nabijte baterie a znovu je vložte do systému.

Vkládání baterií:

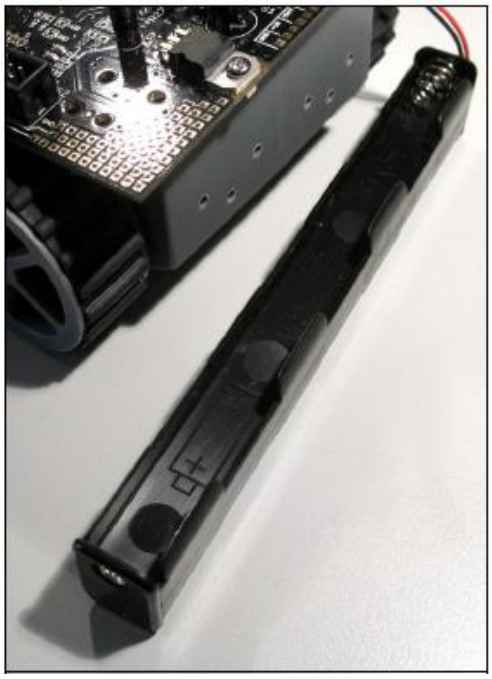
Nejprve musíte uvolnit čtyři šroubky, které upevňují hlavní desku.

Nyní opatrně zvedněte hlavní desku na boční stranu robotu (viz obrázek).



NEMUSÍTE rozpojovat malý konektor se třemi kontakty na destičce nárazníků. Buďte velmi opatrní při manipulaci s hlavní deskou. Dotýkejte se pouze hrany desky plošných spojů nebo velkých plastových dílů, aby nedošlo k výboji statické elektřiny.

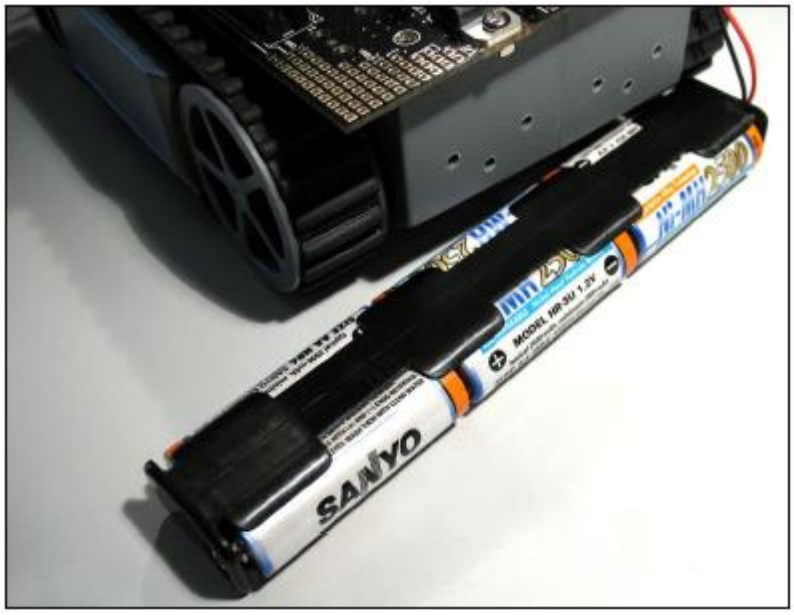
Hlavní deska elektroniky je spojena s motory, enkodéry a držákem baterie pomocí svazku zapájených kabelů. Přesuňte tyto svazky vodičů – opatrně stranou – podle jejich umístění.



Vyjměte držák baterie dozadu (viz obrázek).

Dávejte pozor, aby byl vypínač v poloze “OFF”. Páčka vypínače musí směřovat k nápisu “OFF” a velkému válcovému kondenzátoru na hlavní desce (viz obr.)

Před novým spuštěním robotu, zkontrolujte správnou orientaci baterií.

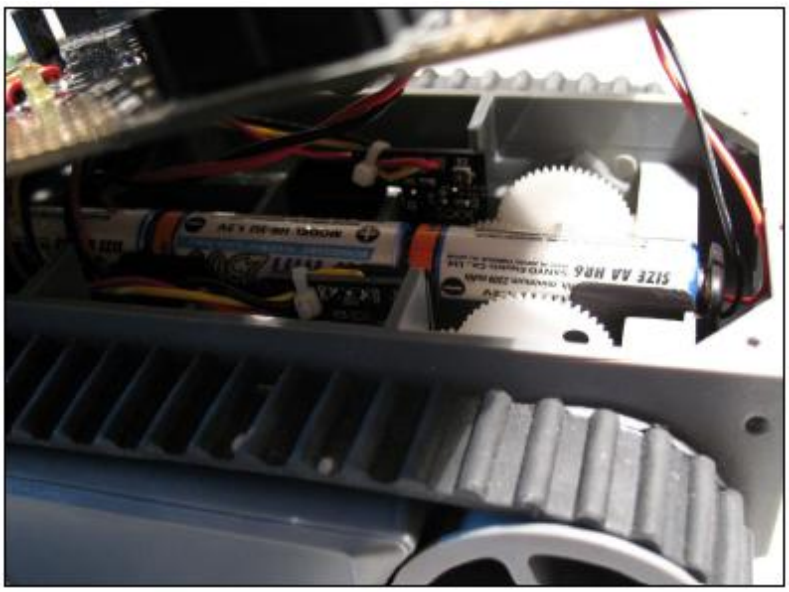


Nyní můžete vložit 6 NiMH baterií se SPRÁVNOU ORIENTACÍ / POLARITOU!

POZOR: Při vložení baterie s opačnou polaritou se přepálí tavná pojistka.

V nejhorším případě však můžeme poškodit součástky elektronických obvodů.

Nejlepší řešení bude, když hned vložíte baterie správným způsobem a vyhnete se všem možným problémům. V držáku baterie jsou navíc značky (kladný (+) kontakt je plochý a záporný (-) pól tvoří pružiny v držáku), které vám pomáhají s orientací.



Polaritu baterií kontrolujte raději třikrát, než rozhodnete, že je vše v pořádku!

Nyní můžete vložit držák baterií zpátky do šasi. Dávejte pozor na kabely. Pozor hlavně na to, aby nebyly blízko převodovek.

Když už máte otevřený robot, můžete nyní rychle zkontrolovat obě převodovky a enkodéry, zda nejsou poškozené tj, chybějící matičky, šrouby nebo jiné součástky. Velmi opatrně a pomalu otočte jednu otáčku zadními koly.

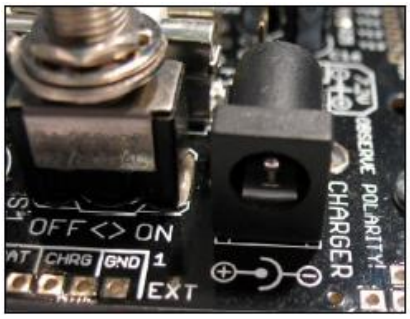
Vždy stačí pootočit půl otáčky dopředu a půl otáčky dozadu. Měli byste cítit jemný odpor, ale kola se musí otáčet volně. Kola převodovky se pohybují volně. Podívejte se také do dodatku A.

Hlavní desku nyní můžete vrátit na šasi. Pomocí dlouhého nástroje nebo prstů zasuňte kabely mezi hlavní desku a plastové přepážky tak, aby hlavní deska pěkně ležela na šasi. Před upevněním ještě zkontrolujte, zda některý kabel není sevřený k šasi nebo převodovce. Nyní můžeme připevnit desku k šasi dotažením čtyř šroubů – dotahujte je opatrně.

3.7. Nabíjení baterie

Pokud jste nenainstalovali předem nabité baterie, jak jsme doporučovali, měli byste nyní připojit externí nabíječku. Při nabíjení přepněte hlavní vypínač napájení do polohy “OFF”. Nabíjení baterie funguje pouze u vypnutého robotu. Hlavní vypínač připojuje baterie buď k elektronickým obvodům RP6 nebo ke konektoru nabíjení.

Zkontrolujte polaritu přívodu podle konektoru pro připojení nabíječky (označený “Charger”) umístěného na robotu hned vedle vypínače napájení.



Polaritu vidíte označenou na hlavní desce před konektorem (viz obrázek). **Záporná svorka je umístěna na VNĚJŠÍM KOVOVÉM KONTAKTU a kladná svorka na VNITŘNÍM VÝVODU.**

Doba nabíjení velmi závisí na typu použité nabíječky a baterie (mikroprocesorem řízená nabíječka například Voltcraft 1A/2A rychlá nabíječka, která využívá princip Delta Peak nebo Ansmann ACS110/410 budou k nabití potřebovat 3 až 4 hodiny, standardní nabíječky například AC48 potřebuje k nabíjení čas asi 14 hodin) – prostudujte pečlivě návod k používání nabíječky.

Během nabíjení nezapínejte hlavní vypínač robotu do polohy “ON”. Před zapnutím robotu, vždy odpojte nabíječku.

3.8. První test

POZOR! Před spuštěním testu přečtěte pozorně celou následující část.

Pokud se objeví nějaký rozdíl od následujícího popisu, měli byste okamžitě vypnout robot a přesně zjistit kde je příčina problému. Když nedostanete uspokojivou odpověď v kapitole “Lokalizace a odstranění problémů”, můžete kontaktovat technickou podporu výrobce.

OK – hotovo. Zapněte robot. Dvě stavové LED ve střední části by měly svítit. Po malém zpoždění zhasnou, další červená LED (SL6) začne blikat a zelená LED (SL1) svítí trvale. Tento stav signalizuje absenci uživatelského programu v paměti řídícího mikroprocesoru. Pokud je v paměti funkční uživatelský program, bude pouze blikat zelená stavová LED SL1.

Žlutá LED PWRON by se měla rozsvítit asi jednu sekundu po zapnutí robotu – odpojením většiny snímačů včetně enkoderů se šetří energie.

Asi po 30 sekundách začne blikat červená LED SL6 a všechny ostatní LED zhasnou. Mikroprocesor robotu se automaticky přepne do pohotovostního režimu a program se dále neprovádí. Pohotovostní režim s nízkou spotřebou se může ukončit přes rozhraní USB, stisknutím tlačítka START/STOP nebo krátkým vypnutím a zapnutím robotu. Během pohotovostního režimu odebírá robot malé množství energie (maximálně 5 mA) – a pokud nechcete systém delší dobu používat, nezapomeňte RP6 vypnout úplně. Program v paměti robotu neprovede automaticky přechod do pohotovostního režimu. Místo toho bude systém

pokračovat čekáním na uživatelský příkaz ze sériového rozhraní (jednoduše vyslání “s”), ze sběrnice I^2C nebo tlačítka Start/Stop.

3.8.1. Připojení rozhraní USB a spuštění RP6Loaderu

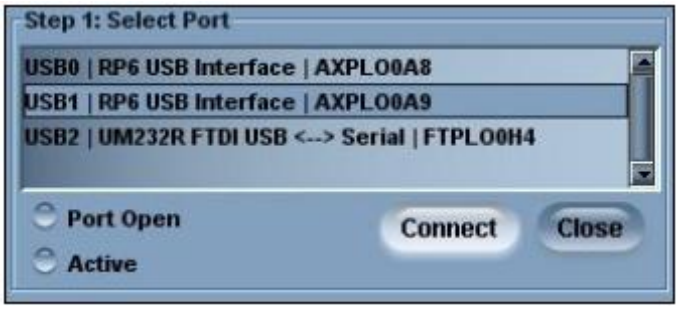
Následovat bude test aktualizace programu přes rozhraní USB. Připojte, prosím, rozhraní USB do PC (vždy se začíná tím, že připojíte rozhraní do PC). Pak zapojte rozhraní USB do konektoru “PROG/UART” umístěného na robotu těsně vedle tlačítka Start/Stop.

Konektor má mechanickou ochranu proti přepólování a 10 vývodovou zástrčku nemůžete bez hrubého násilí zasunout špatně.

Nyní spustíte RP6Loader.

Podle zvoleného jazyka se mohou lišit názvy položek v menu.

Snímky obrazovek ukazují anglickou verzi programu a pokud chcete můžete jazyk změnit v nabídce “Options->Preferences”, následuje výběr položky “Language”
(v současné době jsou k dispozici pouze anglická a německá verze) a stisknutí OK. Po změně jazyka musíte provést restart RP6Loaderu



Otevření portu – Windows

Nyní můžete vybrat USB port. Pokud váš počítač nenabízí další USB sériový adaptér s řadičem FTDI, zobrazí se seznam portů jen jednou, než jej vyberete. Pokud však existuje několik portů, můžete ho identifikovat zaměřením na “RP6 USB Interface” (nebo “FT232R USB UART”), za kterým následuje předem určené pořadové číslo.

Pokud se nezobrazí žádný port, obnovte seznam zařízení pomocí “RP6Loader-->Refresh Port list”.

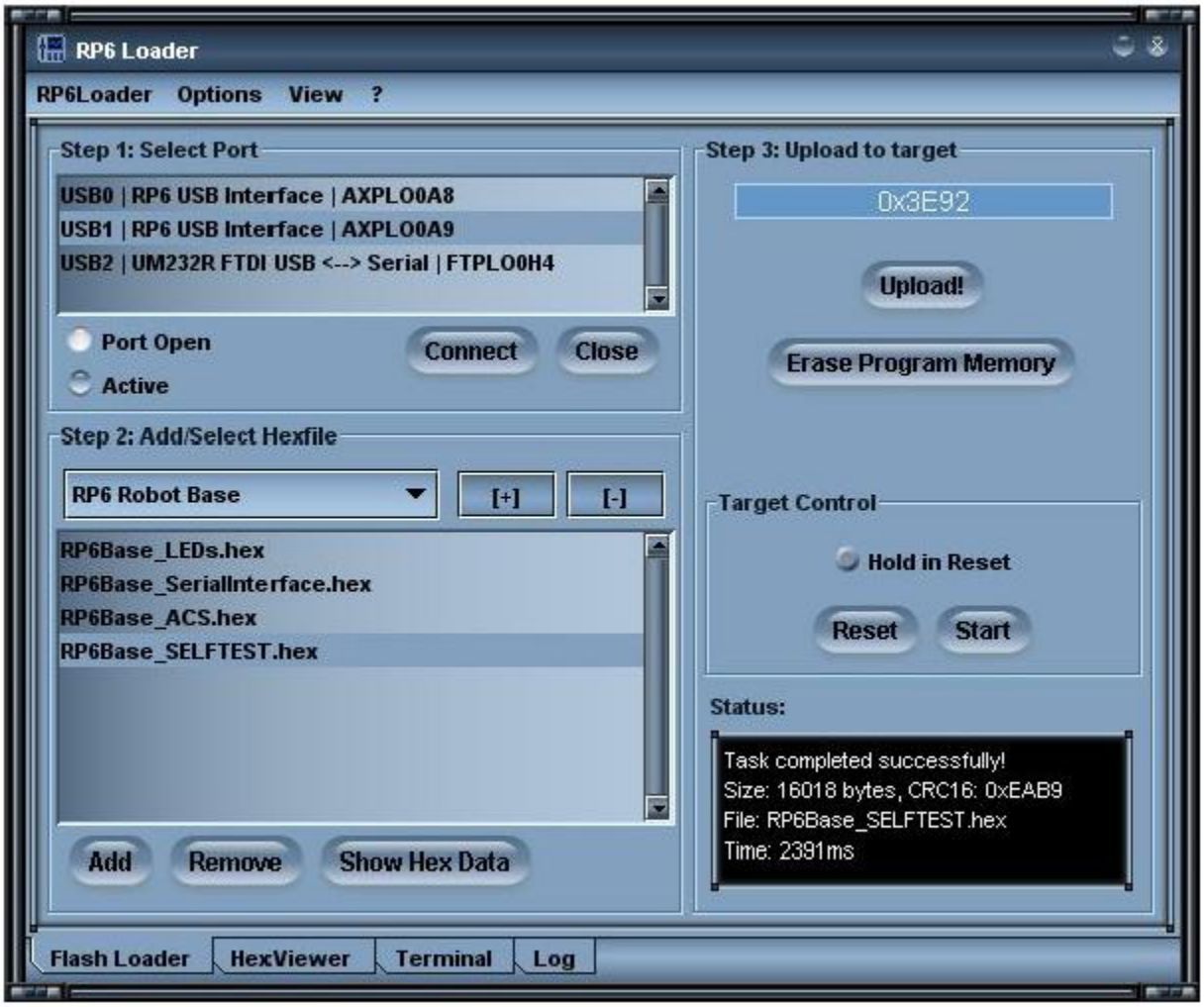
Otevření portu – Linux

Linux obsluhuje USB – sériové adaptéry stejně jako ostatní standardní COM porty. Instalace ovladače D2XX na operační systém Linux není tak jednoduchá a moderní jádra Linux již poskytují ovladač pro standardní virtuální COM port (VCP). Obecně se port používá stejně jako ve Windows, ale budete muset zkusit, na kterém portu je právě připojeno USB rozhraní RP6 a nemusíte odstranit USB port z počítače při odpojení (jinak se musí před otevřením portu znovu spustit RP6Loader).

Virtuální COM porty budou označeny “/dev/ttyUSBx”, kde x představuje číslo portu tj. “/dev/ttyUSB0” nebo “/dev/ttyUSB1”. Zobrazí se také označení standardních COM portů “/dev/ttyS0”, “/dev/ttyS1”. Program RP6Loader si pamatuje naposledy vybraný port a bude ho automaticky nabízet při spuštění programu (většinou se pamatuje jediný přítomný a vybraný port).

Nyní klikněte na tlačítko “Connect”. Program RP6Loader se pokusí otevřít port a zkontroluje komunikaci s bootloaderem robotu. Pokud vše funguje správně, zobrazí černé “stavové” pole, které ukazuje zprávu “Connected to: RP6 Robot Base ...”, doprovázenou změřenou hodnotou napětí baterie. Pokud se spojení nepovede, počkejte chvíli a zkuste to znovu. Pokud selže i druhý pokus, objeví se řada chyb. V takovém případě okamžitě vypněte robot a prostudujte kapitolu “Vyhledávání a řešení problémů” v příloze.

Nízké napětí baterie oznámí program varovnou zprávou. Jakmile uvidíte tuto zprávu, měli byste nabít baterie. Naše rada zní – nabijte baterii, jakmile napětí klesne pod 5,9 V!

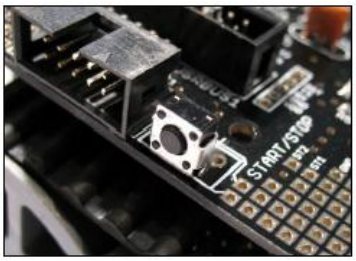


Jakmile je tato počáteční kontrola úspěšná, můžete spustit jednoduchý samo testující program, který zkontroluje, zda dílčí systémy robotu fungují správně. Nejdříve musíte program Selftest přidat do seznamu souborů typu hex. Může se to udělat tlačítkem “Add” a v adresáři příkladů se vybere soubor “RP6Base_SELFTEST\RP6Base_SELFTEST.hex”. Vybraný soubor obsahuje samo testující program v hexadekadickém tvaru – proto se soubor nazývá “soubor HEX”.

Zvolený soubor se nyní objeví v seznamu. Tímto způsobem můžete vybrat další soubory typu HEX ze svých vlastních programů nebo příkladů a přidat je do seznamu. (viz snímek obrazovky, ve které je již přidáno několik souborů HEX). RP6Loader je schopen spravovat několik souborů HEX pro pohodlnou aktualizaci. Oceníte to při použití několika rozšiřujících modulů nebo existenci různých verzí programu. Při ukončení programu se seznam souborů automaticky uloží. Samozřejmě se uloží včetně názvu cesty. Během vývoj programu musíte přidat soubor HEX pouze jednou. Po opakované kompilaci, můžete aktualizovat novou verzi, bez opakovaného přidávání do seznamu (při aktualizaci můžete použít zkratku [STRG+D] nebo [STRG+Y] pro spuštění programu po přenosu dat). Názvy cest se u jednotlivých operačních systémů liší, proto RP6Loader používá samostatní seznamy pro Windows a Linux.

Nyní v seznamu vybereme soubor “RP6Base_SELFTEST.hex” a klikneme na “Upload!” v pravém horním rohu zobrazení se objeví progress bar. Tím se zahájí proces aktualizace programu v mikroprocesoru MEGA32. Aktualizace by měla být hotová během několika sekund (u samo testujícího programu maximálně 5 sekund).

Po dokončení aktualizace vyberte, ve spodní části programového okna, záložku “Terminal” nebo ji můžete alternativně vybrat v nabídce “View”.



Tlačítkem Start/Stop, které je umístěné blízko programovacího konektoru (viz obrázek), spusťte program. Později můžete používat tlačítka v software RP6Loader nebo zkrácenou volbu [STRG]+[S]. Ale nyní však použijeme hardwarové tlačítko, protože můžeme zjistit, zda funguje správně.

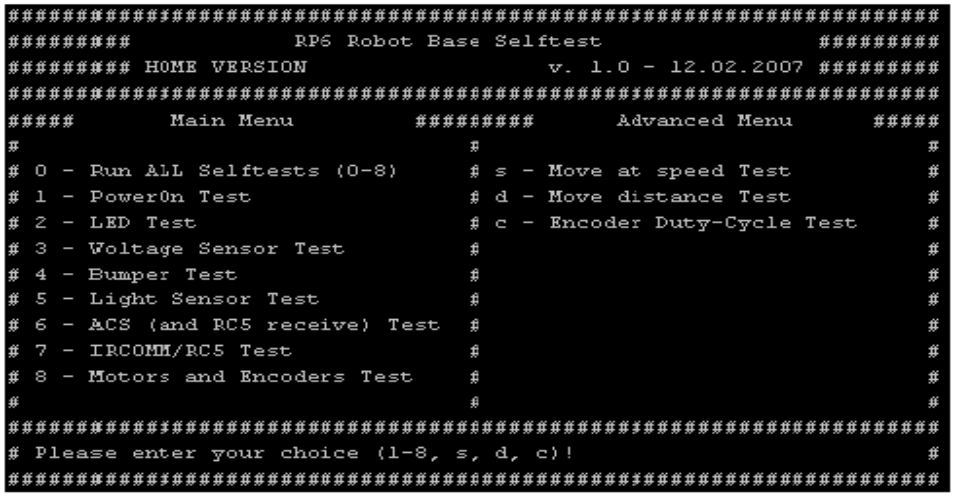
Na terminálu se může objevit varovná zpráva, která říká, že během testu číslo 8 spustí RP6 motory.

POZOR! Pokud běží test číslo 8 (“Test motorů a enkoderů”), zvedněte a držte RP6 v rukách nebo ho alternativně položte na vhodný předmět – který zabrání kontaktu pásů s povrchem. **Během testu číslo 8 se pásy NESMÍ zatěžovat nebo blokovat**. V opačném případě test pravděpodobně selže. Pokud se RP6 dotýká země, mohou motory reagovat nepředvídatelně, výsledkem je chyba testu. Nejdůležitější je, že RP6 může také ujet nějakou vzdálenost. Vyrazí vpřed, dokud mu to dovolí USB kabel…

RP6 musíte držet v rukách nebo alternativně položit na předmět (malou kostku nebo dálkový ovladač). Když RP6 umístíte na předmět, držte robot během testu jednou rukou, aby nečekaně nesklouzl na stůl.

Tato varovná zpráva se zobrazí těsně před testem číslo 8 a musí se před spuštěním testu potvrdit.

Do terminálového okna vložte malé písmeno 'x' a stiskněte Enter (tento postup můžete opakovat, kdykoliv se objeví tato zpráva nebo když se test dokončí…).



V tomto bodě program umístí text nabídky na levou stranu. V dalších verzích programu se může text změnit.

Vložením příslušné číslice nebo písmena a stiskem Enter můžete vybrat a spustit různé testovací programy

Chceme-li spustit všechny standardní testy - napíše '0' a stiskneme Enter!

V terminálovém okně se objeví následující textový výstup:

```
# 0
################################################################################
################################################################################
## Test #1 ##

### POWER ON TEST ###
Please watch the yellow Power0n LED and verify that it lights up!
(it will flash a few times!)
```

Sledujte žlutou LED přeron, která několikrát blikne. Pokud nebliká, může test zmizet dříve, než si všímnete nebo se skutečně objevila chyba. Testovací program však pokračuje, protože neexistuje automatická metoda pro detekci správné funkce – závisí to jen na vás.

V tomto případě LED zobrazuje stav, kdy jsou aktivovány enkodéry, IR přijímač a snímač proudu. Tato zařízení společně s LED odebírají rozumný proud okolo 10 mA --> pro úsporu energie se tato zařízení spínají na dobu nezbytně nutnou.

Program nyní rozbliká stavové LED. Všechny LED několikrát bliknou současně a pak každá z nich samostatně. Zde můžete pozorovat, zda jednotlivé LED fungují správně nebo je některá poškozená.

Výstup bude vypadat následovně:

```
## Test #2 ##

### LED Test ###
Please watch the LEDs and verify that they all work!
Done!
```

Dále se testuje snímač baterie. Ve skutečnosti byl již snímač otestován, když RP6Loader dříve ukázal napětí baterie. Kontrola baterie se teď zopakuje s úplným výpisem:

```
################################################################################
################################################################################
## Test #3 ##

### Voltage Sensor Test ###
Be sure that you are using good accumulators!

Enter "x" and hit return when you are ready!
```

Potvrďte vložením 'x'.

```
# x
Performing 10 measurements:
Measurement #1:  07.20V --> OK!
Measurement #2:  07.20V --> OK!
Measurement #3:  07.20V --> OK!
Measurement #4:  07.20V --> OK!
Measurement #5:  07.20V --> OK!
Measurement #6:  07.20V --> OK!
Measurement #7:  07.20V --> OK!
Measurement #8:  07.20V --> OK!
Measurement #9:  07.20V --> OK!
Measurement #10: 07.20V --> OK!
Done!
```

Tento výstup se vám může zdá být nudný – hodnoty se mohou měnit v celém přípustném rozsahu od 5,5 V do 9,5 V. Pokud se tyto limity překročí, zobrazí se chyba. Když se objeví chybové hlášení, zkontrolujte baterie – nemusí být správně nabité nebo jsou poškozené. Pokud jsou baterie v pořádku, pak může být poškozený snímač (dva rezistory…).

Nyní zkontrolujeme nárazníky. Testujeme je postupně stisknutím mikrospínačů, sledováním LED a podle zpráv zobrazených na terminálu. Každý “náraz” by se měl promítnout na terminálu i LED. Výstupní zpráva zobrazí:

```
## Test #4 ##

Bumper Test
Please hit both bumpers and verify
that both Bumpers are working properly!
The Test is running now. Enter "x" and hit return to stop this test!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
OBSTACLE: LEFT!
OBSTACLE: RIGHT!
FREE: LEFT!
FREE: RIGHT!
```

Pokud se test dokončil, můžete test opustit vložením 'x' + Enter.

Nyní budeme kontrolovat světelné snímače. Při testu těchto snímačů je postupně zakrývejte rukou, posouvejte ruku nad snímač a kontrolujte změnu hodnoty a LED – snižování intenzity světla se musí projevit snižováním naměřené hodnoty. Led se rozsvítí, když na snímač působí jasné světlo. Denní světlo obvykle vytváří hodnoty v rozsahu od 200 do 900.

Pokud na snímače přímo posvítíme baterkou nebo robot namíříme přímo do slunce, může naměřená hodnota překročit 1000. Ve tmavé místnosti může hodnota klesnout pod 100.

Test spustíte vložením 'x' + Enter:

```
## Test #5 ##

### Light Sensor Test ###
Please get yourself a small flashlight!
While the test runs, move it in front of the Robot
and watch if the values change accordingly!
```

```
Enter "x" and hit return when you are ready!
# x
The Test is running now. Enter "x" and hit return to stop this test!
Performing measurements...:
Left: 0510, Right: 0680
Left: 0511, Right: 0679
Left: 0512, Right: 0680
Left: 0560, Right: 0710
Left: 0630, Right: 0750
Left: 0640, Right: 0760
Left: 0644, Right: 0765
[...]
```

Po otestování snímačů ukončíte testovací posloupnost vložením 'x'.

Nyní přikročíme k testu ACS. Zde se nic nepotvrzuje a test se okamžitě spustí. Nyní se oprostíme překážek před robotem, ale pečlivě vyčistíme prostor před robotem, aby nedošlo k detekci překážky.

Test může zobrazovat následující výstup:

```
## Test #6 ##

ACS Test
Please move your hand or other obstacles in front of the Robot
and verify that both ACS channels are working properly!

ACS is set to Medium power/range!

You can also send RC5 Codes with a TV Remote Control
to the RP6 - it will display the Toggle Bit, Device Adress
and Keycode of the RC5 Transmission!
Make sure your remote control transmits in RC5 and not
SIRCS or RECS80 etc.! There are several other formats that will NOT work!

The Test is running now. Enter "x" and hit return to stop this test!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
FREE: LEFT!
OBSTACLE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
FREE: LEFT!
```

Test také umožňuje příjem zpráv z IR dálkového ovladače kompatibilního se systémem RC5. V tomto případě přijímá bit TOGGLE a zobrazuje adresu a kód klávesy.

Pokračuje se ukončením testu vložením 'x'.

Následuje testovací procedura IRCOMM, která se může spustit vložením 'x'. Procedura začíná vysláním IR datových paketů, zobrazením přijatých paketů na terminálu a automatickou kontrolou správnosti přijatých dat (pomocí relativně výkonných IR LED bude IRCOMM obvykle přijímat zpět vlastní signály. Pouze při absenci reflexních předmětů nebo zastíněním systému může dojít k chybě – to jsou však velmi neobvyklé podmínky).

Výstup může vypadat takto:

```
#### TEST #7 ####

IRCOMM Test
[...]

TX RC5 Packet: 0
RX RC5 Packet --> Toggle Bit:0 | Device Address:0 | Key Code:0 --> OK!
TX RC5 Packet: 3
RX RC5 Packet --> Toggle Bit:0 | Device Address:3 | Key Code:3 --> OK!
TX RC5 Packet: 6
RX RC5 Packet --> Toggle Bit:0 | Device Address:6 | Key Code:6 --> OK!
TX RC5 Packet: 9
RX RC5 Packet --> Toggle Bit:0 | Device Address:9 | Key Code:9 --> OK!
TX RC5 Packet: 12
RX RC5 Packet --> Toggle Bit:0 | Device Address:12 | Key Code:12 --> OK!
[...]
TX RC5 Packet: 57
RX RC5 Packet --> Toggle Bit:1 | Device Address:25 | Key Code:57 --> OK!
TX RC5 Packet: 60
RX RC5 Packet --> Toggle Bit:1 | Device Address:28 | Key Code:60 --> OK!
TX RC5 Packet: 63
RX RC5 Packet --> Toggle Bit:1 | Device Address:31 | Key Code:63 --> OK!

Test finished!
Done!
```

Test zabere asi 5 sekund.

Na závěr provedeme test motoru a enkodéru. RP6 musíte zvednout do rukou – pásy se nesmí dotýkat podlahy ani jiných předmětů.

V opačném případě se test zřejmě zbortí. Pokud RP6 položíte na horní část předmětu, jak bylo popsáno, dávejte pozor aby RP6 nesklouzl na stůl.

Tento test nebude trvat dlouho – přibližně 30 sekund. Pečlivě sledujte chybová hlášení tohoto testu. Může se objevit při jediné chybě měření, výskyt chybové zprávy způsobí ukončení testovací posloupnosti. Pokud se očekává spuštění motorů a test se ukončí někde uprostřed, nebuďte z toho smutní. Pokud se to stane, zkuste to znovu – po prostudování kapitoly “Vyhledávání a odstraňování problémů” v příloze.

Testovací procedura postupně zvyšuje rychlost obou motorů až na 50% maximální rychlosti a několikrát změní směr otáčení motorů. Systém bude neustále kontrolovat a hlídat naměřené hodnoty z enkodérů a snímačů proudu. Pokud se během přesunu něco pokazí (tj. zkratuje některý motor nebo zablokuje převodovka – na které upozornila předchozí fáze testu po vložení baterie) sledovaný proud prudce roste do vysokých hodnot a způsobí okamžité ukončení testu.

Vzorek testovacího protokolu (zkrácený):

```
##########################################################################
##########################################################################
#### TEST #8 ####

Automatic speed speed regulation test

##########################################################################
### ATTENTION!!! DANGER!!! WARNING!!!
Make sure that the RP6 can __NOT__ move!
The caterpillar tracks should __NOT__ touch the ground!
(hold it in your hands for example...)
THE RP6 WILL START MOVING FAST! YOU CAN DAMAGE IT IF YOU DO NOT
MAKE SURE THAT IT CAN __NOT__ MOVE!
Make sure both crawler tracks are FREE RUNNING! DO NOT BLOCK THEM!
--> OTHERWISE THE TEST WILL FAIL!
##########################################################################

Enter "x" and hit return when TO START THIS TEST!
Make sure the RP6 can not move!

# x
T: 000 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
T: 000 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 002 |IR: 003 |UB: 07.28V
[...]
Speed Left: OK
Speed Right: OK
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
T: 020 |VL: 000 |VR: 000 |PL: 020 |PR: 020 |IL: 006 |IR: 009 |UB: 07.26V
T: 020 |VL: 001 |VR: 014 |PL: 039 |PR: 030 |IL: 020 |IR: 020 |UB: 07.27V
[...]
Speed Left: OK
Speed Right: OK
T: 040 |VL: 021 |VR: 019 |PL: 037 |PR: 028 |IL: 025 |IR: 021 |UB: 07.25V
T: 040 |VL: 020 |VR: 020 |PL: 037 |PR: 029 |IL: 026 |IR: 022 |UB: 07.25V
T: 040 |VL: 018 |VR: 020 |PL: 044 |PR: 036 |IL: 028 |IR: 023 |UB: 07.23V
T: 040 |VL: 038 |VR: 038 |PL: 055 |PR: 044 |IL: 035 |IR: 029 |UB: 07.23V
T: 040 |VL: 037 |VR: 042 |PL: 055 |PR: 043 |IL: 033 |IR: 028 |UB: 07.24V
T: 040 |VL: 043 |VR: 041 |PL: 052 |PR: 042 |IL: 032 |IR: 026 |UB: 07.23V
T: 040 |VL: 043 |VR: 041 |PL: 052 |PR: 040 |IL: 030 |IR: 024 |UB: 07.24V
T: 040 |VL: 037 |VR: 041 |PL: 052 |PR: 040 |IL: 030 |IR: 023 |UB: 07.24V
T: 040 |VL: 043 |VR: 040 |PL: 050 |PR: 039 |IL: 029 |IR: 022 |UB: 07.24V
Speed Left: OK
Speed Right: OK
T: 060 |VL: 040 |VR: 039 |PL: 053 |PR: 040 |IL: 033 |IR: 024 |UB: 07.24V
T: 060 |VL: 036 |VR: 040 |PL: 053 |PR: 040 |IL: 034 |IR: 026 |UB: 07.24V
T: 060 |VL: 042 |VR: 039 |PL: 052 |PR: 041 |IL: 034 |IR: 027 |UB: 07.23V
T: 060 |VL: 042 |VR: 040 |PL: 063 |PR: 052 |IL: 038 |IR: 032 |UB: 07.22V
T: 060 |VL: 058 |VR: 060 |PL: 068 |PR: 056 |IL: 038 |IR: 032 |UB: 07.25V
T: 060 |VL: 062 |VR: 062 |PL: 067 |PR: 054 |IL: 037 |IR: 029 |UB: 07.22V
T: 060 |VL: 060 |VR: 062 |PL: 067 |PR: 053 |IL: 038 |IR: 028 |UB: 07.23V
[...]
Speed Left: OK
Speed Right: OK
T: 100 |VL: 082 |VR: 078 |PL: 080 |PR: 068 |IL: 043 |IR: 036 |UB: 07.23V
T: 100 |VL: 079 |VR: 079 |PL: 081 |PR: 069 |IL: 047 |IR: 038 |UB: 07.22V
T: 100 |VL: 078 |VR: 082 |PL: 092 |PR: 078 |IL: 049 |IR: 039 |UB: 07.23V
T: 100 |VL: 095 |VR: 099 |PL: 101 |PR: 082 |IL: 055 |IR: 039 |UB: 07.20V
T: 100 |VL: 098 |VR: 100 |PL: 109 |PR: 081 |IL: 056 |IR: 040 |UB: 07.19V
T: 100 |VL: 095 |VR: 099 |PL: 111 |PR: 082 |IL: 062 |IR: 042 |UB: 07.19V
T: 100 |VL: 102 |VR: 101 |PL: 111 |PR: 082 |IL: 058 |IR: 041 |UB: 07.21V
T: 100 |VL: 102 |VR: 101 |PL: 109 |PR: 081 |IL: 056 |IR: 039 |UB: 07.20V
T: 100 |VL: 093 |VR: 100 |PL: 113 |PR: 081 |IL: 063 |IR: 038 |UB: 07.20V
```

```
T: 100 |VL: 104 |VR: 099 |PL: 112 |PR: 082 |IL: 056 |IR: 042 |UB: 07.22V
Speed Left: OK
Speed Right: OK
T: 080 |VL: 086 |VR: 071 |PL: 022 |PR: 000 |IL: 020 |IR: 012 |UB: 07.28V
T: 080 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 001 |IR: 003 |UB: 07.28V
T: 080 |VL: 004 |VR: 011 |PL: 088 |PR: 084 |IL: 051 |IR: 045 |UB: 07.21V
T: 080 |VL: 079 |VR: 101 |PL: 103 |PR: 077 |IL: 064 |IR: 039 |UB: 07.21V
T: 080 |VL: 082 |VR: 076 |PL: 098 |PR: 072 |IL: 061 |IR: 041 |UB: 07.19V
T: 080 |VL: 081 |VR: 081 |PL: 096 |PR: 071 |IL: 055 |IR: 040 |UB: 07.20V
T: 080 |VL: 080 |VR: 082 |PL: 095 |PR: 070 |IL: 057 |IR: 038 |UB: 07.21V
T: 080 |VL: 082 |VR: 080 |PL: 094 |PR: 069 |IL: 058 |IR: 036 |UB: 07.22V
T: 080 |VL: 077 |VR: 080 |PL: 095 |PR: 069 |IL: 056 |IR: 036 |UB: 07.23V
Speed Left: OK
Speed Right: OK
T: 060 |VL: 082 |VR: 079 |PL: 095 |PR: 069 |IL: 054 |IR: 038 |UB: 07.22V
T: 060 |VL: 079 |VR: 079 |PL: 095 |PR: 071 |IL: 058 |IR: 040 |UB: 07.21V
T: 060 |VL: 082 |VR: 081 |PL: 093 |PR: 070 |IL: 056 |IR: 039 |UB: 07.19V
T: 060 |VL: 069 |VR: 070 |PL: 080 |PR: 054 |IL: 048 |IR: 029 |UB: 07.23V
T: 060 |VL: 064 |VR: 059 |PL: 075 |PR: 054 |IL: 046 |IR: 029 |UB: 07.22V
T: 060 |VL: 058 |VR: 057 |PL: 075 |PR: 055 |IL: 043 |IR: 032 |UB: 07.24V
T: 060 |VL: 059 |VR: 059 |PL: 075 |PR: 056 |IL: 046 |IR: 034 |UB: 07.23V
T: 060 |VL: 060 |VR: 059 |PL: 075 |PR: 056 |IL: 046 |IR: 035 |UB: 07.23V
T: 060 |VL: 057 |VR: 060 |PL: 076 |PR: 056 |IL: 047 |IR: 033 |UB: 07.22V
T: 060 |VL: 058 |VR: 061 |PL: 077 |PR: 055 |IL: 045 |IR: 030 |UB: 07.23V
Speed Left: OK
Speed Right: OK
T: 040 |VL: 045 |VR: 035 |PL: 043 |PR: 023 |IL: 027 |IR: 018 |UB: 07.24V
T: 040 |VL: 000 |VR: 000 |PL: 011 |PR: 000 |IL: 013 |IR: 007 |UB: 07.28V
T: 040 |VL: 002 |VR: 000 |PL: 038 |PR: 038 |IL: 015 |IR: 014 |UB: 07.24V
T: 040 |VL: 038 |VR: 061 |PL: 059 |PR: 052 |IL: 035 |IR: 035 |UB: 07.24V
T: 040 |VL: 044 |VR: 043 |PL: 057 |PR: 044 |IL: 035 |IR: 028 |UB: 07.23V
T: 040 |VL: 038 |VR: 039 |PL: 057 |PR: 044 |IL: 035 |IR: 027 |UB: 07.24V
T: 040 |VL: 039 |VR: 042 |PL: 055 |PR: 043 |IL: 033 |IR: 025 |UB: 07.23V
T: 040 |VL: 043 |VR: 041 |PL: 053 |PR: 041 |IL: 032 |IR: 023 |UB: 07.24V
T: 040 |VL: 040 |VR: 041 |PL: 054 |PR: 041 |IL: 032 |IR: 023 |UB: 07.25V
Speed Left: OK
Speed Right: OK
T: 020 |VL: 037 |VR: 040 |PL: 054 |PR: 041 |IL: 031 |IR: 024 |UB: 07.24V
T: 020 |VL: 022 |VR: 019 |PL: 022 |PR: 012 |IL: 017 |IR: 016 |UB: 07.28V
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 004 |IR: 007 |UB: 07.28V
T: 020 |VL: 000 |VR: 006 |PL: 030 |PR: 027 |IL: 020 |IR: 020 |UB: 07.24V
T: 020 |VL: 013 |VR: 019 |PL: 043 |PR: 030 |IL: 029 |IR: 022 |UB: 07.24V
T: 020 |VL: 026 |VR: 020 |PL: 038 |PR: 029 |IL: 027 |IR: 022 |UB: 07.24V
T: 020 |VL: 020 |VR: 021 |PL: 038 |PR: 029 |IL: 028 |IR: 023 |UB: 07.25V
T: 020 |VL: 021 |VR: 020 |PL: 038 |PR: 029 |IL: 028 |IR: 023 |UB: 07.24V
T: 020 |VL: 018 |VR: 019 |PL: 038 |PR: 030 |IL: 027 |IR: 024 |UB: 07.24V
T: 020 |VL: 022 |VR: 020 |PL: 037 |PR: 029 |IL: 027 |IR: 023 |UB: 07.23V
Speed Left: OK
Speed Right: OK

***** MOTOR AND ENCODER TEST OK! *****
```

Měřené hodnoty se vypisují v testu (zleva doprava): T – požadovaná rychlost, VL/VR – změřená rychlost levé/pravé strany, PL/PR – hodnota PWM levé/pravé strany, IL/IR – proud tekoucí do motoru levé/pravé strany, UB – napětí baterie.

Pokud vypadají výstupní hodnoty stejně jako v předchozím výpisu – pak je vše v pořádku.

Pokud něco nefunguje správně a objeví se chybové hlášení, prostudujte v příloze kapitolu “Vyhledávání a odstraňování problémů”.

To je vše. Pokud systém prošel všemi testy, můžete pokračovat následující kapitolou.

4. Programování RP6

Konečně jsme se dostali k části o programování.

4.1. Konfigurace editoru zdrojového textu

Začneme nastavením malého vývojového prostředí. Takzvané “zdrojové texty” programů v jazyce C se musí nějakým způsobem vložit do projektu a editovat.

Samozřejmě nebudeme používat rozsáhlé systémy na zpracování textu jako je OpenOffice nebo Word. Ty zřejmě explicitně nemohou cokoliv zvýraznit. Programový balík Office perfektně umožňuje napsat příručku podobnou této, ale vůbec není vhodný pro vývoj software. Zdrojový kód je prostý text – bez jakéhokoli formátování. Velikost textu, fonty a barvy nejsou pro kompilátor důležité…

Automatické barevné zvýraznění speciálních klíčových slov nebo textových pasáží (tj. komentářů) je však pro lidi velmi užitečné. Tuto a několik dalších vlastností obsahuje editor Programmers Notepad 2 (v následujících kapitolách jej budeme jednoduše nazývat “PN2”), který budeme používat k editaci zdrojových textů (POZOR: Uživatelé Linux budou potřebovat podobný editor. Distribuce Linux obvykle poskytují několik předem nainstalovaných editorů například kate, gedit, exmacs a další).

Vedle zvýraznění klíčových slov a příslušných struktur (které se nazývá “zvýrazněná syntaxe”) nabízí editor elementární správu projektu. V projektu můžete vytvářet balíčky zdrojových souborů. Dále PN2 umožňuje pohodlné volání programů jako AVR-GCC pro kompilaci programů kliknutím na jedinou položku v menu. AVR-GCC je jednoduchý program ovládaný z příkazového řádku zcela bez grafického rozhraní...

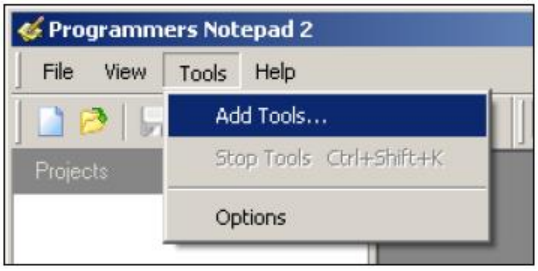
Poslední verzi programátorského editoru můžete najít na domovské stránce projektu:

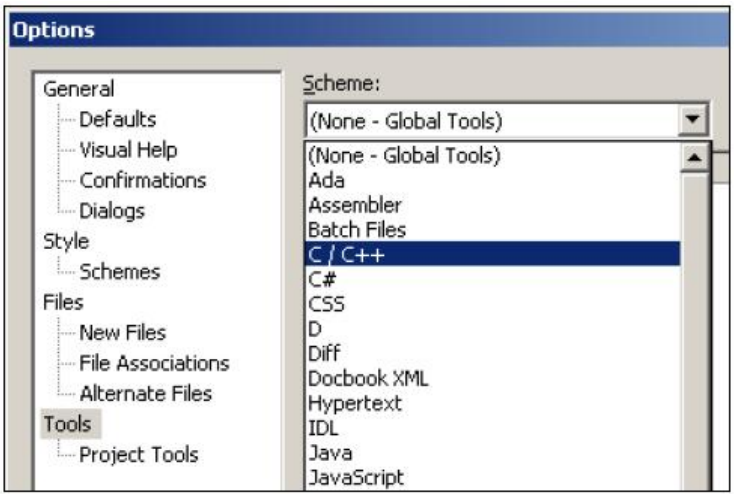
`http://www.pnotepad.org/`

4.1.1. Vytvoření přístupu do menu

POZOR: Tuto kapitolu můžete přeskočit, pokud má PN2 již vytvořené přístupy do menu (tyto přístupy označené “[WinAVR] Make All”, atd... můžete najít v menu. Zkontrolujte, prosím, zda menu obsahuje tyto vstupy). Tuto možnost neobsahují všechny verze PN2. Můžete se také zajímat o přidání dalších programů do menu.

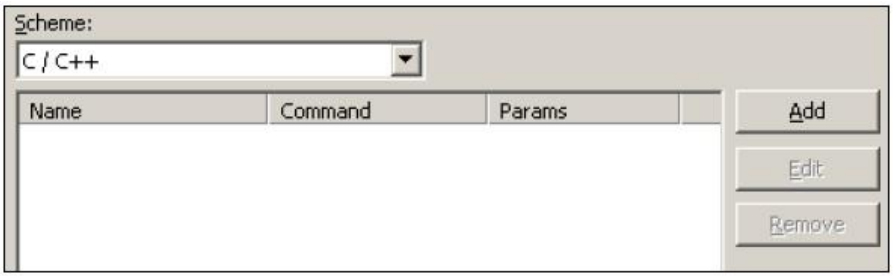
Spusťte PN2 a v nabídce “Tools” vyberte položku “Add Tools...” (viz snímek obrazovky).



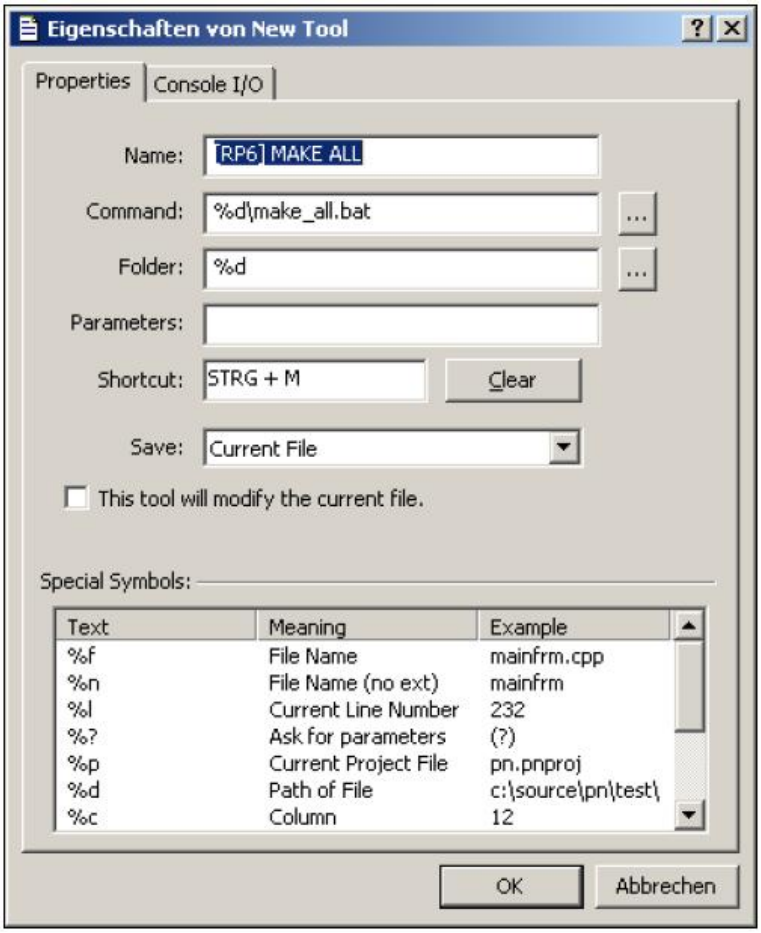


Nyní můžete vložit výběrový dialog, který umožňuje změnit několik nastavení. Budeme však provádět pouze přidávat nové vstupy do nástrojového menu.

Přikročte k výběru “C/C++” ze rozbaleného seznamu “Scheme:”!



Klikněte na “Add”!



Objeví se dialogové okno.

Vložte přesně název podle snímku obrazovky.

Fráze “%d” odkazuje na adresář zvoleného souboru a “%d\make_all.bat” odkazuje na dávkový soubor, který můžete najít v libovolném ukázkovém projektu RP6.

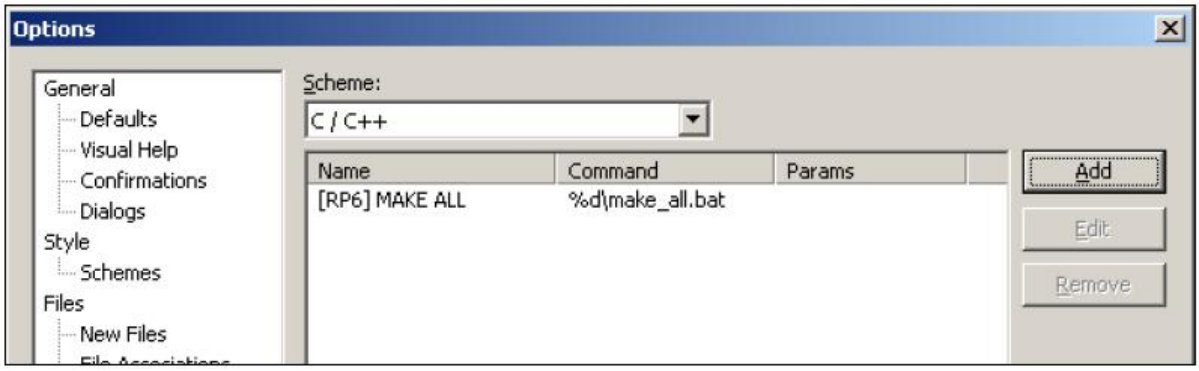
Jako příklad klávesové zkratky můžete z klávesnice vložit [STRG] + [M]!

Tento vstup spustí nástroj “make”, nazvaný dávkový soubor “make_all.bat”, který inicializuje kompilaci souborů v příslušném adresáři zvoleného souboru. Tuto metodu probereme v následující kapitole.

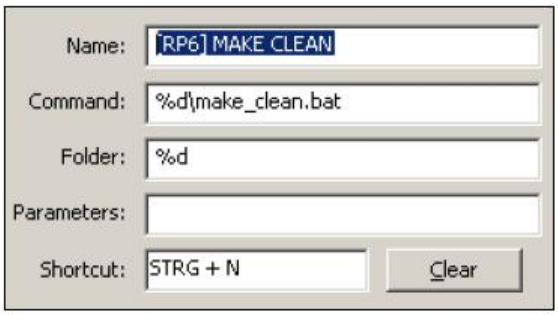
Jako alternativní metodu k “%d/make_all.bat” můžete do pole “Command” jednoduše vložit “make” a “all” do pole “Parameters”.

Ve skutečnosti dávkový soubor jednoduše a přesně provádí zadané povely, ale dávkový soubor zjednoduší spouštění kompilátoru z prostředí Windows Explorer.

Nyní klikněte OK – a nový vstup se zobrazí v seznamu:



...ještě jednou klikněte na “Add”!



Jakmile jste dokončili vstup vytvoření překladu, můžete vložit vše z vedlejšího snímku obrazovky a pak potvrdit kliknutím na OK.

V seznamu se vytvoří nový vstup:

“[RP6] MAKE CLEAN”

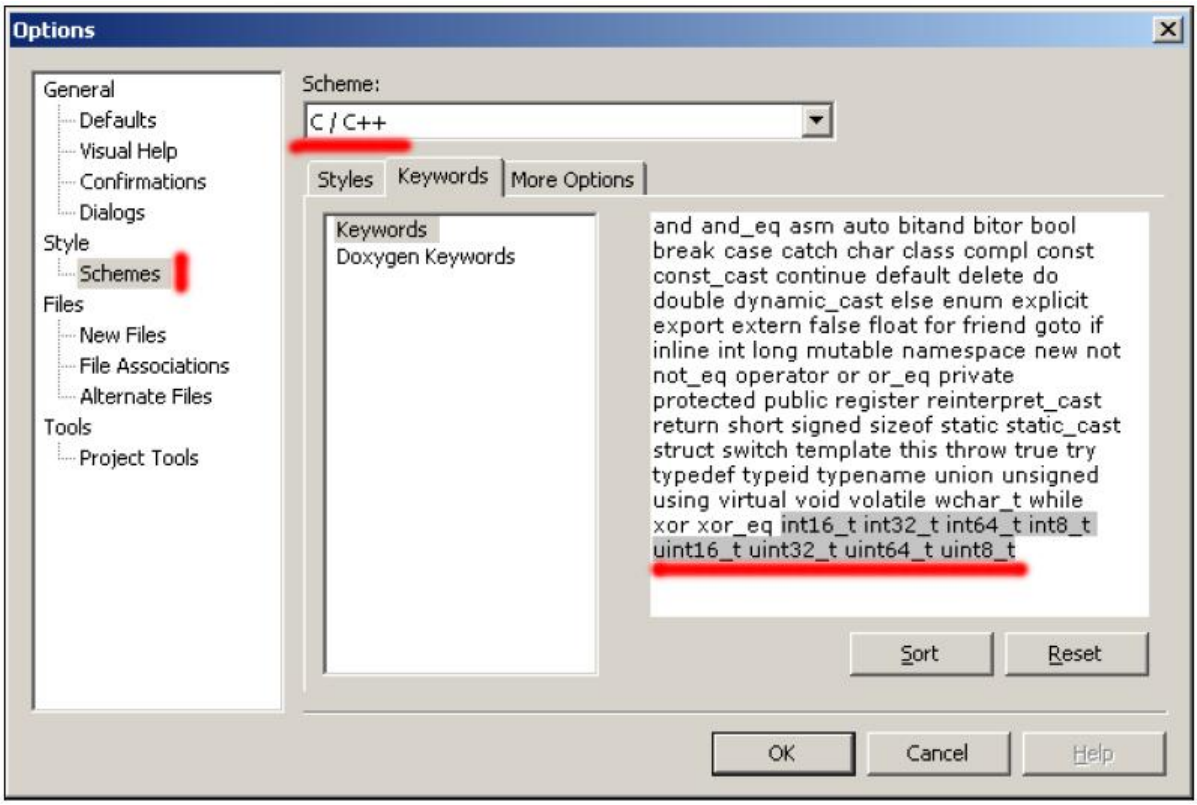
Tento vstup vám umožňuje pohodlné zrušení všech dočasných souborů, které se vytvoří během kompilačního procesu. Obvykle nebudeme tyto dočasné soubory. po úspěšném dokončení překladu, potřebovat. Mimochodem: vytvořený soubor HEX se neodstraní a můžete jej klidně přenést do robota. Jako obvykle můžete (alternativně k “%d/make_clean.bat”) také vložit “make” do pole “Command” a do pole “Parameters” položku “clean”.

Nabídku Options opustíte kliknutím na “OK”.

4.1.2. Konfigurace zvýraznění syntaxe

Dalším nastavením můžete změnit zvýraznění syntaxe. Do schématu standardního jazyka C/C++ můžete přidat několik “klíčových slov”. V dialogovém poli můžete přímo použít Copy & Paste klávesová zkratka ([STRG]+ [C] = kopie, [STRG]+[V] = vložení):

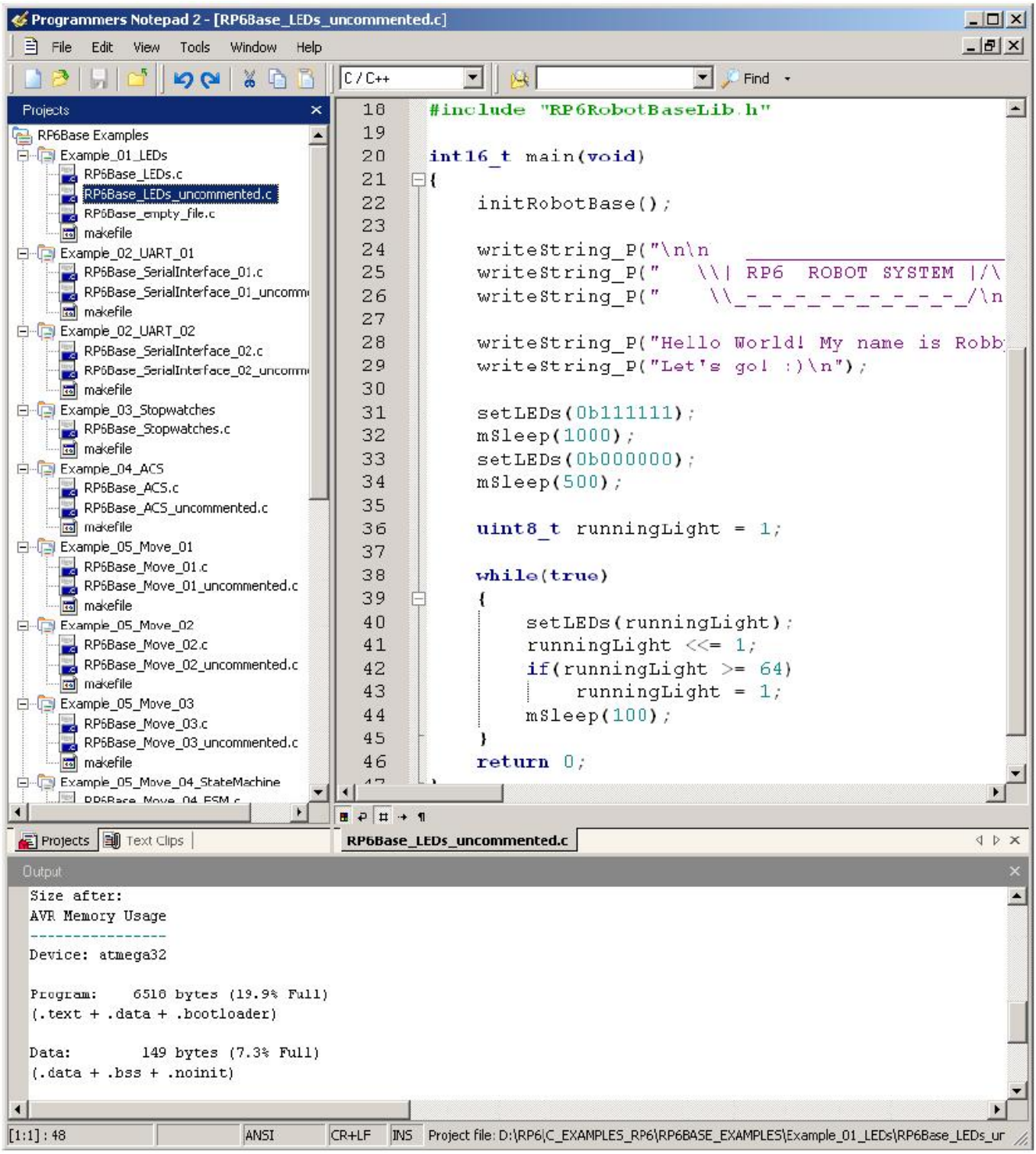
```
int8_t int16_t int32_t int64_t uint8_t uint16_t uint32_t uint64_t
```



Pak klikněte na “Sort” a OK!

Upozornění: *Poslední verze WinAVR a Programmers Notepad (WinAVR-20070525) již obsahují tato klíčová slova v Programmers Notepad! Nejnovější verze prostředí Programmers Notepad se také budou mírně lišit od snímků obrazovek uvedených v této příručce.*

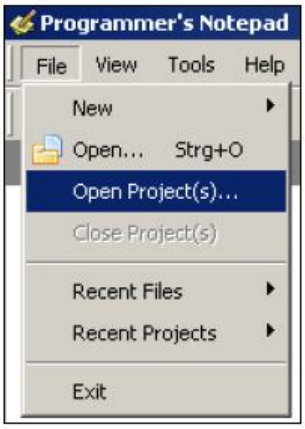
Po přizpůsobení prostředí a otevření ukázkového projektu podle následující části můžete vidět, že se vzhled PN2 může podobat následujícímu snímku obrazovky:



Na levé straně vidíte náhled adresáře se všemi ukázkovými projekty, editor zdrojových textů (vlastnosti jsme probrali v předchozím zvýraznění syntaxe) je na pravé straně a výstup nástroje (v tomto případě výstup překladače) je ve spodní části obrazovky.

Prostředí PN2 můžeme přizpůsobit různými způsoby a vytvořit velké množství užitečných funkcí.

4.1.3. Otevření a kompilace ukázkových projektů



Zkuste, zda všechno funguje správně a otevřte všechny ukázkové projekty:

V nabídce “File” vyberte položku “Open Project(s)”.

Ve standardním dialogu pro výběr souborů můžete v adresáři ukázkových projektů vyhledat dílčí adresář “RP6Base_Examples”.

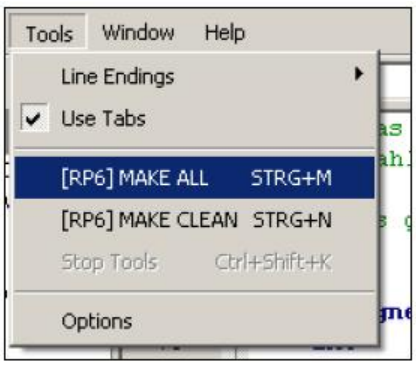
Otevřete soubor “RP6BaseExamples.ppg”, který je projektová skupina PN2. Tento soubor nahraje všechny ukázkové programy a knihovnu RP6Library do seznamu projektů. Tímto způsobem můžete pohodlně brouzdat po ukázkových programech a snadno sledovat funkce umístěné v knihovně RP6Libary.

Nyní otevřete první ukázkový program na vrcholu náhledu adresáře projektů (“Example_01_LEDs” a vyberte soubor “RP6Base_LEDs.c”)! Jednoduše dvakrát klikněte na “RP6Base_LEDs.c” a v editoru se otevře zdrojový soubor!

Ve spodní části obrazovky byste měli vidět výstupní okno PN2. Pokud ne, můžete okno aktivovat v menu volbou View->Output nebo (pokud je okno příliš malé) přesunutím hran pomocí myši (kurzor myši na horní hraně výstupního okna změní svůj tvar na dvojitou šipku...).

Zběžně se podívejte, zda se vám líbí zdrojový text v editoru. Samozřejmě nemůžete očekávat, že budete rozumět celému textu, ale snadno se naučíte, jak s tímto textem zacházet. Pusťte se do toho: zelené řádky textu jsou komentáře, které nejsou součástí vlastního programu, ale pouze dokumentují, co program dělá. Vysvětlíme to (existuje další kopie programu bez komentářů, která ukazuje jak je ve skutečnosti zdrojový text krátký. Komentáře zdrojový soubor značně rozšiřují, ale jsou velmi užitečné pro dokumentaci programu. Nekomentovaná verze se může používat ke kopírování částí kódu do vašich vlastních programů!).

Nyní můžeme zkontrolovat funkci kompilátoru.



Nabídka Tools by měla zobrazovat obě přidana vstupní menu (viz obrázek) nebo alternativně standardní vstupy [WinAVR] prostředí PN2. Můžete vybrat některé z nich, obvykle budou fungovat bez problémů.

Nyní prosím klikněte na “MAKE ALL”!

PN2 nyní zavolá výše popsaný dávkový soubor “make_all.bat”, který spustí “make”. Co znamená pojem “make” vysvětlíme později.

Ukázkový program se přeloží (=“kompiluje”) a vytvoří se soubor hex, který obsahuje speciální kód pro mikroprocesor a může se nahrát do aplikace a provádět!

Kompilátor vytvoří velké množství dočasných souborů (používají přípony jako “.o, .lss, .map, .sym, .elf, .dep”). Nedívejte se na tyto soubory a můžete využít nově vytvořený nástroj “make clean” ke snadnému odstranění těchto souborů! Pro vás je důležitý výsledek překladu pouze soubor HEX! A “make clean” soubor HEX neodstraní.

Spuštění příkazu MAKE ALL v nabídce způsobí následující výstup (výpis je zkrácený a může se od této ukázky nepatrně lišit!):

```
> "make" all
-------- begin -------
[...]

Compiling: RP6Base_LEDs.c

avr-gcc -c -mmcu=atmega32 -I. -gdwarf-2 -Os -funsigned-char -funsigned-bitfields
-fpack-struct
-fshort-enums -Wall -Wstrict-prototypes -Wa,-adhlns=RP6Base_LEDs.lst -I../../RP6lib
-I../../RP6lib/RP6base -I../../RP6lib/RP6common -std=gnu99 -MD -MP –MF
.dep/RP6Base_LEDs.o.d RP6-
Base_LEDs.c -o RP6Base_LEDs.o

Compiling: ../../RP6lib/RP6base/RP6RobotBaseLib.c
[...]

Creating load file for Flash: RP6Base_LEDs.hex
avr-objcopy -O ihex -R .eeprom RP6Base_LEDs.elf RP6Base_LEDs.hex
[...]


Size after:
AVR Memory Usage

Device: atmega32

Program: 6858 bytes (20.9% Full)
(.text + .data + .bootloader)

Data: 148 bytes (7.2% Full)
(.data + .bss + .noinit)

-------- end ------->
Process Exit Code: 0
> Time Taken: 00:01
```

Důležitý řádek je “Process Exit Code: 0” ve spodní části výpisu. Říká nám, že byl proces kompilace a linkování dokončen bez chyb. Ostatní kódy signalizují chyby ve zdrojovém textu, které je třeba před úspěšnou kompilací, nejprve opravit. Pokud jsou ve zdrojovém textu omyly, vystaví překladač na výstupu několik chybových zpráv, které obsahují podrobnější informace o tom, co způsobilo chybu.

Musíte si však uvědomit, že zpráva “Process Exit Code: 0” neznamená program bez chyb‘ Kompilátor není samozřejmě schopen detekovat logické chyby ve vašem programu a nebud chránit robot před nárazem do zdi ;-).

DŮLEŽITÉ: Výstup může obsahovat také upozornění, která mohou být užitečná při identifikaci vážných problémů! Pečlivě si prohlédněte varovná hlášení a pokuste se řešit problémy, které z nich přímo vyplývají! Program uvede také seznam čísel řádků odkazujících na chybové zprávy. Na barevné řádky můžete kliknout a PN2 automaticky skočí na odkazovaný řádek ve zdrojovém textu.

Velmi užitečný je také přehled označený “Využití AVR paměti” na konci výpisu:

```
Size after:
AVR Memory Usage
----------------

Device: atmega32

Program:    6858 bytes (20.9% Full)
(.text + .data + .bootloader)

Data:       148 bytes (7.2% Full)
(.data + .bss + .noinit)
```

Tento přehled ukazuje, že náš program zabírá 6858 byte v paměti a rezervuje 148 byte RAM pro statické proměnné (seznam samozřejmě neobsahuje dynamické proměnné pro haldu a zásobník, ale to pochopíte po prostudování této příručky. Měli byste vždy udržovat volných několik stovek byte na konci RAM). K dispozici máme celkem asi 32 kB (32768 byte) paměti Flash ROM a 2 kB (2048 byte) RAM. Boot loader potřebuje 2 kB z 32 kB Flash ROM – k volnému využití zůstává 30 kB. Vždy bedlivě sledujte velikost programu, dávejte pozor, zda se vejde do dostupné paměti! (Program RP6Loader nedokáže přenést větší programy!).

Předchozí ukázkový program nechá v paměti ROM 23682 volných byte. Ve skutečnosti by mohl tento relativně krátký program RP6Base_LEDs.c zabírat mnohem méně paměti protože obsahuje celou knihovnu RP6Library! Netrapte se tím, je zde dostatek místa pro vaše programy a malé programy nepotřebují příliš mnoho paměti. Knihovna funkcí zabírá více než 6,5 kB paměti Flash, ale již pro vás obsloužila mnoho důležité práce. Váš program bude obvykle menší než knihovna RP6Libary.

4.2. Nahrávání programu do RP6

Nyní se může použít program RP6Loader k nahrávání právě přeloženého programu do robotu. Přidejte, prosím vytvořený soubor HEX do seznamu souborů RP6Loaderu, kliknutím na “Add”, pečlivě vyberte soubor a pak klikněte na “Upload!”, začněte testovacím programem. Nyní se přepněte na terminál a zkontrolujte výstup. Program můžete samozřejmě spustit dříve a sledovat jeho výstup. Na terminálu to snadno provedete stisknutím klávesové zkratky [STRG]+[S], použitím položky “Start” v nabídce nebo vysláním “s” – po resetu musíte samozřejmě chvíli počkat na zprávu “[READY]” z Bootloaderu! S výhodou se může použít kombinace kláves [STRG]+[Y]. Po dokončení nahrávání se touto klávesovou zkratkou okamžitě spustí program!

První ukázkový program je velmi jednoduchý a spustí blikající LED a výstup nějakého textu přes sériové rozhraní.

Před zahájením psaní vašich vlastních programů vás v miniaturním kurzu seznámíme s programovacím jazykem C...

4.3. Proč C? A co to je “GCC”?

Používání programovacího jazyka C je velmi rozšířené – ve skutečnosti je C standardní jazyk, který dříve či později začne používat každý, kdo se zajímá o vývoj software. Kompilátory jazyka C jsou v současné době dostupné pro téměř všechny mikroprocesory, všechny poslední roboty firmy AREXX Engineering (ASURO, YETI a RP6) mohou být programovány v C.

Popularita jazyka C přinesla obrovské množství dokumentace na internetu a v literatuře, která umožní začátečníkům snadné studium programovacího jazyka. Ale pamatujte: C je poměrně složitý jazyk, který se nedá bez předchozích zkušeností naučit během pár dní (takže nevyhazujte robot z okna, když nefunguje přímo tak, jak jste chtěli).

Základy se dají naštěstí snadno pochopit a programátor může neustále rozvíjet znalosti a zkušenosti. Vyžaduje to jisté počáteční úsilí. Jazyk C se nemůžete naučit automaticky – na rozdíl od výuky cizích jazyků.

Ale není to tak obtížné, protože základní vědomosti C se snadno pochopí po studiu jiných programovacích jazyků. Základní pojetí je často velmi podobné.

Stejně jako ostatní roboty, potřebuje RP6 speciální verzi kompilátoru C z řady GNU Compiler Collection (zkráceně: GCC). GCC je univerzální kompilační systém, který podporuje širokou škálu jazyků včetně C, C++, Java, Ada a FORTRAN.

Cílová platforma podporovaná GCC není omezena na AVR. Může se použít pro mnohem větší systémy a jsou známé desítky cílových platforem.

Nejvýznamnější projekt, který používá GCC je samozřejmě slavný projekt operačního systému Linux. Většina programů pro Linux může být kompilována pomocí GCC. Proto je možné tento kompilátor vnímat jako velmi profesionální a stabilní nástroj, který se vyuřívá v několika velkých společnostech.

Vysvětlení: Pokud se v této příručce odkazuje na “GCC” nemusí to nezbytně znamenat kompletní sadu kompilačních nástrojů, ale pouze vlastní kompilátor jazyka C. Původní “GCC” se může používat jako zkratka “GNU C Compiler” – nový význam nutně přišel po nástupu některých dalších programovacích jazyků.

Pokud se chcete dozvědět více o GCC, navštivte oficiální webové stránky GCC:

`http://gcc.gnu.org/`

GCC přímo nepodporuje cílovou platformu AVR a musí se jí přizpůsobit. Upravená verze GCC se nazývá AVR-GCC. Distribuce WinAVR je vytvořena jako uživatelská verze pro používání ve Windows. Uživatelé systému Unix budou mít obvykle vlastní verzi kompilátoru, která je vždy přizpůsobena konkrétní verzi operačního systému.

4.4. C – Zhuštěný kurz pro začátečníky

Tato kapitola podává pouze krátký úvod do programování v jazyce C, probírá pouze nezbytné minimum problematiky používané v RP6. Tato část by měla být přehled obecných možností a metod jazyka C. Představíme několik příkladů a základních principů, ale další zkoumání těchto pojmů je na čtenáři.

Tato kapitola není nic víc než velmi zhuštěný kurz. Kompletní popis přesahuje možnosti této příručky a vyžaduje mnohem objemnější knihu. Obchody naštěstí nabízejí velké množství knih, které se zabývají tímto tématem.

Několik[1] z nich si můžete volně prohlížet na internetu.

4.4.1. Literatura

Následující knihy a učebnice popisují programování v jazyce C zejména pro PC a další velké počítače. Několik drobností z těchto učebnic nelze aplikovat na mikroprocesory AVR – jazyk je sice stejný, ale většina knihoven pro typické využití v PC je pro malé 8bitové mikroprocesory příliš rozsáhlá. Nejlepším příkladem může být funkce “printf”, která musí být na PC. Funkce “printf” je dostupná také pro mikroprocesory, ale vyžaduje velký paměťový prostor a spoustu strojového času. Proto se používání této funkce nepreferuje. Místo toho uvidíte mnohem efektivnější alternativy.

Některé učebnice jazyka C/online knihy (skutečně jen malý výběr):

http://www.its.strath.ac.uk/courses/c/

http://www.eskimo.com/~scs/cclass/notes/top.html

http://www.iu.hio.no/~mark/CTutorial/CTutorial.html

http://en.wikibooks.org/wiki/C

http://www.le.ac.uk/cc/tutorials/c/

http://stuff.mit.edu/iap/c/CrashCourseC.html

Existuje také řada výborných knih – nebudeme zde uvádět jejich seznam, stačí když navštívíte knihovnu nebo knihkupectví.

Pokud hodláte udělat jen několik pokusů s robotem, nemusíte kupovat knihu. Hlavní část zkušenosti s programováním můžete kdykoliv nasbírat při "učení na vlastní práci".

Všechny důležité informace můžete najít na uvedených webových stránkách. Ukázkové programy dostupné na CD RP6 se dají snadno rozšiřovat a ukazují řadu věcí. Učební text v této příručce je vhodný také pro první pokusy.

Učebnici pro začátečníky zaměřenou na AVR můžete najít například na:

http://www.avrtutor.com/

Tyto webové stránky zmiňují také některé nástroje (programovací zařízení atd.) a další skutečnosti, které nebudete u RO6 potřebovat. Přesto se na ně můžete podívat.

[1] Webové vyhledávání hesla “c tutorial” přinese výsledek v miliónu odkazů. Samozřejmě není reálné zpracovat takové množství dat, ale mohou být mezi nimi velmi dobré materiály...

Další informace můžete najít na domovských stránkách WinAVR respektive v PDF dokumentaci WinAVR:

http://winavr.sourceforge.net/

http://winavr.sourceforge.net/install_config_WinAVR.pdf

Speciálně zaměřená dokumentace AVR-LibC:

http://www.nongnu.org/avr-libc/user-manual/index.html

kterou můžete v PDF tvaru najít také na CD RP6.

Samozřejmě nemůžete přečíst všechny učebnice a knihy. Tento seznam je pouze průvodce na cestě k dalším informacím. Učebnice se liší velikostí a podrobností, ale zaručeně pomůže přečtení více než jedné z nich.

Stránky rozsáhlé komunity uživatelů AVR a studnice informací je:

http://www.avrfreaks.net/

Zde můžete najít velmi krásné fórum vyhrazené mikroprocesorům AVR, mnoho obecných informací, projekty, učební texty a fragmenty zdrojových textů.

4.4.2. První program

Jak se obvykle říká – učení prací je nejúčinnější způsob výuky jazyka C. Na základě čtení a zvládnutí všeho, co je uvedeno v tomto zhuštěném kurzu, byste se měli pokusit o vlastní program.

Napřed samozřejmě probereme několik základů, ale držme se myšlenky, která byla řečena výše. Začněte jednoduchým programem v jazyku C pro RP6:

```
1  /*
2   * A small and simple "Hello World" C Program for the RP6!
3   */
4
5  #include "RP6RobotBaseLib.h"
6
7  int main(void)
8  {
9        initRobotBase();
10       writeString("Hello World!\n");
11       return 0;
12 }
```

Pokud jste nikdy neprogramovali v C, pak se vám může “zdrojový text” jevit jako cizí jazyk, ale základní pojetí je snadné a lehce zvládnutelné.

Předchozí krátký program je již plně funkční program, ale pouze inicializuje mikroprocesor a napíše text:

"Hello World!" + CR/LF

do sériového rozhraní. Je to typický příklad programování, který můžete najít ve většině knih (na začátku samozřejmě nebude volání initRobotBase).

Při seznamování s novým programovacím jazykem, můžete tento malý program zkopírovat do textového editoru a pokusit se o jeho kompilaci.

Někdo může namítnout, že lze v adresáři příkladů RP6 nalézt mnohem atraktivnější program "Hello World", který obsahuje běžící světlo na LED a nějaké další textové výstupy.

Nyní probereme program z výpisu 1 a vysvětlíme jednotlivé řádky.

Řádek 1 - 3: `/* A small and simple "Hello World" C Program for the RP6! */`

Jedná se o řádky s komentářem a nebudou interpretovány překladačem. Komentáře se používají k dokumentaci zdrojového textu; začínají /* a končí */.

Dokumentace pomůže porozumět zápisu programu i ostatním lidem, ale bude užitečný pro přehlednost vlastních programů, když se k nim po letech vrátíte.

Komentáře mohou být jakkoliv dlouhé nebo “okomentované“ části zdrojových textů budou při testování jinými programy ignorovány bez ztráty původního kódu. Vedle těchto víceřádkových komentářů, podporuje GCC také jednořádkové komentáře, které začínají “//”. PO “//” bude libovolný text, až do konce řádku, interpretován jako komentář.

Řádek 5: `#include "RP6RobotBaseLib.h"`

Tento řádek vloží do programu knihovnu funkcí RP6, která poskytuje řadu užitečných funkcí a předem definovaných objektů pro nízkou úroveň řízení hardware. Při vkládání knihoven používáme takzvané hlavičkové soubory (s příponou “*.h”), které informují kompilátor, kde hledat příslušné funkce. Hlavičkové soubory se používají pro všechny externí odkazy z dostupných souborů jazyka C. Pečlivě prostudujte obsah RP6RobotBaseLib.h a RP6RobotBaseLib.c – mohou objasnit základní principy. V kapitole o pre-procesoru probereme možnosti direktivy “#include” podrobněji.

Řádek 7: `int main(void)`

Tento řádek definuje nejdůležitější funkci ukázkového programu: funkci main. Zatím se stále učíte a podrobně poznáváte funkce, ale právě nyní můžete přijmout myšlenku, že celý program začíná zde.

Řádek 8 a 12:`{ }`

V jazyku C se mohou pomocí složených závorek '{' a '}' definovat takzvané “bloky”. Bloky sdružují několik příkazů.

Řádek 9: `initRobotBase();`

Zde se volá funkce z knihovny RP6Library. Tato funkce bude inicializovat mikroprocesor AVR a konfigurovat AVR hardwarové moduly. Většina funkcí napsaných pro mikroprocesor nebude fungovat správně, pokud neprovedeme inicializaci pomocí funkce initRobotBase(), nezapomeňte vždy zavolat tuto funkci na začátku programu.

Řádek 10: `writeString("Hello World!\n");`

Tento řádek volá funkci "writeString" z knihovny RP6Library s parametrem řetězce "Hello World!\n". Funkce provede výstup textu do sériového rozhraní.

Řádek 11: `return 0;`

Zde končí náš program. Opouštíme funkci main a vracíme hodnotu nula. Návratový kód se obvykle používá v rozsáhlých systémech (s operačním systémem) jako chybový kód nebo podobný význam, ale není potřeba v malých mikroprocesorových systémech. Tuto návratovou hodnotu přidáváme při návratu proto, abychom splnili konvence standardního jazyka C (a jak uvidíte později, programy pro mikroprocesory nikdy nekončí).

Tento krátký program vám dal první zážitek z programování v jazyce C. Než přejdete k dalším ukázkovým programům, probereme některé další základy jazyka.

4.4.3. Základy jazyka C

Jak již bylo uvedeno dříve, je program jazyka C napsaný v čistém ASCII textu (ASCII = americký standard kódování pro předávání informací). Jazyk C je přísně case sensitivní a pokud se funkce jmenuje "MyFavouriteFunction", budete muset při volání funkce použít přesný tvar tohoto názvu! Zápis volání funkce "myfavouritefunction" nebude akceptován!

Mezi všechny příkazy a symboly můžete vložit libovolný počet mezer, tabulátorů a ukončení řádků aniž by došlo k zásahu do syntaxe programu. Jak jste mohli vidět v ukázkovém programu, byly pro zlepšení čitelnosti příkazy členěné pomocí tabulátorů. Členění zdrojového textu není povinné! Program od řádku 7 můžete vměstnat do jediného řádku tj.:

```
1  int main(void){initRobotBase();writeString("Hallo Welt!\n");return 0;}
```

Je to shodný program, ale text je poněkud matoucí. Nicméně jsme odstranili pouze tabulátory, mezery a konce řádků! Kompilátor se vůbec nestará o formátovací styl zápisu! Samozřejmě budeme potřebovat mezeru jako oddělovač mezi klíčovými slovy a proměnnými, jako je třeba "int" a "main" – a dále se nesmí používat zalomení řádku mezi dvěma uvozovkami (alespoň ne bez escape sekvence)!

Složené závorky { } nám umožňují kombinovat několik výrazů a příkazů do bloků, které jsou nezbytné pro sestavení funkce, podmíněných výrazů a cyklů.

Každý výraz musí být ukončen středníkem ';' aby kompilátor rozpoznal jednotlivé příkazy.

Dříve než začnete psát a kopírovat fragmenty programu z tohoto tutoriálu, chtěli bychom vám dát důležitou radu: většina začátečníků snadno zapomene ukončit příkazy středníkem – nebo použijte středník na nesprávných místech a pak se diví, že se program chová podivně! Zapomenutý středník v některé části programu může mít za následek velké množství chybových zpráv – i když skutečná chyba je pouze jediný omyl. Ve skutečnosti, první chybová zpráva s největší pravděpodobností, identifikuje pravý výskyt chyby.

K běžným chybám začátečníků patří zapomenuté uzavření jedné z několika dvojic závorek nebo špatná syntaxe příkazů. Překladače neakceptují žádné syntaktické chyby! Chvíli trvá, než si zvyknete na všechna tato pravidel, ale metodou pokusů a omylu se budete rychle učit.

Každý program jazyka C začíná hlavní funkcí. V podstatě se budou všechny následující příkazy postupně provádět krok za krokem od začátku do konce.

Mikroprocesor AVR není schopen zpracovat několik příkazů současně! Toto omezení nezpůsobuje žádné problémy. Budeme mít dostatek možností, jak řídit tok programu a skákání do jiných částí programu (toto bude projednáno v následující kapitole).

4.4.4. Proměnné

Nejprve se podíváme na ukládání a čtení dat v paměti RAM. Přístup k datům se provádí pomocí proměnných. Jazyk C zná pro proměnné několik datových typů. V podstatě budeme používat 8, 16 nebo 32 bitové celočíselné datové typy, které se mohou používat ve tvaru se znaménkem nebo bez znaménka. Příslušný rozsah hodnot určuje potřebný počet bitů, které se musí vyhradit pro uložení proměnné.

V programech pro RP6 budeme používat následující datové typy:

Typ	Alternativa	Rozsah hodnot	Poznámky
signed char	int8_t	8 bitů: -128...+127	1 Byte
char	uint8_t	8 bitů: 0..255	'' bez znaménka
int	int16_t	16 bitů: -32768...+32767	2 Byte
unsigned int	uint16_t	16 bitů: 0..65535	'' bez znaménka
long	int32_t	32 bitů: –2147483648...+2147483647	4 Byte
unsinged long	uint32_t	32 bitů: 0...4294967295	'' bez znaménka

Nedostatečná standardizace dovoluje definici několika různých velikosti stejného datového typu. Speciálně to platí pro datový typ "int": pro naše mikroprocesory je velikost 16 bitů, ale u moderních PC je to 32 bitů. Z tohoto důvodu preferujeme moderní standardní definici: int16_t.

Taková definice datového typu je vždy stejná: [u] int N _t

```
kde:   u  :  unsigned
       int:  integer
       N  :  počet bitů tj. 8, 16, 32 nebo 64
       _t :  t je označení pro ”typ”, které zabrání kolizi s jinými symboly
```

Na malém mikroprocesoru se bude pečlivě počítat každý jednotlivý byte a jednoznačná definice datových typů pomůže při sledování zabrané paměti. Podle číslice 16 v názvu datového typu můžete okamžitě rozpoznat 16-bitový datový typ. Písmeno "u" na začátku označí datový typ jako "bez znaménka".

U běžných (klasických) datových typů se pro typ “signed char” používá zkrácené označení “signed” viz předchozí tabulka, stejně tak int a long definují typy se znaménkem a char je bez znaménka, i když to není explicitně napsáno. Důvod pro tyto definice je AVR-GCC varianta kompilátoru, které se používá ve většině případů.

Datový typ “char” se bude používat pro řetězce, protože definice "uint8_t" může vést k nějaké nekompatibilitě se standardními knihovnami jazyka C a "char" je jasný a logický název pro znak nebo řetězec. Podrobnosti o tomto tématu probereme v kapitole RP6Library při textových výstupech přes sériové rozhraní.

Nyní prostě vezmeme na vědomí: pro znaky a řetězce budeme vždy používat typ "char", respektive uintN_t nebo intN_t pro celá čísla!

Aby bylo možné používat proměnné v programu, musíme je nejdříve deklarovat definicí datového typu, názvu a případně počáteční hodnoty této proměnné. Název musí začínat písmenem (včetně podtržítka "_"), a mohou obsahovat čísla. Nicméně konvence pro názvy proměnných vylučuje používání velkého množství speciálních znaků, tj. “äöüß#'[]23|*+-.,<>%&/(){}$§=´°?!^”.

V názvech proměnných se rozlišují malá a velká písmena, to znamená, že názvy aBc a abC jsou různé proměnné! Programátoři tradičně používají malá písmena – alespoň pro úvodní znaky proměnných.

Následující klíčová slova jsou již vyhrazená a NEMOHOU se používat jako názvy proměnných, názvy funkcí nebo jiných symbolů:

```
auto        default     float       long        sizeof      union
break       do          for         register    static      unsigned
case        double      goto        return      struct      void
char        else        if          short       switch      volatile
const       enum        int         signed      typedef     while
continue    extern
```

Dále se v jazyce C používají typy float a double určené pro čísla s plovoucí desetinnou čárkou, ale na malých typech mikroprocesorů AVR se použití těchto datových typů raději vyhneme. Desetinná čísla jsou velmi náročná na výpočetní čas i paměť a obvykle si perfektně vystačíme s celými čísly. Většina programů pro RP6 nebude vyžadovat čísla s plovoucí řádovou čárkou.

Deklarace proměnných je extremně jednoduchá a lze ji předvést na deklaraci proměnné s názvem x:

```
char x;
```

Po vlastní deklaraci je proměnná x na následujících řádcích platná a může se použít například k přiřazení hodnoty 10:

```
x = 10;
```

Alternativně můžeme přiřadit hodnotu do další proměnné y přímo v deklaraci:

```
char y = 53;
```

Základní aritmetické operace, které můžeme používat jsou obvykle:

```
signed char z;          // pamatujte prosím, že „signed” je před char!
z = x + y;        // z nabývá hodnotu z = x + y = 10 + 53 = 63
z = x – y;        // z nabývá hodnotu z = 10 – 53 = -43
z = 10 + 1 + 2 – 5;     // z = 8
z = 2 * x;        // z = 2 * 10 = 20
z = x / 2;        // z = 10 / 2 = 5
```

Programovací jazyk C také poskytuje nějaké užitečné zkratky:

```
z += 10;    // odpovídá: z = z + 10; to v tomto případě znamená z = 15
z *= 2;     // z = z * 2 = 30
z -= 6;     // z = z - 6 = 24
z /= 4;     // z = z / 4 = 8
```

```
z++;        // zkratka pro z = z + 1; to znamená, že z je nyní 9
z++;        // z = 10 // z++ se nazývá „inkrementace z”
z++;        // z = 11 ...
z--;        // z = 10 // z—- se nazývá „dekrementace z”
z--;        // z = 9
z--;        // z = 8 ...
```

V předchozím příkladu jsme použili datový typ “char”. V programech pro RP6 však budeme ve většině případů preferovat standardní datové typy.

V tomto příkladu je `int8_t x;`
shodné se `signed char x;`
a také `uint8_t x;`
je shodné s `unsigned char x; // respektive „char” se skutečně`
`// používá pouze pro znaky jako`
`// znaménkový typ, protože je to tak`
`// nastaveno v kompilátoru.`

4.4.5. Podmíněné příkazy

Podmíněné příkazy, které používají konstrukci "if-else", hrají důležitou roli v toku programu. Tyto výrazy nám umožňují zjistit, zda je podmínka je pravdivá či nepravdivá a rozhodnout, zda se konkrétní část programu provede nebo ne.

Malý příklad:

```
1  uint8_t x = 10;
2  if(x == 10)
3  {
4        writeString("x is equal to 10!\n");
5  }
```

Deklarace na řádku 1 definuje 8-bitovou proměnnou x přiřazuje jí hodnotu 10. Podmínka if na řádku 2 bude úspěšná, pokud je x rovno 10. Tato podmínka bude zřejmě vždy pravdivá a program zpracuje podmíněný blok příkazů. Tento blok vytvoří výstup “x je rovno 10!”. Pokud naopak proměnnou x inicializujeme hodnotou 231, program nevytvoří žádný výstup!

Obecně bude mít podmínka if vždy následující syntaxi:

```
if ( <podmínka X> )
  <blok příkazů Y>
else
  <blok příkazů Z>
```

Pomocí jednoduché angličtiny můžeme také číst: “pokud X pak dělej Y jinak dělej Z”.

Další příklad:

```
1  uint16_t myFavoriteVariable = 16447;
2
3  if( myFavoriteVariable < 16000) // If myFavoriteVariable < 16000
4  {                               // then:
5      writeString("myFavoriteVariable is less than 16000!\n");
6  }
7  else                            // else:
8  {
9      writeString("myFavoriteVariable is greater than or equal to 16000!\n");
10 }
```

Proměnná “myFavoriteVariable” je nastavena na hodnotu 16447, která způsobí výsledek ve formě výstupu “myFavoriteVariable is greater than or equal to 16000!“. V tomto příkladu je obsloužena i nepravdivá část podmíněného příkazu, kdy se provede větvení else.

Jak můžete vidět na názvu proměnné “myFavoriteVariable”, můžete pro název proměnné použít cokoliv až do délky, kterou povolují konvence jazyka C.

Pomocí konstrukce if – then – else lze vytvořit mnohem složitější podmíněné větvení programu:

```
1  if(x == 1) { writeString("x is 1!\n"); }
2  else if(x == 5) { writeString("x is 5!\n"); }
3  else if(x ==  244) { writeString("x is 244!\n"); }
4  else { writeString("x has a different value!\n");}
```

V podmíněných příkazech se mohou používat následující porovnávací operátory:

x == y	Logické porovnání pro shodu
x != y	Logické porovnání pro nerovnost
x < y	Logické porovnání pro “menší než“
x <= y	Logické porovnání pro “menší než nebo rovno“
x > y	Logické porovnání pro “větší než“
x >= y	Logické porovnání pro “větší než nebo rovno“

Programovací jazyk dále nabízí logická spojení:

x && y	pravda, pokud je pravdivý výraz x a zároveň je pravdivý výraz y
x || y	pravda, pokud je pravdivý výraz x nebo výraz y
!x	pravd, pokud je výraz x nepravdivý

Tyto struktury můžeme libovolně spojovat, kombinovat a sdružovat pomocí logických spojení a libovolného počtu dvojic závorek:

```
1  if( ((x != 0) && (!(x > 10))) || (y >= 200)) {
1        writeString("OK!\n");
2  }
```

Předchozí výpis podmíněného příkazu je pravdivý, pokud se x nerovná nule (x != 0) A ZÁROVEŇ x není větší než 10 (!(x > 10)) NEBO pokud je y větší nebo rovno 200 (y >= 200). Pokud je to nezbytné můžeme přidávat libovolný počet dalších podmínek, dokud výraz nesplní všechny potřeby programu.

4.4.6. Switch – Case

Často budeme muset porovnávat proměnnou s velkým množstvím různých hodnot a na základě výsledku porovnání rozhodnout o dalším provádění programu. Samozřejmě můžeme použít větší počet podmíněných příkazů if – then – else, ale programovací jazyk C nabízí mnohem elegantnější metodu pomocí konstrukce switch – else.

Malý příklad:

```
1  uint8_t x = 3;
2
3  switch(x)
4  {
5        case 1: writeString("x=1\n"); break;
6        case 2: writeString("x=2\n"); break;
7        case 3: writeString("x=3\n");    // At this point, "break" is missing,
8        case 4: writeString("Hello\n");  // causing the program to proceed
9        case 5: writeString("over\n");   // with the next two lines
10       case 6: writeString("there!\n"); break; // and stop here!
11       case 44: writeString("x=44\n"); break;
12       // The program will jump to this line if none of the previous
13       // conditions is met:
14       default : writeString("x is something else!\n"); break;
15 }
```

Tento úryvek programu funguje docela podobně jako předchozí příklad podmíněné struktury “ifelse – if –else – if – else...”, ale nyní místo ní použijeme větvení case. Je zde jeden základní rozdíl – pokud je jedna z podmínek pravdivá, provedou se všechny následující případy větvení case. Pokud nechcete takové chování – jednoduše přidáte instrukci “break” a v tomto místě se opustí konstrukce switch – case.

Výstup předchozího příkladu by měl vypadat takto (pří výchozí hodnotě x = 3):

```
x=3
Hello
over
there!
```

Nastavením x = 1 bude výsledný výstup “x=1\n” a při x = 5 bude výsledek:

```
over
there!
```

Nyní jste již možná pochopili, že instrukce “break” ukončí větvení case, program brouzdá přes následující příkazy, dokud se neobjeví konec konstrukce switch nebo další “break”.

Pokud nastavíme hodnotu x = 7, nebude žádné větvení pravdivé. Program nyní provede větvení “default”, které má za výsledek výstup: "`The value of x is something else!\n`".

Textový výstup je samozřejmě pouhý příklad, reálné programy však mohou tyto konstrukce používat pro vytvoření řady odlišných pohybů robota. Několik ukázkových programů na použití konstrukce switch – case je uvedeno při realizaci konečných stavových automatů k implementaci jednoduchého chování základní jednotky robotu.

4.4.7. Cykly

Pokud se mají operace provádět opakovaně, několikrát za sebou, potřebujeme k tomu cykly.

Nechte si na následujícím příkladu předvést základní princip:

```
1  uint8_t i = 0;
2  while(i < 10)                // as long as i is less than 10...
3  {                            // ... repeat the following code:
4        writeString("i=");     // output "i=",
5        writeInteger(i, DEC);  // output the "DECimal" value of i and ...
6        writeChar('\n');       // ... a line-break.
7        i++;                   // increment i.
8  }
```

Úryvek programu očividně obsahuje cyklus s podmínkou “while”, který generuje posloupnost: “i=0\n”, “i=1\n”, “i=2\n”, ... “i=9\n”. Blok příkazů uzavřený mezi složenými závorkami, který následuje za podmínkou “while(i < 10)” v hlavičce cyklu, se bude opakovat tak dlouho, dokud bude podmínka pravdivá. Obyčejnou angličtinou můžeme zápis číst jako: „Opakuj následující blok tak dlouho, dokud je i menší než 10“ . Jelikož je počáteční hodnota i = 0 a k inkrementaci dochází v každém průchodu cyklem, provede program tělo cyklu 10 krát a výstupní číslo bude od 0 do 9. V hlavičce cyklu můžeme používat podmíněné výrazy stejně jako v podmínce if.

Místo cyklu while můžeme použít cyklus “for”, který poskytuje stejnou funkčnost, ale nabízí rozšířené možnosti definice v hlavičce cyklu.

Ukázkový fragment programu může ilustrovat cyklus for:

```
1  uint8_t i;  // we will not initialize i here, but in the loop-header!
2  for(i = 0; i < 10; i++)
3  {
4        writeString("i=");
5        writeInteger(i, DEC);
6        writeChar('\n');
7  }
```

Tento cyklus for bude generovat identický výstup jako předchozí cyklus while. V hlavičce cyklu můžeme implementovat několik dalších věcí.
Základní cyklus for má následující strukturu:

```
for ( <inicializace řídící proměnné> ; <ukončovací podmínka> ;
      <modifikace řídící proměnné> )
{
  <blok příkazů>
}
```

Při práci s mikroprocesory budete často potřebovat nekonečné cykly, které se virtuálně mohou opakovat stále dokola. Ve skutečnosti většina mikroprocesorových programů obsahuje pouze jediný nekonečný cyklus – který uvede program pomocí dobře známých stavů pro ukončení pravidelného průběhu programu nebo neustále provádí činnost, dokud se zařízení nevypne.

Nekonečná smyčka se jednoduše vytvoří pomocí cyklu while nebo for:

```
while(true) { }
```

nebo

```
for(;;) { }
```

V obou případech se bude blok příkazů provádět “neustále“ (respektive dokud mikroprocesor nepřijme nějaký externí signál reset nebo program neukončí smyčku zpracováním příkazu “break”).

Pro úplnost přehled dokončíme popisem cyklu “do – while”, který se může považovat za alternativu standardního cyklu “while”. Na rozdíl od cyklu “while“, provede cyklus “do – while” na začátku blok příkazů i když není splněna podmínka cyklu.

Struktura tohoto cyklu je následující:

```
do
{
  <blok příkazů>
}
while(<podmínka>);
```

Nezapomeňte, prosím na ukončovací středník! (Standardní cyklus while se samozřejmě středníkem neukončuje!)

4.4.8. Funkce

Funkce jsou klíčovým prvkem programovacích jazyků. V předchozích kapitolách jsme již poznali a dokonce i používali funkce, například “writeString”, “writeInteger” a samozřejmě hlavní funkci – main.

Funkce jsou extrémně užitečné při používání stejných programových posloupností na několika místech programu – třeba funkce pro výstup textu, které jsme používali v předchozích kapitolách. Kopírování stejného kódu programu do všech potřebných míst může být velmi nepraktické. Tímto způsobem navíc zbytečně zabíráme větší prostor paměti programu. Použití jedné samostatné funkce nám umožní modifikovat programové moduly na jediném centrálním místě místo modifikace velkého množství kopií. Používání funkcí zjednoduší tok programu a pomáhá udržet přehledný zdrojový text.

Proto jazyk C umožňuje kombinovat programové posloupnosti do funkcí, které vždy musí mít následující strukturu:

```
<Návratový typ> <Název funkce> (<Parametr 1>, <Parametr 2>, ... <Parametr n>)
{
  <Programová posloupnost>
}
```

Princip si vysvětlíme na malém příkladu se dvěma jednoduchými funkcemi a již známou hlavní funkcí:

```
8  void someLittleFunction(void)
9  {
10       writeString("[Function 1]\n");
11 }
12
13 void someOtherFunction(void)
14 {
15       writeString("[Function 2 - something different]\n");
16 }
17
18 int main(void)
19 {
20       // Always start an RP6-program by calling this function!
21       initRobotBase();
22
23       // A few function calls:
24       someLittleFunction();
25       someOtherFunction();
26       someLittleFunction();
27       someOtherFunction();
28       someOtherFunction();
29       return 0;
   }
```

Program zobrazí na výstupním zařízení následující text:

```
[Funkce 1]
[Funkce 2 – něco jiného]
[Funkce 1]
[Funkce 2 – něco jiného]
[Funkce 2 – něco jiného]
```

Hlavní funkce obsluhuje vstupní bod a libovolný program v jazyku C bude začínat voláním této funkce. Proto MUSÍ každý program jazyka C poskytovat hlavní funkci “main“.

V předchozím příkladu hlavní funkce začínala voláním funkce initRobotBase z knihovny RP6Library, která inicializuje hardwarové prostředky mikroprocesoru. Základem funkce initRobotBase je struktura velmi podobná dvěma funkcím v tomto příkladu. V hlavní funkci programu se několikrát volají dvě předem definované funkce a provádí se programový kód těchto funkcí.

Vedle definice funkcí popsané v předchozím příkladu, můžeme také používat parametry a návratové hodnoty. V příkladu se používá jako parametr i návratová hodnota výraz “void”, který znamená, že se zde nepoužívá žádný parametr nebo návratová hodnota. Parametr “void” vždy označuje funkce bez návratových hodnot respektive bez parametrů.

Ve funkci můžeme definovat velký počet parametrů a jednotlivé parametry se oddělují čárkou.

Příklad demonstruje základní princip:

```
1  void outputSomething(uint8_t something)
2  {
3        writeString("[The following value was passed to this function: ");
4        writeInteger(something, DEC);
5        writeString("]\n");
6  }
7
8  uint8_t calculate(uint8_t param1, uint8_t param2)
9  {
10       writeString("[CALC]\n");
11       return (param1 + param2);
12 }
13
14 int main(void)
15 {
16       initRobotBase();
17
18       // Now execute a few function calls with parameters:
19       outputSomething(199);
20       outputSomething(10);
21       outputSomething(255);
22
23       uint8_t result = calculate(10, 30); // return value...
24       outputSomething(result);
25       return 0;
26 }
```

Výstup:

```
[Touto funkcí byla zpracována následující hodnota: 199]
[Touto funkcí byla zpracována následující hodnota: 10]
[Touto funkcí byla zpracována následující hodnota: 255]
[CALC]
[Touto funkcí byla zpracována následující hodnota: 40]
```

Knihovna RP6 Library poskytuje celou řadu funkcí. Krátký pohled na kód několika modulů s ukázkových programů objasní zkladní principy vývoje programů s podporou těchto funkcí.

4.4.9. Pole, řetězce, ukazatele...

Na vysvětlení čeká ještě velké množství dalších zajímavostí programovacího jazyka C, ale podrobnosti necháme na studium dostupné literatury!

Většina ukázkových programů je srozumitelná bez dalšího studování. Ve zbývající části tohoto stručného kurzu popíšeme pouze krátký přehled několika příkladů a konceptů, které samozřejmě nejsou příliš podrobné.

Nejprve probereme pole. Pole vám umožňuje uložit předem definovaný počet prvků se stejným datovým typem. Následující vzorek pole se může použít k uložení 10 byte:

```
uint8_t myFavouriteArray[10];
```

Na jediném řádku jsme deklarovali 10 proměnných se stejným datovým typem, které mohou být adresovány indexem:

```
myFavouriteArray[0] = 8;
myFavouriteArray[2] = 234;
myFavouriteArray[9] = 45;
```

Každý prvek pole může být chápán jako standardní proměnná.

Pozor: index vždy začíná od 0 a deklarace pole obsahuje n prvků – z toho plyne rozsah indexu od 0 do n-1! Ukázkové pole poskytuje 10 prvků s indexy od 0 do 9.

Pole jsou velmi užitečná pro ukládání většího počtu proměnných se stejným datovým typem a může se s nimi snadno manipulovat v cyklech:

```
uint8_t i;
for(i = 0; i < 10; i++)
  writeInteger(myFavouriteArray[i],DEC);
```

Předchozí úryvek kódu vytvoří výstup všech prvků pole (v tomto případě bez oddělovačů a odsazení řádků). Velmi podobný cyklus se může použít pro naplnění pole hodnotami.

V jazyku C jsou řetězce obslouženy velmi jednoduchým konceptem. Standardní řetězce budou tvořit ASCII znaky, které pro každý znak zabírají jeden byte. Nyní se v jazyku C řetězce jednoduše definují jako pole, která můžeme chápat jako pole znaků. Tento koncept umožňuje definovat a ukládat do paměti předem sestavený řetězec "abcdefghijklmno":

```
uint8_t aSetOfCharacters[16] = "abcdefghijklmno";
```

Dříve probrané ukázkové programy vždy obsahovaly několik funkcí UART pro výstup řetězců přes sériové rozhraní. Základem těchto řetězců jsou pole. Nepracuje se však s celým polem. Tyto funkce se budou pouze vázat na adresu prvního prvku v poli. Proměnná, která obsahuje adresu tohoto prvního prvku, se nazývá “ukazatel”. Můžeme vytvořit ukazatel na požadovaný prvek pole zapsáním &MyFavouriteArray[x], kde x odkazuje na indexovaný prvek. Několik takových výrazů můžeme najít v ukázkových programech, tj.:

```
uint8_t * PointerToAnElement = &aCharacterString[4];
```

Na této úrovni však nemusíme zcela zvládnout tento koncept, přesto porozumíte většině ukázkových programů nebo psaní vlastních programu.

4.4.10. Tok programu a přerušení

Jak bylo probráno dříve, program bude zpracovávat základní instrukce postupně shora dolů. Vedle tohoto standardního chování existuje řízení běhu programu pomocí podmíněných skoků, cyklů a funkcí.

Mimo těchto obvyklých prostředků existují takzvaná “přerušení”. Tato přerušení může generovat několik hardwarových modulů (časovač, TWI, UART, externí přerušení atd.) a vyžadují okamžitou pozornost od mikroprocesoru. Správnou odezvu mikroprocesor zajistí opuštěním normálního běhu programu a skokem na takzvanou rutinu pro obsluhu přerušení (ISR). Tato reakce na přerušení je virtuálně nezávislá na toku programu. Nemějte obavy! Všechny potřebné ISR byly připraveny do knihovny RP6Library a postarají se o všechny nezbytné události. Nemusíte implementovat vlastní obsluhu přerušení. V této části rychle probereme všechny základní věci, které musíte znát o těchto speciálních funkcích přerušení.

Základní struktura ISR je následující:

```
ISR ( <vektor přerušení> )
{
  <blok příkazů>
}
```

tj. pro levý enkodér, připojený na externí přerušení 0:

```
ISR (INT0_vect)
{
  // Na každé hraně signálu provedeme zvýšení dvou čítačů:
  mleft_dist++;         // ujetá vzdálenost
  mleft_counter++;      // měření rychlosti
}
```

Tyto ISR obsluhy přerušení nemůžeme volat přímo! Volání obslužné ISR se provádí automaticky a může se vyskytnout kdykoliv! Kdykoliv a v libovolné části programu může volání obsluhy přerušení zastavit normální běh programu (s výjimkou vlastní rutiny pro obsluhu přerušení nebo v případě zakázaného přerušení). Při události přerušení se provádí příslušná funkce ISR a po skončení ISR bude pokračovat provádění programu od místa, kde byl normální běh opuštěn. Toto chování vyžaduje zařazení všech časově kritických částí programu do funkcí ISR (nebo zákaz přerušení na krátký okamžik). Pokud se zpoždění nezpracuje jako událost přerušení, mohou se prodloužit periody zpoždění, které mikroprocesor počítá pomocí instrukčních cyklů.

Knihovna RP6Library používá přerušení ke generování modulovaného signálu 36kHz pro infračervené snímače a IR komunikaci. Dále se používají pro dekódování RC5, funkce časování a zpoždění, měření z enkodérů, obsluhu modulu TWI (sběrnice I^2C) a několik dalších aplikací.

4.4.11. Preprocesor jazyka C

V této kapitole stručně probereme preprocesor jazyka C, který se již používal v předchozích ukázkových programech na řádku: `#include "RP6RobotBaseLib.h"`!

Preprocesor vyhodnotí tento příkaz ještě před spuštěním kompilačního procesu GCC. Příkazová řádka `#include "file"` vloží obsah specifikovaného souboru do vkládaného místa. Náš ukázkový program vloží soubor RP6BaseLibrary.h, který poskytuje definice všech uživatelsky dostupných funkcí a řídí zpracování překladu. Možnosti preprocesoru mají však několik dalších voleb a umožňují definici konstant (které mohou obshovat pevné hodnoty systému):

```
#define THIS_IS_A_CONSTANT 123
```

Tento výraz definuje textovou konstantu “THIS_IS_A_CONSTANT“ s hodnotou “123”. Preprocesor jednoduše nahradí všechny odkazy definovanou hodnotou. Konstanty mohou být chápány jako náhrada textu! V následujícím výrazu:

```
writeInteger(THIS_IS_A_CONSTANT,DEC);
```

bude “THIS_IS_A_CONSTANT” nahrazen hodnotou “123” a shodně:

```
writeInteger(123,DEC);
```

(mimochodem: parametr “DEC” ve funkci writeInteger je jen další konstanta – v tomto případě definiční konstanta číselného základu hodnoty 10 – pro desítkový číselný systém).

Preprocesor také ovládá jednoduché podmínky if:

```
1  #define DEBUG
2
3  void someFunction(void)
4  {
5        // Now execute something...
6        #ifdef DEBUG
7           writeString_P("someFunction has been executed!");
8        #endif
9  }
```

Textový výstup se vytvoří pouze při definici “DEBUG” (nemusíte do ní přiřazovat hodnotu – stačí jednoduše definovat DEBUG). To je užitečné při aktivaci několika textových výstupů pro ladění během vývoje programu, zatímco při normálním překladu můžete odstranit všechny pomocné textové výstupy úpravou jediné řádky. Bez definice DEBUG v předchozím ukázkovém programu zabrání preprocesor průchodu obsahu programového řádku 7 do kompilátoru.

Knihovna RP6Library také nabízí makra, které se definují pomocí výrazu `#define`. Makra umožňují zpracování parametrů podobně jako funkce. Následující příklad ukazuje typickou definici makra:

```
#define setStopwatch1(VALUE) stopwatches.watch1 = (VALUE)
```

Tato definice vám umožňuje volat makro stejně jako normální funkci (tj. `setStopwatch1(100);`).

Důležitý detail: za definicemi preprocesoru se obvykle nevkládá středník!

4.5. Makefile

Nástroj “Make“ zjednodušuje proces kompilace automatickým zpracováním velkého počtu činností, nutných pro kompilaci programů v jazyce C. Automatizované zpracování je definováno v takzvaném “Makefile“, který obsahuje všechny nezbytné posloupnosti příkazů a informace pro zpracování kompilace projektu. Tyto makefile poskytujeme pro všechny ukázkové projekty RP6, ale samozřejmě můžete vytvořit libovolné makefile pro vlastní projekty. Nebudeme podrobně probírat strukturu souborů makefile, můžete to nastudovat v příslušné příručce. U všech projektů RP6 se můžete soustředit na následující čtyři přístupy. Další aspekty nejsou pro začátečníky důležité a můžete je ignorovat.

```
TARGET = programName

RP6_LIB_PATH = ../../RP6lib

RP6_LIB_PATH_OTHERS = $(RP6_LIB_PATH)/RP6base $(RP6_LIB_PATH)/RP6common

SRC += $(RP6_LIB_PATH)/RP6base/RP6RobotBaseLib.c
SRC += $(RP6_LIB_PATH)/RP6common/RP6uart.c
SRC += $(RP6_LIB_PATH)/RP6common/RP6I2CslaveTWI.c
```

Naše makefile obsahují mezi příkazovými řádky komentáře. Komentáře v makefile začínají “#” a budou během zpracování ignorovány.

Ukázkové projekty RP6 poskytují přizpůsobené makefile připravené pro použití a nebudete je muset měnit, dokud neplánujete vkládání nových zdrojových souborů jazyka C do struktury projektu nebo začnete měnit názvy souborů.

Tvorba makefile začíná specifikací názvu souboru programu, který obsahuje hlavní funkci v řádku “TARGET”. Název musíte specifikovat bez přípony, nikdy sem prosím nepřidávejte příponu “.c”! Řadu dalších přípon budete muset bohužel specifikovat a nejlepší bude prostudovat detaily komentářů u existujících příkladů makefile!

Druhý vstup “RP6_LIB_PATH” umožňuje specifikovat cestu ke knihovním souborům RP6Library. Definujte, prosím úplnou cestu tj. například “../RP6lib” or “../../RP6lib” (in which “../” to znamená “o jednu adresářovou úroveň výš”).

Třetí vstup RP6_LIB_PATH_OTHERS se používá ke specifikaci všech dalších adresářů. Knihovna RP6Library je rozdělena do řady dílčích adresářů a ve svém projektu musíte definovat cesty ke všem potřebným adresářům.

Na závěr definujete všechny zdrojové soubory jazyka C ve vstupu “SRC” (nevkládejte žádné hlavičkové soubory s příponou “.h”, které se automaticky vyhledají ve všech specifikovaných adresářích!), které se používají pod souborem s hlavní funkcí. Dále musíte specifikovat všechny použité knihovní soubory RP6Library.

Co znamená výraz $(RP6_LIB_PATH)? Dobrá, je to způsob jak používat proměnné v makefiles! Již jsme definovali “proměnnou” s názvem RP6_LIB_PATH. Jakmile je proměnná deklarována, může se úspěšně použít obsah proměnné pomocí zápisu $(<proměnná>) v textu makefile. Tato užitečná vlastnost výrazně zmírní složitost zápisu makefile...

Obvykle nemusíte v RP6 makefile nic měnit. Když se chcete podívat na další informace o této problematice můžete prostudovat příručku:

http://www.gnu.org/software/make/manual/

4.6. Knihovna funkcí RP6 (RP6Library)

Knihovna funkcí RP6 (zkráceně RP6Library nebo RP6Lib) poskytuje velké množství lowlevel funkcí pro ovládání hardware RP6. S touto knihovnou funkcí se obvykle nemusíte zabývat všemi detailními parametry hardware robotu a mikroprocesoru. Abyste mohli programovat robot RP6, nemusíte samozřejmě přečíst 300 stránek katalogového popisu mikroprocesoru ATmega32. Některé důležité části katalogového listu však prostudovat musíte, pro správné pochopení knihovny RP6Library.

Ve skutečnosti jsme se úmyslně vyhnuli perfektnímu vyladění všech funkcí RP6Library, abychom nechali nějakou práci také pro vás! Budete vyzváni k přidání několika funkcí a k optimalizaci existujících! Pokládejte knihovnu RP6Library jako dobrý výchozí bod, ale ne jako optimální řešení.

Tato kapitola probírá nejdůležitější funkce a ukazuje krátké příklady. Pokud vás zajímají další podrobnosti, můžete si přečíst komentáře v knihovních souborech a prostudovat funkce a jejich použití v příkladech.

4.6.1. Inicializace mikroprocesoru

```
void initRobotBase(void)
```

Hlavní funkční blok VŽDY začíná voláním této funkce! Inicializuje hardwarové moduly mikroprocesoru. Pokud program neprovede inicializaci, nemusí mikroprocesor fungovat správně. Částečně se hardwarové moduly inicializují bootloaderem, ale ne všechny.

Příklad:

```
1  #include "RP6RobotBaseLib.h"
2
3  int main(void)
4  {
5        initRobotBase(); // Initialization - ALWAYS CALL THIS FIRST!
6
7  // [...] Program code...
8
9        while(true);      // Infinite loop
10       return 0;
11 }
```

Principielně by měl program pro RP6 vypadat podobně. Nekonečná smyčka na řádku 9 funguje jako předpokládaný konec programu. Přeskočení nekonečné smyčky může vyvolat neočekávané chování programu!

Připomeňme si opět princip nekonečné smyčky: obvykle se bude nekonečná smyčka používat k provádění vlastního kódu programu. Takže odstraníte středník na řádku 9 a nahradíte ho blokem příkazů (uzavřeným ve složených závorkách). Na řádcích před hlavní funkcí můžete definovat vlastní funkce (v tomto případě na řádku 2) a pak lze tyto funkce volat z libovolného místa hlavní smyčky.

4.6.2. Funkce UART (sériové rozhraní)

V předchozím rychlokurzu jazyka C již bylo použito několik funkcí z knihovny RP6Library, například funkce UART. Tyto funkce umožňují přenášení textových zpráv přes sériové rozhraní robotu do a z PC (nebo jiného mikroprocesoru).

4.6.2.1. Vysílání dat

```
void writeChar(char ch)
```

Tato funkce vysílá jeden 8-bitový ASCII znak přes sériové rozhraní.

Použití je jednoduché:

```
writeChar('A');
writeChar('B');
writeChar('C');
```

To by mělo vytvořit výstup “ABC”. Funkce může také přenášet přímo ASCII kódy tj.:

```
writeChar(65);
writeChar(66);
writeChar(67);
```

Výsledkem by měl být výstup “ABC”, protože libovolný ASCII znak může být reprezentován číslem. Dekadické číslo 65 odpovídá znaku 'A'. Speciální komunikační software může znaky interpretovat přímo v binární hodnotě.

Často budete potřebovat vyslat něco takového:

```
writeChar('\n');
```

což na terminálovém programu vyvolá nový řádek.

```
void writeString(char *string)a writeString_P(STRING)
```

Tyto funkce jsou důležité pro ladění programů, protože umožňují vysílání libovolné textové zprávy na PC. Užitečné mohou být samozřejmě také pro přenos dat.

Nyní si vysvětlíme rozdíl mezi writeString a writeString_P. S funkcí writeString_P se bude pracovat pouze, pokud je text uložen v paměti FLASH-ROM (paměť programu) a funkce bude na výstup postupně vyčítat řetězec z FLASH-ROM. Naproti tomu funkce writeString bude brát řetězec uložený do paměti RAM. Pamatujte, že mikroprocesor má relativně malou paměť RAM (2 kB). Takže, když chcete na výstup poslat pevný textový řetězec, preferujte použití funkce writeString_P. Samozřejmě pro přenos dynamických dat, která jsou nějak dostupná v RAM, se musí použít writeString.

Používání příslušné funkce je stejně jednoduché jako používání writeChar (pozor místo apostrofu se používají uvozovky):

```
writeString("ABCDEFG");
```

vytvoří výstup “ABCDEFG”, ale jak bylo uvedeno výše, řetězec se nejdříve uloží do paměti POM a před vysíláním se přesune do RAM.

```
writeString_P("ABCDEFG");
```

vytvoří stejný výstup “ABCDEFG”, ale text nebude zabírat prostor v paměti RAM!

```
void writeStringLength(char *data, uint8_t length, uint8_t offset);
```

Tuto funkci můžete použít kdykoliv budete potřebovat výstup textu s definovanou délkou nebo posunem.

Například:

```
writeStringLength("A**BCD**EFG", 3, 1);
```

výstup: “BCD”

```
writeStringLength("ABCD**EF**G", 2, 4);
```

výstup: “EF”

Tato funkce však zabírá místo v paměti RAM pro uložení řetězce a je určena pro zpracování dynamických textů. Tuto funkci například používá writeIntegerLength.

```
void writeInteger(int16_t number, uint8_t base);
```

Tato velmi užitečná funkce vytvoří výstup celočíselných hodnot ve tvaru ASCII testu. Z předchozích příkladů si pamatujeme, že writeChar(65) předá na výstup znak 'A' místo čísla 65...

Proto potřebujeme funkci, které vytvoří výstup čísel ve tvaru textových řetězců.

Příklad:

```
writeInteger(139, DEC);
```

výstup: “139”

```
writeInteger(25532, DEC);
```

výstup: “25532”

Funkce umožňuje výstup čísel z celého rozsahu 16-bitových celých čísel v rozsahu od -32768 do 32767. Pokud však předpokládáte použití čísel, které překračují tyto meze, musíte funkci modifikovat nebo vytvořit zvláštní funkci.

Nyní se můžeme zamyslet, proč se používá druhý parametr “DEC”. Odpověď je skutečně jednoduchá: tento parametr řídí výstupní formát čísla. Místo DEC - dekadický (základ 10) můžeme samozřejmě používat několik alternativních výstupních formátů například binární (BIN, základ 2), osmičkový (OCT, základ 8) nebo hexadecimální (HEX, základ 16).

Několik příkladů:

```
writeInteger(255, DEC);
```

výstup: “255”

```
writeInteger(255, HEX);
```

výstup: “FF”

```
writeInteger(255, OCT);
```

výstup: “377”

```
writeInteger(255, BIN);
```

výstup: “11111111”

Tyto funkce jsou pro řadu aplikací neobyčejně užitečné. Speciálně pro výstup celých čísel ve formátu HEX nebo BIN, protože tyto formáty umožňují přímé sledování nastavení jednotlivých bitů.

```
void writeIntegerLength(uint16_t number, uint8_t base, uint8_t length);
```

Tato funkce je jiná varianta writeInteger, která umožňuje definovat počet (délku) zobrazovaných číslic, Když je délka čísla pod definovaným počtem, funkce přidá úvodní nuly. Když délka přesahuje definovaný počet, funkce zobrazí pouze definovaný počet číslic.

Jako obvykle ukážeme chování funkce na několika příkladech:

```
writeIntegerLength(2340, DEC, 5);
```

výstup: “02340”

```
writeIntegerLength(2340, DEC, 8);
```

výstup: “00002340”

```
writeIntegerLength(2340, DEC, 2);
```

výstup: “40”

```
writeIntegerLength(254, BIN, 12);
```

výstup: “000011111110”

4.6.2.2. Příjem dat

Příjem dat přes sériové rozhraní je zcela založeno na přerušení. Přijímaná data se automaticky na pozadí ukládají do takzvaného kruhového registru.

Jednoduchý příjem byte/znaku lze z registru číst pomocí funkce:

```
char readChar(void)
```

Funkce vrací následující volný znak v bufferu a po přečtení ho z bufferu zruší.

Pokud je kruhový registr prázdný, funkce vrátí 0. Velikost bufferu můžete kontrolovat pomocí funkce:

```
uint8_t getBufferLength(void)
```

před voláním funkce readChar, jinak nemůžete rozhodnout, zda je 0 skutečnou hodnotou dat nebo ne.

Skupina znaků se může postupně číst z bufferu pomocí

```
uint8_t readChars(char *buf, uint8_t numberOfChars)
```

V parametru této funkce musíte předat ukazatel na pole a počet přijímaných znaků. Funkce vrací skutečný počet znaků zapsaných do pole. To je užitečné, pokud buffer obsahuje méně znaků, než je specifikováno parametrem numberOfChars.

Když je buffer úplně plný, nově přijatá data NEPŘEPÍŠOU data v bufferu. Místo toho se nastaví příznak (UART_BUFFER_OVERFLOW) ve stavové proměnné (uart_status), který signalizuje přetečení bufferu.

Programy byste měli psát tak, aby se tato situace nenastala. Přetečení bufferu se obvykle objeví, když je příliš vysoká rychlost přenosu dat nebo dojde k dlouhodobému zaneprázdnění programu a ten není schopen přečíst data z bufferu. Můžete se vyhnout používání dlouhých zpoždění mSleep. Pokud potřebujete, můžete zvětšit velikost kruhového registru. Velikost bufferu je nastavena na 32 znaků. V souboru RP6uart.h, můžete změnit definici UART_RECEIVE_BUFFER_SIZE.

Rozsáhlejší ukázkový program můžete najít na CD-ROM (Example_02_UART_02).

4.6.3. Funkce zpoždění a časovače

Mikroprocesorové programy mají často vytvořit zpoždění nebo musí určitý čas počkat než se provede nějaká akce.

K tomuto účelu nabízí funkce také knihovna RP6Library. Pro zajištění relativně přesného zpoždění funkce využívají jeden ze tří časovačů ATmega32, který je nezávislý na běhu programu nebo přerušení, které by mohlo narušit zpožďovací rutiny.

Při používání těchto funkcí musíte být opatrní! Použití těchto funkcí během automatické regulace rychlosti a ACS (bude vysvětleno později) může způsobit problémy. Pokud potřebujete používat automatickou regulaci rychlosti a ACS, používejte pouze velmi krátká zpoždění, menší než 10 milisekund. Místo blokování programu pomocí zpoždění, můžete upřednostnit funkci “stopky”, která bude probrána v následující části.

```
void sleep(uint8_t time)
```

Tato funkce zastaví normální provádění programu na definovanou časovou periodu. Zpoždění se specifikuje s rozlišením 100µs (100µs = 0,1ms = 0.0001s, což je pro vnímání lidskými smysly příliš krátký čas...). Použití proměnné s velikostí 8 bitů umožňuje nastavení zpoždění až 25500 µs = 25,5 ms. Zatímco normální program “spí”, přerušení se stále zpracovává. Tato funkce pouze zpozdí zpracování normálního programu. Jak bylo uvedeno dříve, využívá funkce hardwarový časovač a příliš neovlivňuje další události přerušení.

Příklady:

```
sleep(1);         // zpozdeni 100µs
sleep(10);        // zpozdeni 1ms
sleep(100);       // zpozdeni 10ms
sleep(255);       // zpozdeni 25.5ms
```

```
void mSleep(uint16_t time)
```

Pokud však potřebujete delší zpoždění, můžete preferovat funkci mSleep, která umožňuje definovat periodu zpoždění v milisekundách. Maximální perioda zpoždění je 65535 ms, neboli 65.5 sekund.

Příklady:

```
mSleep(1);        // zpozdeni 1ms
mSleep(100);      // zpozdeni 100ms
mSleep(1000);     // zpozdeni 1000ms = 1s
mSleep(10000);    // zpozdeni 10 seconds
mSleep(65535);    // zpozdeni 65.5
```

Stopky

Problém těchto standardních zpožďovacích funkcí je v tom, že zcela zastaví běh normálního programu. To lze akceptovat pouze, když určitá část programu musí čekat určitou časovou periodu, zatímco u jiné části se předpokládá, že pokračují ve zpracování úkolu...

Jedna z hlavních výhod používání hardwarových časovačů je nezávislost na běhu normálního programu. Pomocí těchto časovačů implementuje RP6Library univerzální takzvané “Stopky”. Autor si vybral tento neobvyklý název pro podobnost se skutečnými stopkami.

Tyto “Stopky” budou jednoduše zvyšovat počet zpracování. obvykle se funkce časovače přizpůsobuje potřebám jednotlivých programů, ale stopky umožňují “současné” zpracování několika úloh – aspoň z vnějšího pohledu na mikroprocesor.

RP6 poskytuje osm 16-bitových stopek (Stopwatch1 až Stopwatch8), které lze spustit, zastavit, nastavit a číst. Stejně jako u funkce mSleep můžeme počítat s rozlišením jedné milisekundy, které znamená zvyšování čítače v intervalech po 1 ms. Tato metoda se nedá použít pro velmi kritické časování, protože kontrola obsahu čítače není příliš přesná.

Následující příklad předvádí použití stopek:

```
1  #include "RP6RobotBaseLib.h"
2
3  int main(void)
4  {
5        initRobotBase(); // Initialize the micro-controller
6        writeString_P("\nRP6 Demo Program for Stopwatches\n");
7        writeString_P("_____________________________________\n\n");
8
9        startStopwatch1(); // Start Stopwatch1
10       startStopwatch2(); // Start Stopwatch2
11
12       uint8_t counter = 0;
13       uint8_t runningLight = 1;
14
15       // Main loop:
16       while(true)
17       {
18             // A small LED running light:
19             if(getStopwatch1() > 100)  // Did 100ms (= 0.1s) pass by?
20             {
21                 setLEDs(runningLight); // Set the LEDs
22                 runningLight <<= 1;    // Next LED (shift operation)
23                 if(runningLight > 32)  // Last LED?
24                     runningLight = 1;  // Yes, restart with LED1!
25                 setStopwatch1(0);      // Reset Stopwatch1 to zero
26             }
27
28             // Output a counter level in the terminal:
29             if(getStopwatch2() > 1000) // Did 1000ms (= 1s) pass by?
30             {
31                 writeString_P("CNT:");
32                 writeInteger(counter, DEC); // Output counter level
33                 writeChar('\n');
34                 counter++;          // Increment the counter
35                 setStopwatch2(0);   // Reset Stopwatch2 to zero
36             }
37       }
38       return 0;
39 }
```

Program je skutečně jednoduchý. Každou sekundu se pošle obsah čítače přes sériové rozhraní a zvýší čítač (řádky 29 až 36). Zároveň se provádí zpracování jednoduchého programu běžícího světla na LED (řádky 19 až 26) s intervalem obnovy 100 ms.

Zde se používají Stopwatch1 a Stopwatch2, které začínají na řádku 9 respektive 10. Pak se rozběhnou čítače ve stopkách. Nekonečná smyčka (na řádcích 16 až 37) trvale kontroluje, zda stopky nepřekročily definovanou úroveň. Podmínka na řádku 19 například řídí běhání světla a kontroluje, zda na stopkách uběhl od posledního vynulování čas 100 ms. Pokud je tato podmínka splněna, rozsvítí se následující LED, vynuluje se čítač (řádek 25) a čeká se dalších 100ms. Stejný postup se používá u druhého časovače, který naopak kontroluje intervaly 1000ms, respektive 1 sekundy.

Na CD můžete najít mírně rozšířenou verzi tohoto programu Je to jen malý příklad, ale můžete sestavit mnohem složitější systémy se stopkami, které spolehlivě spouští a zastavují různé události...

Ukázkový program na CD také obsahuje běžící světlo a čítač (v programu jsou celkem 3 čítače...), ale je zde implementovaná samostatná funkce, která se bude volat z nekonečné smyčky.

Rozdělení kódu programu do samostatných funkcí pomáhá udržet přehlednost složitých programů a také je jednoduší opakované používání programových modulů pomocí Kopíruj a Vlož. Takže například kód běžícího světla se může, bez větších změn, použít v dalších programech...

Pro řízení stopek bylo implementováno několik maker.

```
startStopwatchX()
```

Spustí stopky X. Příkaz neprovede nulování stopek a ty pokračují ve zvyšování hodnoty posledního obsahu čítače.

Příklady:

```
startStopwatch1();
startStopwatch2();
```

```
stopStopwatchX()
```

Zastaví stopky X.

Příklady:

```
stopStopwatch2();
stopStopwatch1();
```

```
uint8_t isStopwatchXRunning()
```

Vrací informaci, zda stopky X běží.

Příklad:

```
if(!isStopwatch2Running) {
  // Stopky byly zrusene, zde se to muze napravit...
}
```

```
setStopwatchX(uint16_t preset)
```

Toto makro nastavuje čítač stopek X na zadanou hodnotu.

Příklady:

```
setStopwatch1(2324);
setStopwatch2(0);
setStopwatch3(2);
setStopwatch4(43456);
```

```
getStopwatchX()
```

Makro vrací obsah čítače stopek X.

Příklady:

```
if(getStopwatch2() > 1000) { ... }
if(getStopwatch6() > 12324) { ... }
```

4.6.4. Stavové LED a nárazníky

```
void setLEDs(uint8_t leds)
```

Tato funkce umožňuje ovládání 6 stavových LED. Místo obvyklých dekadických čísel se mohou jednoduše používat binární konstanty. Binární konstanty jsou uváděny ve tvaru: 0bxxxxxx. LED potřebují binární číslou pouze se 6 číslicemi.

Příklady:

```
setLEDs(0b000000);      // zhasne vsechny LED
setLEDs(0b000001);      // blikne StatusLED1 a vypne vsechny ostatni LED
setLEDs(0b000010);      // StatusLED2
setLEDs(0b000100);      // StatusLED3
setLEDs(0b001010);      // StatusLED4 a StatusLED2
setLEDs(0b010111);      // StatusLED5, StatusLED3, StatusLED2 a StatusLED1
setLEDs(0b100000);      // StatusLED6
```

Alternativní možnosti jsou:

```
statusLEDs.LED5 = TRUE;       // blikne LED5 v registru LED
statusLEDs.LED2 = FALSE;      // zhasne LED2 v registru LED
updateStatusLEDs();           // provede zmeny
```

Zde se blikne StatusLED5 a zhasne StatusLED2, ale nezmění se stav ostatních LED! Na rozdíl od jiných částí programu je toto řízení LED jednodušší.

Pozor: statusLEDs.LED5 = TRUE; NEROZSVÍTÍ přímo LED5! Tento příkaz pouze nastaví příslušný bit v proměnné. LED5 bude svítit po provedení funkce updateStatusLEDs();!

Dva vývody portu s LED se dále používají ke kontrole stavu nárazníků. V případě čtení stavu nárazníků mikroprocesor rychle přepne směr vývodů do vstupního režimu a zkontroluje, zda jsou sepnuté připojené mikrospínače. Pro kontrolu nárazníků poskytujeme dvě funkce. První funkce:

```
uint8_t getBumperLeft(void)
```

bude číst stav levého nárazníku, zatímco:

```
uint8_t getBumperRight(void)
```

bude číst spínač pravého nárazníku.

Mikroprocesor zpracuje tyto funkce velmi rychle, tak ani nepostřehnete, že LED zhasly, přestože je několik instrukčních cyklů Pin nastaven na vstup. Samozřejmě nemůžete tuto funkci volat opakovaně bez vložení několika milisekundového zpoždění.

Vývody portu s LED by měly být ovládány pouze pomocí předem definovaných funkcí! Porty s nárazníky jsou chráněny rezistory, ale pokud se vývody nastaví na výstup s nízkou úrovní a zároveň se sepne spínač nárazníku, poteče do portu větší proud. Takový proud je samozřejmě nežádoucí (obvody AVE mají třístavové výstupy – při nastavení do plovoucího stavu LED zhasnou).

Příklad:

```
if(getBumperLeft() && getBumperRight())  // oba narazniky...
  escape();       // zde definujte vlastni funkci tj. coufnuti a otoceni
else if(getBumperLeft())                        // levy...
  escapeLeft();  // opet couvani a otaceni doprava
else if(getBumperRight())                       // pravy...
  escapeRight();   // opet couvani a otaceni doleva
mSleep(50);      // rychlost kontroly narazniku 20 krat za sekundu (20Hz)...
```

Stisknutí nárazníku rozsvítí LED 6 a 3. Je to záměrné a nelze odstranit. Nárazníky se však neaktivují příliš často, takže nás to příliš neruší.

Na číslicové výstupy se čtyřmi zbývajícími LED můžeme připojit další snímače. Přes tranzistorové pole se také mohou spínat větší zátěže, jako jsou LED nebo motory. To je pouze krátký výčet věcí, které mohou být, pomocí příslušných funkcí, ovládány pomocí těchto čtyř portů, ale pamatujte na existující funkce pro LED a nárazníky.

***Pozor**: porty mikroprocesoru vždy chraňte rezistorem pro omezení proudu s hodnotou minimálně 470Ω, který vložíte mezi vývod portu a snímač nebo spínač!*

Během bootovací fáze umožňuje RP6 zhasnout LED. To je užitečné pro potlačení aktivity portu během probíhajícího bootování, zejména pokud je na port s LED připojeno další zařízení.

První byte vnitřní EEPROM (adresa 0) je vyhrazen pro ovládání režimu LED. Nepoužívejte tento byte pro vlastní programy (přepsání tohoto byte nic nepoškodí, ale může vás znepokojit, že se po zapnutí RP6 dlouho nerozsvítí LED...).

Na robotu RP6 existuje řada věcí, které se musí neustále sledovat, pokud mají fungovat správně. Například ACS musí vysílat IR impulsy v přesných intervalech a kontrolovat příjem. K tomu nemůžeme použít automatickou funkci přerušení, protože obsluha přerušení musí být co nejrychlejší. Nezbývá než opakované volání několika funkcí z hlavního programu. V dobře navrženém programu se bude zdát, že tyto úlohy běží na pozadí.

V této kapitole probereme všechny funkce pro ACS a podobné systémy a poskytneme další podrobnosti. Předpokládáme však, že nejjednodušší bude uvést několik podrobností, které usnadní zvládnutí funkce nárazníků a jejich implementaci v programu.

Nyní, když řešíme úlohy na pozadí, mohou vedle velkých věcí běžet nějaké další (menší) úlohy – například sledování nárazníků. Je to jednoduchá a rychlá úloha, kterou můžete provádět v hlavní smyčce. Při automatické kontrole nárazníků můžete v hlavní smyčce opakovaně volat tuto funkci:

```
void task_Bumpers(void)
```

(v předchozí kapitole byly probrány podrobnosti ovládání). Tato funkce bude automaticky, v intervalu 50ms, kontrolovat stav snímačů nárazníků (stisknutý nebo ne) a jejich aktuální stav zapíše do proměnných

```
bumper_left a bumper_right
```

Tyto proměnné můžete použít kdekoliv v programu tj. v podmíněných příkazech, cyklech atd. nebo je přiřazovat do dalších proměnných.

Příklad:

```
#include "RP6RobotBaseLib.h"

int main(void)

      initRobotBase();  // Initialize the Microcontroller

      setLEDs(0b001001); // Turn LEDs 1 and 4 on (both green)

      while(true)
      {
            // Set the LEDs depending on which
            // bumpers are pressed down:
            statusLEDs.LED6 = bumper_left;  // Left bumper pressed
            statusLEDs.LED4 = !bumper_left; // Left bumper released
            statusLEDs.LED3 = bumper_right; // Right bumper pressed
            statusLEDs.LED1 = !bumper_right; // Right bumper released
            // Both bumpers pressed down:
            statusLEDs.LED2 = (bumper_left && bumper_right);
            statusLEDs.LED5 = statusLEDs.LED2;
            updateStatusLEDs(); // update LEDs...

            // Check bumper status:
            task_Bumpers(); // Frequently call this from the main loop!
      }
      return 0;
}
```

Ukázkový program používá, ke zobrazení stavu nárazníků, stavové LED. Stisknutí levého nárazníku rozsvítí LED6 a zhasne LED4. Naopak uvolnění levého nárazníku zhasne LED6 a rozsvítí LED4. Stisknutí levého nárazníku rozsvítí LED6 kdykoliv, ale zde chceme ukázat obecné používání LED a vy můžete řízení LED použít jinde. Příklad funguje stejně pro pravý nárazník LED3 a LED1. Stisknutí obou nárazníků rozsvítí LED2 a LED5.

Výbava automatické kontroly nárazníků byla evidentně vytvořena nějakým voláním samostatně definované funkce, které pokaždé automaticky zjistí změnu stavu nárazníků. Obvykle se budou nárazníky spínat jen zřídka a jejich sledování v hlavním programu se provádí jen, když je to nezbytné.

Jazyk C nám umožňuje definovat ukazatele na funkce a volat tyto funkce bez předchozí definice funkce v knihovně. Obvykle s při překladu vyžaduje definice funkcí ve vlastní knihovně, jinak dojde ke zkolabování kompilátoru.

Tato metoda nám umožňuje používat samo definované funkce - takzvanou “Obsluhu událostí”. Výsledkem stisknutí nárazníku bude automatické volání předem definované vyhrazené funkce (během 50ms). Tato speciální funkce musí být registrována jako obsluha události a bude poskytovat specifickou vlastnost: funkce nesmí vracet žádnou hodnotu a nemá parametr (návratová hodnota i parametr musí být “void”). Proto se funkce označuje jako: void bumpersStateChanged(void). Měli byste například registrovat obsluhu události na samém začátku hlavní funkce. Registrace obsluhy události se může provést následující funkcí:

```
void BUMPERS_setStateChangedHandler(void (*bumperHandler)(void))
```

Tento příkaz nemusíte přesně chápat – je to dlouhý příběh s krátkým významem pro tuto funkci. Parametrem je ukazatel na funkci...

Vysvětlíme to na tomto jednoduchém příkladu:

```
1  #include "RP6RobotBaseLib.h"
2
3  // Our "Event Handler" function for the bumpers.
4  // This function will be called automatically by the RP6Library:
5  void bumpersStateChanged(void)
6  {
7        writeString_P("\nBumper status changed:\n");
8
9        if(bumper_left)
10             writeString_P(" - Left bumper pressed down!\n");
11       else
12             writeString_P(" - Left bumper released!\n");
13       if(bumper_right)
14             writeString_P(" - Right bumper pressed down!\n");
15       else
16             writeString_P(" - Right bumper released\n");
17 }
18
19 int main(void)
20 {
21       initRobotBase();
22
23       // Register the Event Handler:
24       BUMPERS_setStateChangedHandler(bumpersStateChanged);
25
26       while(true)
27       {
28            task_Bumpers(); // Automatically check bumpers at 50ms intervals
29       }
30       return 0;
31 }
```

Program bude reagovat na změnu stavu nárazníků pouze jednou výstupem okamžitého stavu nárazníků. Například při stisknutí pravého nárazníku může být výstup:

Stavy nárazníku se změnily:
- levý nárazník nebyl stisknutý,
- pravý nárazník byl stisknutý.

Stisknutí obou snímačů nárazníků způsobí:

Stavy nárazníku se změnily:
- levý nárazník byl stisknutý,
- pravý nárazník byl stisknutý.

Zřídka se stane, aby se oba nárazníky stisknuly současně, většinou dostanete zprávu, že se stisknul jen jeden nárazník. Pokud jej stisknete větší silou, pravděpodobně se ukáže pouze jedna zpráva. Je to kvůli intervalu 50 ms.

Můžete poznamenat, že ukázkový program nikdy nevolá přímo funkci bumpersStateChanged. Knihovna RP6Library to zařídí automaticky při každé změně stavu nárazníků z funkce task_Bumpers. Ve skutečnosti task_Bumpers nejdříve nezná naši funkci bumpersStateChanged a tato fukce se musí volat pomocí ukazatele, který se správně nastaví na řádku 24.

Obsluha události se může samozřejmě vedle textového výstupu dále rozšiřovat – tj. domníváme se, že robot zastaví a couvá/zatáčí. To se však neprovádí ve vlastní obsluze události, ale někde jinde v programu. V obsluze události možná nastavíte proměnno příkazu, která se kontroluje v hlavním programu pro identifikaci možného pohybu. Vždy se snažte, aby byla obsluha události co nejkratší.

V obsluze události můžete použít všechny funkce z knihovny RP6Library, ale u funkcí “otáčení” a “pohyb” musíte být opatrní – probereme to v dalších kapitolách. V obsluze události NIKDY nepoužívejte režim blokování těchto funkcí (opakované stisknutí nárazníků například nebude příliš pracné).

Základní myšlenka obsluhy událostí se také používá v řadě dalších funkcí. Například ACS – které má velmi podobné používání voláním obsluhy události při každé změně stavu snímač předmětů.

Obsluhu události používáme také při příjmu kódů RC5 z dálkového ovladače. Libovolný příjem signálu kódovaného systémem RC5 inicializuje volání příslušné funkce obsluhy události. Pro tuto činnost není třeba používat obsluhu události – samozřejmě můžete pro kontrolu změn jednoduše použít podmíněný příkaz, ale obsluha události zjednoduší návrh programu. Opakování je matka moudrosti.

Doporučení: CD nabízí řadu podrobných ukázkových programů na toto téma.

4.6.5. Čtení ADC hodnot (baterie, proud motorů a snímače osvětlení)

Jak bylo popsáno v kapitole 2, je k ADC (analogově číslicový převodník) připojeno několik snímačů. Knihovna RP6Library samozřejmě poskytuje funkci pro čtení naměřených ADC hodnot:

```
uint16_t readADC(uint8_t channel)
```

Tato funkce vrací 10-bitovou hodnotu (0...1023) a pro hodnoty snímačů vyžaduje 16-bitovou proměnnou. Je možné číst následující kanály:

```
ADC_BAT           --> Snímač napětí baterie
ADC_MCURRENT_R    --> Snímač proudu pravého motoru
ADC_MCURRENT_L    --> Snímač proudu levého motoru
ADC_LS_L          --> Levý snímač osvětlení
ADC_LS_R          --> Pravý snímač osvětlení
ADC_ADC0          --> Volný ADC kanál pro individuální snímače
ADC_ADC1          --> Volný ADC kanál pro individuální snímače
```

Doporučení: dva konektory volných ADC kanálů nejsou obsazeny. Můžete sem zapájet konektory se standardní roztečí 2,54 mm a možná přidat dva kondenzátory 100 nF a pokud snímač (např. IR snímač vzdálenosti SHARP) potřebuje vyšší napájecí proud, tak ještě velký elektrolytický kondenzátor 470µF.

To vyžaduje určité zkušenosti s pájením! Pokud zkušenosti nemáte, může být lepší řešení v použití rozšiřujícího modulu.

Příklady:

```
uint16_t ubat = readADC(ADC_BAT);
uint16_t iMotorR = readADC(ADC_MCURRENT_R);
uint16_t iMotorL = readADC(ADC_MCURRENT_L);
uint16_t lsL = readADC(ADC_LS_L);
uint16_t lsR = readADC(ADC_LS_R);
uint16_t free_adc0 = readADC(ADC_ADC0);
uint16_t free_adc1 = readADC(ADC_ADC1);
if(ubat < 580)
  writeString_P("Warning! Low battery level!");
```

Jako reference se používá základní napájecí napětí 5 V, ale funkce se může modifikovat tak, že se použije vnitřní referenční napětí 2,56V obvodu ATmega32 (viz katalogový list MEGA32). Standardní snímače RP6 to obvykle nepotřebují.

Před dalším zpracováním ADC výstupu se provede posloupnost několika měření, výsledky se uloží do pole a vypočítá střední nebo minimální/maximální hodnota.

Zpracování několika hodnot může snížit chybu měření. Předpokládáme, že metoda “střední hodnoty” je například potřeba při měření napětí baterie. Napětí baterie se velmi mění podle zátěže, zvláště při změně podmínek zatížení vlivem motorů.

Analogicky můžeme snímače nárazníků realizovat pomocí ADC měření a v hlavním programu jednoduše použít pohodlnou funkci:

```
void task_ADC(void)
```

která během krátkého času vyhodnotí všechny ADC kanály. Volání této funkce provede postupné čtení všech ADC kanálů “na pozadí” (tím se ušetří čas, měření se automaticky spustí i přečte...) a výsledky se uloží v předem definovaných proměnných.

Každé ADC měření vyžaduje určitý čas a funkce readADC může blokovat průběh programu. Vlstní měření nepotřebuje programový zásah, během tohoto času můžeme provádět něco jiného (ADC je samostatný hardwarový modul).

Měření z jednotlivých kanálů se uloží do následujících 16-bitových proměnných, které mohou být kdykoliv použity na libovolném místě programu:

```
ADC_BAT:          adcBat
ADC_MCURRENT_L:   adcMotorCurrentLeft
ADC_MCURRENT_R:   adcMotorCurrentRightADC_LS_L: adcLSL
ADC_LS_R:         adcLSR
ADC_ADC0:         adc0
ADC_ADC1:         adc1
```

Jak vidíte, na rozdíl od funkce readADC, musíte při volání funkce task_ADC() používat tyto proměnné.

Příklad:

```
#include "RP6RobotBaseLib.h"

int main(void)
{
	initRobotBase();
	startStopwatch1();
	writeString_P("\n\nJust a sample ADC evaluation program...\n\n");
	while(true)
	{
		if(getStopwatch1() > 300) // Every 300ms...
		{
			writeString_P("\nADC Left-sided light-sensor: ");
			writeInteger(adcLSL, DEC);
			writeString_P("\nADC Right-sided light-sensor: ");
			writeInteger(adcLSL, DEC);
			writeString_P("\nADC Battery: ");
			writeInteger(adcBat, DEC);
			writeChar('\n');
			if(adcBat < 600)
				writeString_P("Warning! Low battery level!\n");
			setStopwatch1(0); // Reset Stopwatch1 to zero
		}
		task_ADC(); // ADC evaluation - this has to be called
	}               // permanently from the main loop!
	return 0;       // But then you can NOT use readADC anymore!
}
```

Na výstupu tohoto programu budou v intervalu 300 ms hodnoty změřené na obou snímačích osvětlení a napětí baterie. Jak uvidíte, program při poklesu napětí baterie pod cca 6 V, vydá varování.

4.6.6. ACS – Anti kolizní systém

Na rozdíl od CCRP5, který používá malý koprocesor, byl anti kolizní systém robotu RP6 implementován přímo na základním mikroprocesoru ATmega32. Tato architektura vyžaduje o něco větší programovací úsilí, ale na druhé straně umožňuje modifikaci a přizpůsobení zákazníkem. Konstrukce RP5 neumožňovala žádnou změnu software koprocesoru.

Rozsah detekce ACS, respektive vysílací výkon obou IR LED lze ovládat pomocí následujících funkcí:

```
void setACSPwrOff(void)       --> Zrušení ACS IR-LED
void setACSPwrLow(void)       --> Krátký dosah
void setACSPwrMed(void)       --> Střední dosah
void setACSPwrHigh(void)      --> Dlouhý dosah
```

Jelikož je ACS kompletně implementován v software, vyžaduje časté volání následující funkce z hlavní smyčky:

```
void task_ACS(void)
```

Tato funkce zcela ovládá ACS. Další zpracování se může provádět podobným způsobem jaký byl předveden u nárazníků.

Knihovna RP6Lib poskytuje dvě proměnné:

`obstacle_left` a `obstacle_right`

Každá z nich se nastaví na TRUE, jakmile ACS detekuje překážku. Pokud se na TRUE nastaví obě proměnné, překážka bude umístěna přímo před robotem.

Volitelně můžete použít obsluhu události ACS.

```
void ACS_setStateChangedHandler(void (*acsHandler)(void))
```

Tato funkce registruje obsluhu události, která musí mít následující tvar:

```
void acsStateChanged(void)
```

Funkci můžete pojmenovat libovolně.

Ukázkový program demonstruje, jak se funkce používá. Začínáme registrací obsluhy události (řádek 44), pak aktivujeme všechny snímače včetně IR přijímače (řádek 46 – bez toho to samozřejmě nefunguje) a nastavíme výkon vysílače pro ACS IR LED (řádek 47). Hlavní smyčka opakovaně volá funkci task_ACS().

Další vyhodnocení se provede automaticky. Funkce acsStateChanged způsobí volání, jakmile dojde ke změně stavu ACS, který se objeví při detekci překážky nebo když zase zmizí. Program zobrazí okamžitý stav ACS textovou zprávou na terminálu a pomocí LED.

```c
#include "RP6RobotBaseLib.h"

void acsStateChanged(void)
{
	writeString_P("The ACS-status has changed!   L: ");

	if(obstacle_left)  // Obstacle on the left
		writeChar('o');
	else
		writeChar(' ');

	writeString_P(" | R: ");

	if(obstacle_right) // Obstacle on the right
		writeChar('o');
	else
		writeChar(' ');

	if(obstacle_left && obstacle_right) // Obstacle in the middle?
		writeString_P("   Amidships!");
	writeChar('\n');

	statusLEDs.LED6 = obstacle_left && obstacle_right; // In the middle?
	statusLEDs.LED3 = statusLEDs.LED6;
	statusLEDs.LED5 = obstacle_left;     // Obstacle on the left
	statusLEDs.LED4 = (!obstacle_left);  // LED5 inverted!
	statusLEDs.LED2 = obstacle_right;    // Hindernis on the right
	statusLEDs.LED1 = (!obstacle_right); // LED2 inverted!
	updateStatusLEDs();
}

int main(void)
{
	initRobotBase();

	writeString_P("\nRP6 ACS - Testprogram\n");
	writeString_P("_____________________\n\n");

	setLEDs(0b111111);
	mSleep(1000);
	setLEDs(0b001001);

	// Register the ACS Event Handler:
	ACS_setStateChangedHandler(acsStateChanged);

	powerON(); // Activate the IR receiver (incl. encoders etc.)
	setACSPwrMed(); // set the ACS medium transmit power.

	while(true)
	{
		task_ACS(); // Frequently call the task_ACS function!
	}
	return 0;
}
```

Ukázkový program také demostruje, jak se blikají a zhášejí jednotlivé LED.

Robot byste měli připojit k PC a sledovat výstup na terminálu a také pozorovat LED. A pak už jenom posouvejte ruku nebo jiný předmět přímo před robot.

Jsou známé zdroje rušení, které ovlivňují ACS. Některé typy zářivkových svítidel a podobných světelných zdrojů může virtuálně oslepit robot nebo rapidně snížit citlivost. Pokud zjistíte nějaké problémy, můžete zaít zhášet všechny rušivé zdroje osvětlení (doporučení: robot můžete umístit přímo před obrazovku, která se může pokládat za potenciální zdroj problémů stejně jako zářivky použité v podsvícení).

Dosah detekce samozřejmě velmi závisí na povrchu překážky. Černé povrchy obvykle neodrážejí stejné množství světla jako jasně bílý povrch. ACS může ignorovat tmavé předměty.

Kritická situace může nastat při podpoře ACS uttrazvukovými snímači nebo dokonalými IR snímači.

Dříve než pustíte robot na výlet po místnosti, měli byste provést několik jednoduchých testů ACS zkouškou schopnosti detekovat několik různých předmětů. Speciálně zkuste najít předměty, které NEJDE správně detekovat. Tento test vám umožní odstranit nevhodné překážky před zahájením provozu robotu ... ale ve srovnání s koprocesorem CCRP5, nezpůsobí chyba ACS žádné problémy, protože nárazník chrání před poškozením IR LED.

4.6.7. Funkce IRCOMM a RC5

IR přijímač umožňuje robotu RP6 přijímat signály ze standardního televizního dálkového ovladače, ale program je omezen na používání kódování RC5. Většina univerzálních dálkových ovladačů (viz obrázek) může být na tento kód naprogramována – prostudujte návod vašeho dálkového ovladače, zda může nastavit kód RC5. Pokud v tabulce kódů chybí kód RC5, můžete jednoduše otestovat ovladač od jiného výrobce.

Systém ACS bude vysílání dálkového ovládání RC5 ignorovat a tento signál obvykle neruší detekci překážek. Systém bude stále schopen detekce překážek, ale může reagovat pomaleji, protože je provoz ACS omezen na přestávky mezi vysíláním povelů RC5. Pokud dálkové ovládání nepoužívá RC5, může ACS fungovat chybně.

Vhodný software umožňuje ovládání RP6 libovolným infračerveným dálkovým ovladačem.

Vysílání IR signálů lze s výhodou použít také pro IRCOMM. Obě vysílací diody na předním panelu robotu jsou umístěny nahoře a míří na strop. Odrazy od stropu a dalších předmětů nebo přímý dosah umožňují vzájemnou komunikaci mezi roboty nebo základnovou stanicí. Komunikace je relativně pomalá (přenos datového paketu zabírá přibližně 20 ms plus krátkou mezeru), ale umožňuje vysílání krátkých příkazů a jednotlivých naměřených hodnot. Dosah vysílače je omezen na vzdálenost okolo 2 až 4 metrů uvnitř jediné místnosti (závisí na světelných podmínkách, překážkách, povrchu stropu i umístění horní rozšiřující desky).

Komunikační dosah budete moci rozšířit přidáním další IR LED (řízené například dalším tranzistorem typu MOSFET s větší kapacitou a menším sériovým rezistorem).

Synchronizace činnosti ACS se řídí funkcí task_ACS(), která se musí opakovaně volat z hlavní smyčky pro zajištění pracování příjmu IR signálů – a dále pro správu přenosu přes IRCOMM.

Datové pakety RC5 tvoří adresa zařízení, kód tlačítka a přepínací bit. Pěti bitová adresa zařízení říká, které zařízení se ovládá – například televize, video, Hi-Fi systém atd. Pro naši aplikaci se může adresa zařízení použít pro adresování několika jednotlivých robotů. Šest bitů kódu tlačítka odpovídá tlačítku stisknutému na dálkovém ovladači, ale může dobře posloužit k přenosu dalších údajů. Umožňuje sice přenos pouze šesti bitů, ale 8-bitová data lze předávat dvěma samostatnými přenosy nebo se dva bity použijí k adresaci či přepínání významu.

Standardní dálkové ovladače používají přepínací bit k rozlišení trvalého stisknutí nebo opakovaného stisknutí stejného tlačítka. My však využíváme přepínací bit pro další funkčnost komunikace mezi roboty.

Datový paket RC5 se může vyslat následující funkcí:

```
void IRCOMM_sendRC5(uint8_t adr, uint8_t data)
```

ve které `adr` odpovídá adrese zařízení a `data` je kód tlačítka respektive hodnota dat. Parametr `adr` vám umožní nastavit přepínací bit na nejvýznamnějším bitu (MSB) následující aplikací konstanty TOGGLEBIT:

```
IRCOMM_sendRC5(12 | TOGGLEBIT, 40);
```

Tento příkaz vyšle datový paket RC5 do zařízení s adresou 12, aktivuje přepínací bit a hodnota dat je 40.

```
IRCOMM_sendRC5(12, 40);
```

Toto je stejná situace bez aktivace přepínacího bitu.

Stejně jako u nárazníků a ACS, je možné příjem RC5 dat řešit jako obsluhu události. Jakmile se přijmou RC5 data, bude obsluha události automaticky volat funkci task_ACS(). To například umožňuje sestavení programu, který otočí robot doleva při příjmu kódu 4 a otočení doprava při kódu 6...

Jeden ukázkový program nabízí tuto funkčnost: veškeré pohyby se ovládají pomocí IR dálkového ovladače.

Předepsaný tvar obsluhy události musí odpovídat:

```
void receiveRC5Data(RC5data_t rc5data)
```

ale funkci můžete samozřejmě libovolně přejmenovat.

```
void IRCOMM_setRC5DataReadyHandler(void (*rc5Handler)(RC5data_t))
```

Tato funkce umožňuje registraci předem definované obsluhy události tj.:

```
IRCOMM_setRC5DataReadyHandler(receiveRC5Data);
```

Po této definici se bude příslušná funkce volat při každém příjmu platného kódu RC5.

Další možnosti: `RC5data_t` je zvláštní, předem definovaný datový typ, který obsahuje adresu zařízení typu RC5, přepínací bit a kód tlačítka (respektive datovou hodnotu). Tato data můžete použít stejně jako ostatní ordinální proměnné pomocí následujících identifikátorů:

```
rc5data.device, rc5data.toggle_bit, rc5data.key_code
```

CD poskytuje ukázkový program, který ukazuje použití těchto proměnných.

Pozor: Nikdy neaktivujte trvale výstup IRCOMM! IR LED a budící obvod MOSFET jsou navrženy pro pulzní provoz a umožňují přenos pouze impulsů s periodou kolem jedné milisekundy! V opačném případě dojde při nabité baterii k velkému nárůstu spotřeby proudu. Neupravujte funkce IRCOMM, pokud neovládáte celou související problematiku. Zvláště se nesmí modifikovat obsluha přerušení pro řízení infračerveného obvodu.

4.6.8. Funkce snižování spotřeby

V předchozích kapitolách jsme používali funkci powerON(), ale nepopsali jsme její význam. RP6 může ušetřit část energie vypnutím ACS, odměřovacího systému, proudových snímačů a LED PowerON. Vypnutí těchto snímačů uspoří 10 mA.

Zapnutí snímačů se může provést voláním makra:

```
powerON()
```

a pro úsporu energie a vypnutí snímačů můžete zavolat:

```
powerOFF()
```

Obě makra pouze nastaví I/O vývody.

Před používáním ACS, IRCOMM nebo ovládání motorů se musí provést makro powerON()! Jinak nebudou napájeny příslušné obvody snímačů. Snímače musí fungovat, protože rutiny ovládání motorů vyžadují zpětnovazební signály z enkodérů a proudových snímačů.

Pokud byste zapomněli volat powerON(), motory se, okamžitě po spuštění, zastaví. Tento chybový stav se indikuje blikáním čtyř červených stavových LED.

4.6.9. Funkce pohonného systému

Knihovna RP6Library poskytuje pohodlné funkce pro ovládání pohonného systému robotu. Některé funkce budou automaticky regulovat rychlost motoru pomocí zpětné vazby z enkodérů, kontrolovat proud tekoucí do motorů, automaticky ujedou přesnou vzdálenost a zpracují řadu dalších úkolů. Tyto možnosti jsou velmi pohodlné – ale jen podobné jako u jiných robotických systémů – při náročném používání musíme předpokládat nějaké mimořádné vlastnosti.

Momentální stav vývoje nesplňuje optimálně naše představy. Existuje řada věcí, které lze vylepšit!

```
void task_motionControl(void)
```

Funkci task_motionControl budeme opakovaně volat z hlavní smyčky programu, jinak nebude automatická regulace fungovat! Opakované volání z hlavního programu sebou přináší volání této funkce při každém průchodu hlavní smyčky. Volání funkce v intervalu 10 až 50 milisekund

bude dostatečné, ale lepší je volání funkce v mnohem kratších intervalech. Častější volání funkce nezpůsobí žádné problémy, protože časování je řízené hardwarem. Ze stejného důvodu můžeme měnit interval volání funkce v rozsahu od 1ms do více než 10ms. Velmi časté volání funkce nezabírá příliš mnoho strojového času, jelikož se funkce celá zpracuje v předem definovaném minimálním intervalu.

Pokud se funkce používá správně, bude automatická regulace udržovat požadovanou velikost konstantních otáček motoru.

Regulace rychlosti je zajištěna přesnou regulační odchylkou, zjištěnou při každém měřicím cyklu a jejich sčítáním (takzvaný integrační regulátor). Tato chybová hodnota se používá pro nastavení napětí motoru pomocí PWM modulu mikroprocesoru. Pokud je rychlost příliš nízká, bude regulační chyba kladná a napětí motoru by se mělo v určitém poměru zvýšit. Když je rychlost příliš vysoká, musí se napětí snížit. Tato metoda bude svižně nastavovat rychlost RP6 na relativně konstantní hodnotu PWM (ve které se malé odchylky považují za normální). Regulace umožňuje stabilizaci rychlosti nezávislou na napětí baterie, zátěži (hmotnost, povrchové podmínky, sklon atd.) a výrobní toleranci. Pokud se budete pokoušet pohánět robot pevnou hodnotou PWM, může rychlost robotu velmi záviset na účinnosti motorů, zátěži a napětí baterie. V rozdílné rychlosti levého a pravého motoru se mohou dále projevit důsledky výrobních tolerancí.

Rutina regulace rychlosti odpovídá také za reverzní chod motoru, který se musí otáčet jedním směrem a kdykoliv změnit směr otáčení i při rychlosti 15cm/s bez poškození motorů a převodovek. Pokud se má provést změna směru otáčení, rychlost robotu se automaticky sníží na nulu, následuje změna směru a následně zvýšení otáček na nastavenou rychlost.

Řídící systém rychlosti a směru otáčení musí dále sledovat spotřebu proudu v motorech. Při překročení proudových podmínek systém automaticky zastaví motory. Toto bezpečnostní opatření chrání motory před přetížením a přehřátím, které může trvale zničit motory.

Pokud se během 20 sekund objeví tři události překročení proudu, provede ochranný systém nouzové zastavení a spustí blikání čtyř stavových LED. Před další činností se musí robot resetovat.

Dále systém monitoruje poškození enkodérů nebo motorů (které se může projevit, pokud se v nich budete často šťourat...). Pokaždé, když se to stane, může funkce řízení pohybu zvedat hodnotu PWM až na maximum a robot se stane neřiditelný ... a samozřejmě musíte předpokládat nežádoucí zkušenosti! V každém případě se v takovém případu robot úplně zastaví.

Musíme také stručně uvést, že řízení pohybu také obsahuje funkce pro pohyb na definovanou vzdálenost a otáčení o definovaný úhel.

Jak můžete vidět, je funkce velmi důležitá pro automatickou regulaci motoru. Klíčová skutečnost je, že vlastní funkce motionControl nemá žádný parametr jako například požadovanou rychlost. Provozní parametry se musí nastavovat přes další funkce, které nyní podrobně probereme.

```
void moveAtSpeed(uint8_t desired_speed_left, uint8_t desired_speed_right)
```

Tato funkce nastavuje provozní rychlost. Jednotlivé parametry budou definovat požadovanou rychlost levého a pravého motoru. Opakované volání funkce motionControl (jak bylo popsáno v předchozí kapitole) zajistí nastavení hodnot pro regulaci rychlosti. Nastavení této hodnoty na nulu vyvolá zpomalování následované úplným vypnutím modulu PWM.

```
getDesSpeedLeft() a getDesSpeedRight()
```

Tato makra umožňují čtení okamžitého nastavení hodnoty rychlosti. Jak můžete vidět v následujícím ukázkovém programu, používá se velmi jednoduše:

```
1  #include "RP6RobotBaseLib.h"
2
3  int main(void)
4  {
5        initRobotBase(); // Initialize the Microcontroller
6
7        powerON();  // Activate Encoders & Motor current sensors (IMPORTANT!)
8
9        moveAtSpeed(70,70); // set desired speed
10
11       while(true)
12       {
13             // Frequently call the motionControl function from the
14             // main loop - it will adjust both motor speeds:
15             task_motionControl();
16             task_ADC(); // has to be called for Motor current sensors!
17       }
18       return 0;
   }
```

... a nyní se začne RP6 pohybovat! Robot samozřejmě nebude reagovat na některé překážky a pohybuje se pouze dopředu! Systém se bude pouze pokoušet regulovat rychlost a automaticky nastavovat výkon motorů – tj. stoupání a sjíždění svahu.

BUĎTE OPATRNÍ: Toto chování může být velmi nebezpečné pro vaše vlasní prsty – dávejte pozor, aby se prsty nedotýkali pásové dráhy a hnacích kol. Udržujte v čistotě desku plošných spojů a vedení pásů. Existuje velké nebezpečí úrazu! Jak jsme již zmínili, výkon motoru se při zátěži automaticky zvýší a motory jsou poměrně výkonné!

Parametry rychlosti pro funkci moveAtSpeed nejsou specifikovány v cm/s nebo ekvivalentních jednotkách, ale v jednotkách rychlosti otáčení.

Rychlost robotu závisí především na reálném obvodu pásů a kol neboli jinak řečeno na rozlišení enkodérů. Můžeme zde uvažovat toleranci od 0,23 do 0,25 mm na každý segment enkodéru. Proto se má rozlišení enkodérů měřit.

Systém bude měřit rychlost otáčení v intervalech 200ms, který odpovídá rychlosti 5 krát za sekundu. Takže jednotkou je “počet segmentů enkodéru za 200ms”. Hodnota 70, která se používá v příkladu na předchozí stránce, může být interpretována jako 70 • 5 = 350 započítání

segmentů enkodéru za sekundu (což odpovídá rychlosti cca 8 až 8,7 cm/s – podle reálného rozlišení enkodéru). Minimální regulovatelná rychlost otáčení je kolem 10 • 5 = 50 a maximální rychlost otáčení je přibližně 200 • 5 = 1000. Kapitola 2 již zmiňovala příčinu tohoto omezení rychlosti, ale přesto doporučujeme omezit hodnotu na 160 při trvalém pohybu a hodnotu 200 používat pouze na krátký čas!

```
getLeftSpeed() a getRightSpeed()
```

Tato makra umožňují čtení měřené hodnoty rychlosti otáčení. Budou vracet hodnoty ve stejných jednotkách jak bylo popsáno výše.

```
void changeDirection(uint8_t dir)
```

Tato funkce nastaví směr otáčení motorů. Jak již bylo probráno, robot nejdříve zpomalí, pak změní směr a nakonec zrychlí na předchozí nastavení rychlosti.

Podporuje následující parametry:

FWD – dopředu

BWD – dozadu

LEFT – otáčení doleva

RIGHT – otáčení doprava

Makro:

```
getDirection()
```

umožňuje čtení okamžitého směru.

Příklad:

```
#include "RP6RobotBaseLib.h"

int main(void)
{
    initRobotBase(); // Initialize the Microcontroller
    powerON();         // Activate Encoders and Motor current sensors!

    moveAtSpeed(60,60); // Set desired speed
    startStopwatch1();  // Start Stopwatch1

    while(true)
    {
        if(getStopwatch1() > 4000) // Have 4000ms (= 4s) passed by?
        {
            // Change moving direction:
            if(getDirection() == FWD) // If we are driving forwards,
                changeDirection(BWD); // then set direction to backwards!
            else if(getDirection() == BWD) // If we are driving backwards,
                changeDirection(FWD); // then set direction to forwards!
            setStopwatch1(0); // Reset Stopwatch1
        }
        task_motionControl(); // Automatic motion control
        task_ADC(); // has to be called for the current sensors.
    }
    return 0;
}
```

V tomto ukázkovém programu RP6 se nejprve rozjede dopředu – to je výchozí nastavení pohybu po reset. Použijeme jedny stopky pro měření 4 sekundového čekání a pak obrátíme směr. Na řádku 16 a 18 se určí okamžitý směr otáčení a určí příslušné změny. To se opakuje ve 4 sekundových intervalech, které způsobí, že se robot stále pohybuje dopředu a dozadu.

Je jasné, že robot stále ignoruje jakékoliv překážky!

Pomocí funcí, které jsme dosud probrali, není snadné ujet specifikovanou vzdálenost. K tomuto účelu slouží dvě speciální funkce:

```
void move(uint8_t desired_speed,uint8_t dir,uint16_t distance,uint8_t blocking)
```

Funkce move umožňuje, aby robot ujel specifikovanou vzdálenost. Musíme předat požadovanou rychlost, směr (FWD nebo BWD) a vzdálenost v počtu enkodérových přírustků.

Makro:

```
DIST_MM(DISTANCE)
```

je užitečné pro převod vzdálenosti z milimetrů na přírustky enkodéru. Nejdříve samozřejmě musíte kalibrovat rozlišení enkodéru (viz dodatek). Následující ukázkový příklad ukazuje, jak se makro používá.

Robot se pokusí ujet požadovanou vzdálenost co nejpřesněji. Funkce motionControl začne zrychlovat na nastavenou rychlost a krrátce před dosažením požadované vzdálenosti zpomalí, aby ji nepřejel. Přesnost je kolem 5 mm, což můžeme obvykle považovat za docela dobrou.

Funkce nepodporuje pohyb na velmi krátkou vzdálenost pod 5 cm, ale to lze samozřejmě zlepšit!

Parametr dráhy, nazvaný “blokování” je speciální vlastnost, kterou není třeba podrobně popisovat.

Funkce obvykle nastaví pouze několik proměnných a okamžitě se vrátí do programu. Robot je pak ovládán “na pozadí” funkcí motionControl. To je užitečné pro provádění dalších úkolů, jako je například vyhýbání překážek. Jenomže když má robot pouze sledovat předem určenou geometrický obrazec, můžete to změnit parametrem blokování.

Nastavení parametru na “TRUE” (to znamená 1), bude funkce ve smyčce volat funkci motionControl dokud se nedosáhne zadaná vzdálenost. Program nedokáže tuto funkci opustit – místo toho bude na potřebnou dobu “blokovat” normální průběh programu.

Nastavení parametru na “FALSE” způsobí, že funkce bude provádět vše, co již bylo popsáno. Po nastavení příkazu “zahájení přesunu do definované vzdálenosti“ se okamžitě vrátí do hlavního programu. Pokud zavoláte další funkce, které nastavují rychlost nebo dávají další pohybové příkazy, nemusí se program chovat korektně. Raději počkejte na dokončení původního pohybového příkazu nebo můžete alternativně zrušit příkaz.

Funkce:

```
uint8_t isMovementComplete(void)
```

se může používat ke kontrole, zda byl dokončen pohybový příkaz. Pokud není pohybový příkaz dokončen, bude návratová hodnota “FALSE”.

Kdykoliv byl pohybový příkaz překažený, tj. například při detekci překážky, můžete ukončit všechny pohyby voláním funkce:

```
void stop(void)
```

který zastaví všechny pohyby.

Někdy může být výhodné použít pohybovou funkci pro otáčení jednoduchým nastavením směrového parametru na LEFT nebo RIGHT místo FWD nebo BWD a zadáním vhodné hodnoty vzdálenosti, která odpovídá úhlu natočení. Tato metoda je poněkud neohrabaná a není příliš výkonná. Z tohoto důvodu nabízíme specializovanou funkci pro otáčení na místě:

```
void rotate(uint8_t desired_speed,uint8_t dir,uint16_t angle,uint8_t blocking)
```

Tato funkce se chová stejně jako příkaz “move”, pouze se místo vzdálenosti určí úhel natočení. Parametr blokování se může používat také v této funkci.

Následující program ukazuje, jak se používají obě funkce:

```
#include "RP6RobotBaseLib.h"

int main(void)
{
	initRobotBase();
	setLEDs(0b111111);
	mSleep(1500);

	powerON(); // Activate encoders and motor current sensors!

	while(true)
	{
		setLEDs(0b100100);
		move(60, FWD, DIST_MM(300), true); // Move 30cm forward
		setLEDs(0b100000);
		rotate(50, LEFT, 180, true);  // Rotate 180° to the left
		setLEDs(0b100100);
		move(60, FWD, DIST_MM(300), true); // Move 30cm forward
		setLEDs(0b000100);
		rotate(50, RIGHT, 180, true); // Rotate 180° to the right
	}
	return 0;
}
```

Robot se pojede 30 cm dopředu, otočí se o 180° doleva, přejede 30 cm dozadu, otočí se o 180° doprava a začne od začátku. Pokud byste nastavili blokování všech parametrů na FALSE, program nemusí vůbec fungovat. Hlavní smyčka nevolá funkci task_motionControl a všechny pohybové funkce se zavolají v jediné posloupnosti. Změna pouze jednoho blokovacího parametru na FALSE způsobí, že program více méně nefunguje tak, jak jste zamýšleli. Jedna pohybová fáze se úplně přeskočí.

Při požadavku na provádění posloupnosti, musíme vždy nastavit parametr blokování na TRUE! Můžete ještě provést elementární reakci na překážku s parametrem blokování nastaveným na TRUE – pomocí obsluhy události. Tyto obsluhy událostí volané z libovolného místa nezávisí na nastavení parametru blokování na TRUE! Tato metoda však může v některých složitějších situacích zkolabovat.

Řídící jednotka obecně, při vyhýbání překážek, interpretuje příkazy a u podobných procesů doporučujeme používání neblokovaného režimu a nastavení parametru blokování na FALSE. Jak jsme zmínili v této kapitole dříve, tento režim umožňuje funkce jízdy/otáčení pro ovládání pohybu robotu nezávislé na dalším chodu programu.

Několik podrobných příkladů těchto témat najdete na CD.

4.6.10. task_RP6System()

V posledních kapitolách se naučíme, že je pro správnou funkci ACS/IRCOMM, řízení pohybu, nárazníky a vyhodnocení ADC na pozadí nezbytné časté volání čtyř funkcí uvnitř hlavní smyčky. Právě pro zjednodušení a lepší přehlednost programu nabízí knihovna RP6Library následující funkci:

```
void task_RP6System(void)
```

která bude postupně volat funkce:

```
task_ADC();
task_ACS();
task_bumpers();
task_motionControl();
```

Většina ukázkových programů na CD bude používat pouze tuto funkci – sotva kdy budeme potřebovat některou z těchto funkcí přímo.

4.6.11. Funkce sběrnice I²C

Na konci této kapitoly se zaměříme na funkce sběrnice I²C, které mohou být využity pro komunikaci s dalšími mikroprocesory a rozšiřujícími moduly.

Existují dvě verze funkcí sběrnice I²C – jedny pro režim slave a druhé pro režim master.

Pozor: nemůžete používat obě verze funkcí současně!

Vložit můžete pouze jednu ze dvou verzí a musíte dávat pozor, aby byla uvedena v makefile. Do makefile se již musí přidat příslušné vstupy ukázkových programů – ve většině příkladů mohou být zbaveny komentářů. Ještě jednou: použijte pouze jeden z těchto vstupů! V opačném případě vystaví kompilátor chybové hlášení (je to kvůli tomu, že se při vložení obou verzí, může vektor přerušení od TWI definovat dvakrát).

4.6.11.1. I²C slave

Na základní jednotce robotu je důležitější režim slave, protože se mnoho obecných úloh přidává na další mikroprocesor nebo řízení dalšího robotu. Existuje ukázkový program, který umožňuje přístup téměř ke všem funkcím základní jednotky robotu přes sběrnici I²C (RP6Base_I2CSlave).

Základ obou režimů (master a slave) využívá přerušení. Čistě softwarová implementace I²C v režimu slave není jednoduchá (dokonce přímo nemožná). Režim master může být snadno implementován v software, ale pro udržení shodné struktury v obou režimech, použijeme také přerušení. Další výhodou je, že přenosy dat v režimu master se mohou provádět na pozadí což šetří čas.

```
void I2CTWI_initSlave(uint8_t address)
```

Tato funkce bude inicializovat TWI modul mikroprocesoru jako I²C slave a umožňuje definovat adresu jako parametr. V adrese můžete současně definovat, zda bude řadič reagovat na takzvané “Všeobecné volání” nebo ne. Když je sběrnice adresována nulou, nazývá se tento stav “všeobecné volání - General Call”. Tuto funkci můžete například použít ke snadnému současnému přepnutí všech řadičů připojených ke sběrnici do režimu se sníženou spotřebou.

Příklad:

```
I2CTWI_initSlave( adr | TWI_GENERAL_CALL_ENABLE );   // povoluje general call
I2CTWI_initSlave(adr);                               // zakazuje general call
```

Registry I²C

Obvykle mohou být I²C periferie řízeny přes několik registrů, které umožňují čtení/zápis. Proto jsou slave rutiny navrženy tak, aby poskytovaly řadu “registrů” (v této aplikaci pole 8-bitových proměnných), které lze číst nebo zapisovat ze zařízení typu master.V případě čtení dat z registru nebo zápisu dat do registru musí zařízení master vyslat adresu slave a následně adresu registru.

Existují zde dvě pole typu uint8_t. Jedno pro čtení a druhé pro zápis do registru.

Pole a proměnné se obecně nazývají:

```
I2CTWI_readRegisters, I2CTWI_writeRegisters a I2CTWI_genCallCMD
```

Registry pro čtení se nazývají I2CTWI_readRegisters a registry pro zápis I2CTWI_writeRegisters. Do proměnné I2CTWI_genCallCMD se ukládá většina zbývajících přijatých příkazů General Call. Výměna dat v režimu slave kompletně pracuje s těmito registry.

V případě, že máte data dostupná na sběrnici vložíte je do pole I2CTWI_readRegisters. Master nyní může tato data číst přes adresu příslušného místa v poli (shodnou s číslem registru). Pro příklad, pokud master musí přečíst data snímače ze slave, má slave nejprve vložit informaci do předem určené pozice v poli I2CTWI_readRegisters. Pak může master číst data přenesením čísla registru a následuje čtení vlastní informace. Číslo registru se automaticky zvýší, což umožňuje, aby master četl několik registrů v jediném průběhu.

Podobný postup se bude provádět při zápisu dat. Master na začátku vyšle číslo registru a pak začne přenášet data. Analogicky se, stejně jako při čtení, automaticky zvýší adresa registr. To masteru umožňuje zápis do několika registrů v jediném průběhu. Celá obsluha režimu slave běží, díky přerušení, na pozadí.

Pokud se při zápisu používají data současně s adresou, nemusí být datová struktura konsistentní. Pokud se čtou data z jednoho registru, může mezitím master přepsat další data svázaná s tímto registrem. Lepší způsob obsluhy těchto přenosů je přechodné uložení adresy umístění. Čtení dat může také vést k nejednoznačnosti souvisejících proměnných (tj. dolního a horního byte u 16-bitových proměnných).

```
I2CTWI_readBusy a I2CTWI_writeBusy
```

Obsluha přerušení nastaví proměnnou I2CTWI_writeBuisy na TRUE – a to se může použít ke kontrole přístupu zápisu do těchto dat. Pokud je proměnná nastavena na FALSE, můžeme data přenášet z registrů do dočasných proměnných a použít je pro další zpracování.

Tuto situaci demonstruje následující ukázkový program a příklad “slave” na CD – je zde příkazový registr, který zařízení master používá k přenosu příkazů do slave (tj. “začni se pohybovat dopředu, rychlostí 100”). Hlavní smyčka zařízení typu slave trvale vyhodnocuje registr 0 dokud je proměnná I2CTWI_busy nastavena na FALSE. Po příchodu master příkazu do registru 0 bude následovat přenos dat z registru 0 a registrů 1 až 6 do dočasných proměnných, které se mohou později vyhodnotit. Parametr obvykle závisí na obsahu příkazové proměnné. Parametr 1 může například popisovat hodnotu rychlosti pohybového příkazu a parametr 2 směr. Ostatní parametry mohou být v tomto parametru ignorovány.

Proměnná I2CTWI_readBusy funguje podobně – nastaví se, jakmile se přečte registr a umožňuje kontrolu připravenosti zápisu registrů a chrání před nejednoznačným čtením. Aktuální implementace nemůže zaručit konsistenci na 100%, protože se během zápisu do registrů může zrušit TWI přerušení, což může způsobit další komplikace...

Tento příklad ukazuje, jak jednoduchý může být program I²C slave:

```
1   #include "RP6RobotBaseLib.h"
2   #include "RP6I2CslaveTWI.h"  // Include the I2C Library file (!!!)
3        // ATTENTION: do not forget to add this to the Makefile (!!!)
4
5   #define CMD_SET_LEDS 3 // LED command, which should be received
6                          // through the I2C Bus
7   int main(void)
8   {
9         initRobotBase();
10        I2CTWI_initSlave(10); // Initialise TWI set slave address to 10
11        powerON();
12
13        while(true)
14        {
15              // did we receive some command and is there NO write access?
16              if(I2CTWI_writeRegisters[0] && !I2CTWI_writeBusy)
17              {
18                    // save register contents:
19                    uint8_t cmd = I2CTWI_writeRegisters[0];
20                    I2CTWI_writeRegisters[0] = 0; // and reset cmd reg (!!!)
21                    uint8_t param = I2CTWI_writeRegisters[1]; // Parameter
22
23                    if(cmd == CMD_SET_LEDS) // LED command received?
24                          setLEDs(param); // set LEDs with the parameter
25              }
26              if(!I2CTWI_readBusy) // No read activities?
27                    // Proceed by writing current LED state to register 0:
28                    I2CTWI_readRegisters[0] = statusLEDs.byte;
29        }
30        return 0;
31  }
```

Vlastní program nebude nic dělat (evidentně) a tak potřebujete master pro ovládání zařízení typu slave. V tomto případě je slave na sběrnici I²C dostupný na adrese 10 (viz řádek 10). Program poskytuje dva registry pro zápis a jeden registr pro čtení dat. První registr (= registr číslo 0) se používá pro příjem příkazů. Tento zjednodušený příklad používá příkaz “3” pro nastavení LED (může to být libovolné číslo). Při příjmu libovolného příkazu – bez zápisového přístupu (viz řádek 16) – uloží program obsah příkazového registru 0 do proměnné “cmd” (řádek 19) a vynuluje příkazový registr 0, aby vyloučil opakované provádění příkazu! Program pokračuje uložením parametru z registru 1 do další dočasné proměnné a kontrolou příkazu 3 (řádek 23). Pokud je porovnání pravdivé, nastaví se LED podle hodnoty přijatého parametru.

Řádek 26 kontroluje, zda neprobíhá přístup čtení a uloží aktuální hodnotu registru LED do registru 0, který umožňuje čtení.

Pokud je řídící jednotka na základní desce naprogramována tímto způsobem, může jednotka master nastavit LED na základní desce přes sběrnici I²C a přečíst zpět aktuální stav LED.

Program nebude dělat nic jiného – mnohem podrobnější ukázkový program, který umožňuje virtuální ovládání všech dostupných funkcí robotu, můžete najít na CD. Předchozí příklad pouze demonstruje základní principy. Základem všeho je 16 registrů pro zápis a 48 registrů pro čtení. Pokud někdo potřebuje více nebo méně registrů, musí přizpůsobit příslušné definice knihovny RP6Library v hlavičkovém souboru RP6I2CSlaveTWI.h.

4.6.11.2. I²C master

V režimu master se může používat TWI modul mikroprocesoru ATmega32 k ovládání dalších zařízení, mikroprocesorů nebo snímačů přes sběrnici I²C.

```
void I2CTWI_initMaster(FREQ)
```

Tato funkce inicializuje modul TWI jak master. Režim master samozřejmě nevyžaduje adresu – ale pro TWI modul máme definovat kmitočet datového přenosu. Kmitočet definujete v kHz pomocí parametru FREQ. Obvyklá hodnota je 100 kHz, která se může nastavit parametrem 100. Přenosová rychlost může používat hodnoty až do 400. Horní limit TWI modulu se nesmí překročit.

Podle specifikace Atmel (viz katalogový list), nemůže TWI modul MEGA32 v režimu master fungovat s přenosovou rychlostí vyšší než maximálně 220 kbit/s. Přenosová frekvence 400 kHz by vyžadovala hodinovou frekvenci vyšší než 14,4 MHz, ale kvůli úspoře energie používáme hodinovou frekvenci 8 MHz. To však způsobuje jen malou komplikaci v časování, která neovlivní vaši práci. V režimu slave není žádný problém a může se používat přenosovou rychlost 400 kbit/s. Pokud skutečně potřebujete rychlou komunikaci, můžete se pokusit nastavit režim 400 kbit/s uvidíte, co se stane se zařízením slave nebo rozšiřujícím modulem RP6 CONTROL M32, který je taktován na 16 MHz. Obvykle může také pracovat na 8 MHz, ale s drobným časovým omezením. Ale nemůžeme to zaručit!

Přenos dat

Existuje řada funkcí pro přenos dat na sběrnici I²C. V podstatě jsou všechny tyto funkce podobné, ale umožňují přenos různého počtu byte.

```
void I2CTWI_transmitByte(uint8_t adr, uint8_t data)
```

přenáší jeden byte do specifikované adresy.

```
void I2CTWI_transmit2Bytes(uint8_t adr, uint8_t data1, uint8_t data2)
```

přenáší dva byte na specifikovanou adresu. Tuto funkci budete často potřebovat u řady I²C zařízení, které vyžadují formát dat:

adresa slave – adresa registru – data

```
void I2CTWI_transmit3Bytes(uint8_t adr, uint8_t data1, uint8_t data2, uint8_t data3)
```

používá se docela často, zvláště při zápisu slave komunikace s datovým formátem:

adresa slave – příkazový registr – příkaz – parametr 1

```
void I2CTWI_transmitBytes(uint8_t targetAdr,uint8_t *msg,uint8_t numberOfBytes)
```

Základem této funkce bude přenos maximálně 20 byte na specifikovanou adresu. Pro přenos větších bloků dat můžete v hlavičkovém souboru zvětšit konstantu I2CTWI_BUFFER_SIZE.

Při zadání registru pro přenos dat můžete jednoduše použít první byte bufferu.

Používání těchto funkcí je skutečně jednoduché:

```
I2CTWI_transmit2Bytes(10, 2, 128);
I2CTWI_transmit2Bytes(10, 3, 14);
I2CTWI_transmit3Bytes(64, 12, 98, 120);
```

Předchozí příklad postupně přenese dvakrát dva byte do zařízení slave s adresou 10 a dále tři byte do slave s adresou 64. Další funkce transmitXBytes se používá podobným způsobem.

Další příklad:

```
uint8_t messageBuf[4];
messageBuf[0] = 2;                  // zde muzete specifikovat adresu registru
messageBuf[1] = 244;                // data...
messageBuf[2] = 231;
messageBuf[3] = 123;
messageBuf[4] = 40;
I2CTWI_transmitBytes(10,&messageBuf[0],5);
```

Tímto způsobem můžete přes sběrnici I^2C přenést více byte (v tomto případě 5).

Výše popsané funkce nebudou blokovat průchod programu, dokud nebude sběrnice I^2C zaneprázdněná. Zaneprázdnění sběrnice I^2C způsobí, že funkce čeká, než se dokončí všechny přenosy. Proto vám kontrola dokončení před voláním funkce umožní provádět další činnost během probíhajícího přenosu. Přenos dat pomocí sběrnice I^2C je časově relativně náročná v porovnání s rychlostí mikroprocesoru a kontrolou můžete uspořit spoustu času.

Následující makro signalizuje zda je TWI modul zaneprázdněn nebo ne:

```
I2CTWI_isBusy()
```

pokud je modul volný, mohou se přenášet nová data.

Příjem dat

Knihovna RP6Library poskytuje několik alternativ příjmu dat. Nejprve představíme pár blokových funkcí, které byly navrženy analogicky k funkcím zápisu. Dále probereme funkce pro příjem dat na pozadí.

Nejprve jednoduchá bloková funkce pro čtení dat:

```
uint8_t I2CTWI_readByte(uint8_t targetAdr);
```

Tato funkce čte jeden byte ze zařízení slave. Tato funkce může být použita samostatně, obvykle před tím přenese číslo registru pomocí I2CTWI_transmitByte.

Pokud byste chtěli například číst registr 22 ze zařízení slave s adresou 10:

```
I2CTWI_transmitByte(10, 22);
uint8_t result = I2CTWI_readByte(10);
```

Následující funkce umožňuje čtení několika byte:

```
void I2CTWI_readBytes(uint8_t targetAdr, uint8_t *messageBuffer, uint8_t
numberOfBytes);
```

Příklad:

```
I2CTWI_transmitByte(10, 22);
uint8_t results[6];
I2CTWI_readBytes(10,results,5);
```

Tento útržek programu přečte 5 byte z registru 22 zařízení slave s adresou 10. Pokud jsou data skutečně přečtena z registru 22, tak je slave změní. Někdy se číslo registru zvýší automaticky (stejně jak to dělá kód funkce slave v knihovně RP6Library) a jindy se slave chová úplně jinak. Musíte to nastudovat v dokumentaci obvodu.

Čtení dat na pozadí je trochu složitější úkol. Nejdříve začnete požadovaným počte byte ze zařízení slave. Proces běžící na pozadí začne přijímat byte ze zařízení slave. Ve zbývajícím čase může mikroprocesor provádět jiné úkoly, aniž by rušil obsluhu přerušení. V průběhu komunikace musíte samozřejmě opakovaně volat funkci z hlavní smyčky a kontrolovat příchod požadovaných dat ze zařízení slave nebo výskyt chyby komunikace. Při příchodu dat funkce automaticky vyvolá předem definovanou funkci pro obsluhu události, která provede další zpracování dat a ta může bezprostředně vyvolat následující množinu dat z dalších registrů. Každý požadavek bude označen vlastním ID.

```
void task_I2CTWI(void)
```

je funkce, která se má opakovaně volat z hlavní smyčky. Tato úloha byla navržena pro kontrolu bezchybného dokončení všech přenosů a případně pro volání obsluhy události.

```
void I2CTWI_requestDataFromDevice(uint8_t requestAdr, uint8_t requestID, uint8_t
numberOfBytes)
```

Tato funkce umožňuje vyžádání dat ze zařízení slave. Po zavolání funkce proces na pozadí automaticky příjme data.

Date můžeme vyzvedávat postupně voláním funkce:

```
void I2CTWI_getReceivedData(uint8_t *msg, uint8_t msgSize)
```

Data se mohou vyzvednout přímo při volání obsluhy události. Obsluha to registruje voláním funkce:

```
void I2CTWI_setRequestedDataReadyHandler(void (*requestedDataReadyHandler) (uint8_t))
```

Obsluha události musí mít následující tvar:

```
void I2C_requestedDataReady(uint8_t dataRequestID)
```

Tato obsluha se bude volat s ID datového požadavku uvedeným v parametru. ID můžeme použít k rozlišení různých zařízení typu slave.

Vedle obsluhy requestedDataReady existuje obsluha události pro zpracování chyb. Kdykoliv se v přenosu dat objeví chyba, tj. nereaguje zařízení typu slave, zavolá se speciální obsluha přerušení. Registruje se pomocí této funkce:

```
void I2CTWI_setTransmissionErrorHandler(void (*transmissionErrorHandler) (uint8_t))
```

Obsluha musí mít tvar:

```
void I2C_transmissionError(uint8_t errorState)
```

parametr funkce definuje kód chybového stavu. Přehled chybových kódu “režimu I2C master” najdete v hlavičkovém souboru.

Ve skutečnosti můžete obsluhu události použít k detekci chyb všech funkcí běžících na pozadí, v hlavním programu a blokovacích funkcí.

Na CD existuje několik ukázkových programů pro režim master – podrobně je probereme v následující kapitole.

4.7. Ukázkové programy

CD obsahuje docela dost krátkých ukázkových programů, které demonstrují základní funkčnost robotu. Většina těchto příkladů je skutečně jednoduchá a bez složitě optimalizovaných řešení. Předpokládá se, že většina příkladů bude výchozím bodem pro vaše vlastní programy. Původní záměr je ponechat některé zajímavé úlohy na vás – můžete klidně znuděně nahrát předem sestavené programy do robotu, nebo ne?

Několik ukázkových programů je více či méně určeno pro zkušené uživatele. Do této kategorie patří zejména programy pro ovládání robotu, které umožňují simulaci chování některých zvířat. S jedním takovým programem se RP6 chová jako můra, která hledá nejjasnější zdroj světla a přitom se vyhýbá překážkám. Vysvětlení detailů takového programu však překračuje rozsah této příručky a u všech dokonalých aplikací se musíte obrátit na příslušnou literaturu.

Samozřejmě si můžete na internetu vyměňovat vlastní programy s ostatními uživateli. Knihovna RP6Library a všechny ukázkové programy jsou uvolněny pod open source licencí “GPL” (General Public License). Ta umožňuje libovolnou modifikaci a zveřejnění odvozených programů podle pravidel GPL.

Rodina mikroprocesorů AVR je velmi populární a existuje spousta ukázkových programů pro MEGA32, které jsou volně dostupné na internetu. Musíte však mít stále na paměti, že se tyto programy musí upravit na hardwarovou platformu RP6 a knihovnu RP6Library. Jinak programy obvykle nefungují (běžné problémy tvoří jiné uspořádání vývodů, odlišné používání hardwarových modulů například časovačů, jiná frekvence hodin atd.).

Výjimku tvoří aplikace sběrnice I^2C, všechny ukázkové programy byly navrženy tak, aby běžely pouze na základní jednotce robotu – bez rozšiřovacích modulů. Přestože obvykle nic neruší, měli byste rozšiřující modul používat až po odzkoušení všech ukázkových programů a dokonalém seznámení se základní jednotkou.

Každá programovatelná rozšiřující stavebnice je dodávána s příslušnými ukázkovými programy. Změny a další software mohou být dostupné také na domácí stránce výrobce (například programy pro rozšiřující modul RP6 CONTROL M32 jsou obsaženy na CD RP6).

V některých případech může být programování rozšiřujících modulů (například RP6-M32) snadnější, protože se nemusíte zabývat časově kritickými záležitostmi, jako jsou ACS, regulace motoru atd.

Dále RP6-M32 poskytuje výkon a paměť CPU pro řešení náročnějších úloh.

```
Příklad 1: “Hello World”- Program funguje jako běžící LED světlo
Adresář: <RP6Examples>\RP6BaseExamples\Example_01_LEDs\
Soubor: RP6Base_LEDs.c

Program vyšle výstupní hlášení na sériové rozhraní, proto byste měli robot připojit
k PC a sledovat výstup na terminálu v programu RP6Loader.

V tomto ukázkovém programu se robot nepohybuje. Robot můžete položit na stůl vedle
počítače.
```

Tento program vyšle krátký text “Hello World” přes sériové rozhraní a pak řeší běžící světlo pomocí LED na hlavní desce.

Drobný detail je, že se světlo pomocí operátoru “posuv doleva” – který se použil v předchozích příkladech bez vysvětlení:

```
1  setLEDs(runningLight); // Set the LEDs
2  runningLight <<= 1;    // Next LED (shift-left Operation)
3  if(runningLight > 32)  // Last LED?
4      runningLight = 1;  // Yes, so, let's start again!
```

Význam operátoru vysvětlíme hned. Základ operace posun doleva “<<” (viz řádek 2) je umožnění posunu bitů v proměnné o definovaný počet míst doleva. Samozřejmě můžeme také používat ekvivalentní operátor posun doprava “>>”.

To znamená, že výraz `runningLight <<= 1;` posouvá všechny bity v proměnné `runningLight`
o jedno místo doleva. Mimochodem: je to jen zkratka výrazu:

```
runningLight = runningLight << 1;
```

která stejně funguje také u výrazů += a *=.

Proměnná runningLight začíná hodnotou 1, která znamená, že je nastaven pouze bit číslo 1. Každá operace posun způsobí pohyb bitu o jeden krok doleva.

Výsledkem používání proměnné pro ovládání LED bude “posouvání” světelného bodu --> běžící světlo! Aby lidské oko vnímalo efekt jako běžící světelný bod, používá se funkce mSleep, která mezi jednotlivými průchody smyčkou generuje zpoždění 100ms.

Pokud se v proměnné runningLight nastaví bit 7 (což znamená, že hodnota > 32), měli bychom se vrátit na začátek jednoduchým nastavením bitu 1 v proměnné (na řádcích 3 a 4).

```
Příklad 2: Další aplikace sériového rozhraní
Adresář: <RP6Examples>\RP6BaseExamples\Example_02_UART_01\
Soubor: RP6Base_SerialInterface_01.c

Tento program vyšle hlášení na sériové rozhraní

V tomto ukázkovém programu se robot nepohybuje!
```

Tento ukázkový program demonstruje používání funkcí writeInteger a writeIntegerLength, které se použily k výstupu několika celočíselných hodnot v různých formátech přes sériové rozhraní.

Navíc je zde demonstrována nová proměnná “timer” zavedená do knihovny RP6Lib ve verzi 1.3. Může se používat pro měření času s rozlišením 100µs.

```
Příklad 3: Program otázka-a-odpověď
Adresář: <RP6Examples>\RP6BaseExamples\Example_02_UART_02\
Soubor: RP6Base_SerialInterface_02.c

Tento program bude vysílat hlášení na sériovou linku

V tomto ukázkovém programu se robot nepohybuje!
```

Poněkud složitější příklad je krátký dialog otázka-a-odpověď, ve kterém robot položí čtyři jednoduché otázky, a vy byste měli na terminálu odpovědět vložením libovolné odpovědi. Robot zareaguje textovou zprávou nebo spuštěním krátkého běžícího světla. Program demonstruje, jak se přijímají data ze sériového rozhraní a použití těchto dat k dalšímu zpracování. Dále se naučíte používání funkce writeStringLength a je zde také další příklad konstrukce switch-case.

Starší funkce pro příjem dat přes sériové rozhraní nahradily v poslední verzi RP6Lib, mnohem výkonnější. Tento ukázkový program demonstruje jejich přímé použití. Nyní je možné zpracovávat vstupy s libovolnou délkou a tím lépe reagovat na špatné vstupy.

```
Příklad 4: Ukázkový program Stopky
Adresář: <RP6Examples>\RP6BaseExamples\Example_03_Stopwatches\
Soubor: RP6Base_Stopwatches.c

Tento program vysílá hlášení na sériové rozhraní

V tomto ukázkovém programu se robot nepohybuje!
```

V tomto programu se používají čtvery stopky. První z nich vytváří běžící světlo s obnovovacím intervalem 100ms a další se použily pro tři zvyšování počítadel v různých intervalech a jejich hodnoty se vysílají přes sériové rozhraní.

```
Příklad 5: Ukázkový program ACS & Nárazníky
Adresář: <RP6Examples>\RP6BaseExamples\Example_04_ACS\
Soubor: RP6Base_ACS.c

Tento program vysílá hlášení na sériové rozhraní

V tomto ukázkovém programu se robot nepohybuje!
```

Přestože název souboru naznačuje pouze aplikaci ACS, demonstruje tento program používání nárazníků
a ACS, pomocí příslušných obsluh událostí. Příklad ukazuje stav obou kanálů ACS na LED a výstupu na sériovém rozhraní. Stav nárazníků se pouze přenáší přes sériové rozhraní.

Stačí, když vložíte ruku na přední část robotu a stisknete nárazníky!

Pomocí programu můžete otestovat ACS několika předměty – zkontrolujte, jak dobře se detekují jednotlivé předměty. Můžete také změnit vysílací výkon ACS. Výchozí hodnota by měla být nastavena na střední úroveň.

```
Příklad 6: Robot se pohybuje po kružnici

Adresář: <RP6Examples>\RP6BaseExamples\Example_05_Move_01\

Soubor: RP6Base_Move_01.c

POZOR: Robot se bude v tomto programu pohybovat! Po nahrání programu odstraňte kabel
mezi PC a robotem. Robot položte na velkou volnou plochu! Musíte poskytnout volný
prostor 1m x 1m nebo raději 2m x 2m.
```

Robot je konečně připraven spustit svoje motory! Ukázkový program nechává robot jezdit po kružnici tím, že motory běží s různou rychlostí. Pro pohyb robotu poskytněte v tomto i všech následujících programech dostatečně velký volný prostor. Některé programy vyžadují volnou plochu asi 1 nebo 2 čtvereční metry.

Pokud robot během pohybu po kružnici narazí do překážky, zastaví se a začnou blikat dvě LED! Program teď čeká na restart.

```
Příklad 7: Robot se pohybuje dopředu a dozadu – s otočkou o 180°
Adresář: <RP6Examples>\RP6BaseExamples\Example_05_Move_02\
Soubor: RP6Base_Move_02.c

POZOR: Robot se bude v tomto programu pohybovat!
```

V předchozí kapitole jsme již předvedli malé příklady pohybů ovládaných časem. Na rozdíl od nich nechá tento příklad robot jet dopředu na určitou vzdálenost, otočí se o 180°, jede zpátky na stejnou vzdálenost a provede druhé otočení o 180° a opět začne od začátku.

Příklad používá blokovaný režim funkcí pro jízdu a otáčení, který bude automaticky volat task_RP6System. Proto nemusíme tuto funkci volat opakovaně.

```
Příklad 8: Pohyb robotu po čtverci
Adresář: <RP6Examples>\RP6BaseExamples\Example_05_Move_03\
Soubor: RP6Base_Move_03.c

POZOR: Robot se bude v tomto programu pohybovat!
```

Nyní se robot pokusí pohybovat po čtverci s délkou strany 30cm. Po dokončení jízdy se otočí a pohybuje se opačným směrem.

To bude fungovat pouze s přesně kalibrovanými enkodéry, jinak nemusí být pravé úhly přesné (tj. například 80° nebo 98°). Příklad tedy předvádí také složitosti při přesném ovládání pásového vozidla se zpětnou vazbou jen od signálu z enkodérů. V kapitole 2 jsme již tento problém probírali. V závislosti na povrchu podlahy budou pásy klouzat a smýkat, což sníží skutečnou vzdálenost v porovnání se změřenou hodnotou. Správné programování umožňuje kompenzaci těchto chyb, ale raději se nesnažte dosáhnout 100% přesnost pouze pomocí enkodérů. Pro přesné ovládání polohy by se měly použít externí snímače.

```
Příklad 9: Exkurze – konečný stavový automat
Adresář: <RP6Examples>\RP6BaseExamples\Example_05_Move_04_FSM\
Soubor: RP6Base_Move_04_FSM.c

Tento program bude vysílat hlášení na sériové rozhraní

V tomto ukázkovém programu se robot nepohybuje!
```

U složitějších programů nemusí být probrané metody dostatečné. Například když ovládání chování robotu vyžaduje takzvaný konečný stavový automat (zkráceně FSM) – zřejmě nebude možné snadno vyřešit pomocí jednoduchých metod. Tento ukázkový programu demonstruje jednoduchý konečný stavový automat, který změní svůj stav při stlačení nárazníku. Pro pochopení odzkoušejte činnost programu tak, že dvakrát stisknete nárazník a přitom budete pečlivě sledovat terminál a stav LED. Stiskněte nárazník a pak ho pomalu pusťte!

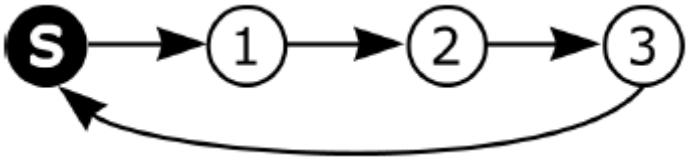
V jazyku C se konečný stavový automat vytvoří pomocí konstrukcí switch-case nebo alternativně skupinou podmíněných příkazů vytvořenou pomocí konstrukce “if-else-if-else”... Větvení pomocí switch/case přinese mnohem přehlednější zdrojový text.

Podívejte se na jednoduchý příklad:

```
#include "RP6RobotBaseLib.h"

#define STATE_START  0
#define STATE_1      1
#define STATE_2      2
#define STATE_3      3

uint8_t state = STATE_START;

void simple_stateMachine(void)
{
   switch(state)
   {
      case STATE_START: writeString("\nSTART\n");   state = STATE_1;
        break;
      case STATE_1:     writeString("State 1\n");   state = STATE_2;
        break;
      case STATE_2:     writeString("State 2\n");   state = STATE_3;
        break;
      case STATE_3:     writeString("State 3\n");   state = STATE_START;
        break;
   }
}

int main(void)
{
   initRobotBase();
   while(true)
   {
       simple_stateMachine();
       mSleep(500);
   }
   return 0;
}
```

Ukažme si tento příklad zredukovaný do nezbytného základního principu. Stavový automat obsahuje různé stavy a přechody mezi nimi. V našem příkladu máme čtyři stavy: STATE_START a STATE_1 až SATE_3.

Stavový automat můžeme znázornit také následujícím stavovým diagramem:



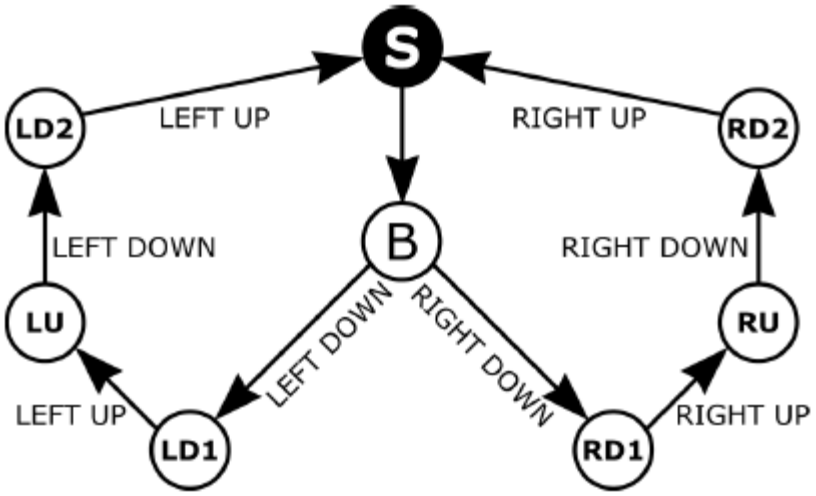
“S” je počáteční stav. Není zde použitý žádný podmíněný stav, proto bude systém postupně měnit stavy až do stavu 3 a restart v kterémkoliv kroku začne posloupnost od stavu S. Pro lepší vizualizaci je mezi změny jednotlivých stavů vloženo zpoždění 500ms.

Program bude vytvářet následující výstup:

```
START
Stav 1
Stav 2
Stav 3


START
Stav 1
Stav 2
...
... atd.
```

Program může tento výpis prodlužovat do nekonečna.

Ukázkový program v souboru `RP6Base_Move_04_FSM.c` obsahuje mnohem složitější stavový automat, představovaný 8 stavy. Základní strukturu ukazuje následující stavový diagram (v tomto přehledném schématu zkracujeme označení jednotlivých stavů):



Stavový automat začíná ve stavu S a okamžitě přechází do stavu B (přitom zobrazí krátkou textovou zprávu). Po přechodu do stavu B, bude systém čekat než se stiskne jeden z nárazníků. Pokud se stiskne levý nárazník, přejde stavový automat do stavu LD1 (“Levý dole 1”) a při stisknutí pravého nárazníku do stavu RD1. Následující přechod je podmíněný opět uvolněním nárazníku. Pokud se tak stane, změní se stav na některý ze stavů LU nebo RU. V obou těchto stavech bude automat reagovat pouze na některý mikrospínač nárazníků (levý nebo pravý) a naopak ignorovat ostatní aktivity systému. Pouze když znovu stisknete vybraný nárazník, přejde stavový automat do stavů RD2 respektive LD2. Opakované uvolnění nárazníku vrátí systém do stavu S.

Tento ukázkový příklad bude samozřejmě vysílat příslušnou zprávu při každé změně stavu a podle toho nastaví stav LED, ale v diagramu není místo pro další informace. Ukazuje jen obecnou vrstvu konečného stavového automatu.

```
Příklad 10: Konečný stavový automat, část 2
Adresář: <RP6Examples>\RP6BaseExamples\Example_05_Move_04_FSM2\
Soubor: RP6Base_Move_04_FSM2.c

POZOR: Robot se bude v tomto programu pohybovat!
```

Nyní vyzkoušíme konečný stavový automat s pohyblivým robotem! Program je skutečně jednoduchý. Nejdříve bliká LED 5. Přitom robot po nás požaduje: “Můžete, prosím, někdo stisknout levý nárazník?”. Když to uděláte, přesune se RP6 asi o 15 cm dozadu a začne blikat stavová LED 2. Obvykle nyní předpokládáte, že někdo stiskne pravý nárazník, který způsobí, že se robot opět přesune 15 cm dopředu a cyklus se spustí od začátku.

Struktura konečného stavového automatu není nijak zvláštní. Je velmi podobná prvnímu příkladu, který jste viděli výše, kde bylo v programu napsáno “START Stav1 Stav2...”, ale je rozšířen přidáním několika podmíněných výrazů. V případě, že robot čeká, než se dokončí pohyb, používají dva stavy volání funkce “isMovementComplete()”. Uvnitř těchto stavů, program dále provádí jednoduché části kódu, kde pomocí stopek přepíná LED v intervalu 500ms. Tato část může být samozřejmě nahrazená libovolnou posloupností kódu – třeba běžícím světlem z příkladu 1.

Můžeme popsat řadu stránek o konečných stavových automatech a podobných tématech, ale jak jsme již uvedli – tento materiál je pouze příručka a ne referenční kniha konečných stavových automatů. A tak raději přejdeme k dalším ukázkovým programům…

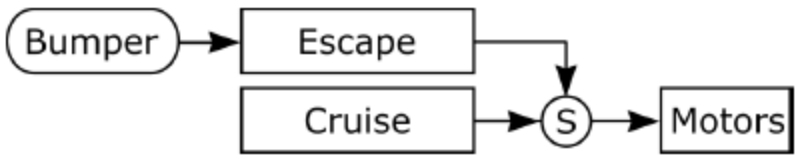
```
Příklad 11: Řízené chování robotu
Adresář: <RP6Examples>\RP6BaseExamples\Example_05_Move_04\
Soubor: RP6Base_Move_04.c

POZOR: Robot se bude v tomto programu pohybovat!
```

Předchozí příklady automatů vedou k následujícímu programu, který implementuje jednoduché řízení chování robotu. Pro zjednodušení používáme pouze dva způsoby chování. V dalších příkladech budeme základní program postupně rozšiřovat, až se vytvoří jednoduché chování hmyzu. První krok vytvoří náš “hmyz” pouze se dvěma malými snímači, které hlásí kolize.

První dva způsoby chování se nazývají “Projížďka” a “Únik”. Chování “Projížďka” pouze říká robotu “jeď dopředu” a nedělá žádné další aktivity. Samozřejmě můžeme přidat další věci, tj. občasné projíždění zatáček nebo zrychlování a zpomalování podle napětí baterie. Kvůli modularitě strukturovaného programu radíme raději navrhnout samostatný program, který bude zahrnovat toto chování.

Na rozdíl od “Projížďky” je chování “Únik” docela složité a bude aktivní, jakmile nárazníky zjistí kolizi. Podle toho, který nárazní se stlačí, robot couvne o několik centimetrů zpět, natočí se a následně se vrátí k řízení chování “Projížďka”.

Tuto strukturu můžeme vyjádřit jednoduchým diagramem:



Pomocí “Supresor” S chování “Únik” potlačí výstup chování “Projížďka” a systému vnutí řízení motorů vlastními povely. Obvykle má “Únik” vyšší prioritu než “Projížďka”.

Pomocí těchto dvou velmi elementárních chování, robot obvykle začne projížďkou a bude zkoumat okolní prostředí. Jednoduchý senzorický systém tvoří dva snímače pro detekci kolizí. Samozřejmě se nejedná o příliš složité chování.

Představte si situaci, ve které se můžete pohybovat uvnitř vašeho domu a vaše vnímání se omezí na dva prsty – žádné oči, žádné uši, žádné další smysly nepomáhají, máte jen tyto dva prsty, kterými se můžete stále dotýkat pouze věcí přímo před sebou...

Více snímačů umožní složitější chování robotu. Zvířata jsou obvykle vybavena velkou spoustou snímačů, které mohou pomocí analogových hodnot rozlišit intenzitu vněmu. Můžete si jednoduše představit jak se zvýší "inteligence", když se robot vybaví několika zvířecíma očima...

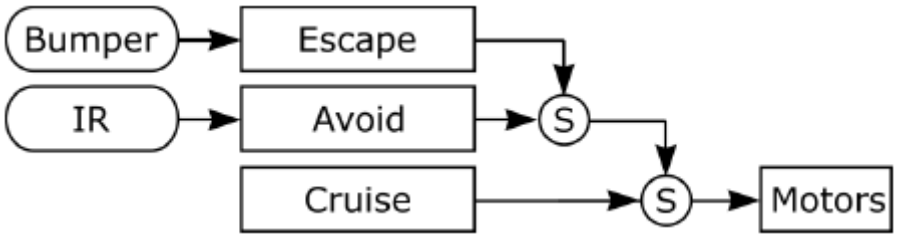
Zde použijeme postup, který zavedl Rodney Brooks (http://people.csail.mit.edu/brooks/) kolem roku 1985, nazývaný architektura Subsumption.

Původní publikace můžete najít zde:
http://people.csail.mit.edu/brooks/papers/AIM-864.pdf
http://people.csail.mit.edu/brooks/papers/how-to-build.pdf

ale můžete najít další relevantní dokumenty a souhrny (internet poskytuje mnoho dalších informací – stačí jen spustit prohledávání!)

```
Příklad 12: Řízené chování robotu 2
Adresář: <RP6Examples>\RP6BaseExamples\Example_05_Move_05\
Soubor: RP6Base_Move_05.c

POZOR: Robot se bude v tomto programu pohybovat!
```

Následující krok přidá chování "Vyhýbání", které místo kolizí používá ACS (proti kolizní systém) k vyhýbání překážkám. Strukturu se třemi typy chování ukazuje následující obrázek:



Kolize se samozřejmě musí chápat jako velmi důležité události a proto má chování "Únik" stále nejvyšší prioritu. "Únik" potlačí příkazy "Vyhýbání" a podobně "Vyhýbání" potlačí příkazy "Projížďky".

Jakmile infračervené snímače detekují překážku, chování "Vyhýbání" – jak již název naznačuje – spustí manévr vyhýbání. Pokud překážku zjistil levý kanál ACS, zatočí robot doprava a u pravého kanálu zatočí doleva. Pokud překážku detekují oba kanály současně, bude se robot otáčet doleva nebo doprava podle toho, který kanál podal zprávu jako první. Robot se bude otáčet o něco déle, než oba kanály opět ohlásí volný prostor. Pro řízení této periody zpoždění se používají stopky.

Příklad 13: LDR – snímače osvětlení
Adresář: <RP6Examples>\RP6BaseExamples\Example_06_LightDetection\
Soubor: RP6Base_LightDetection.c

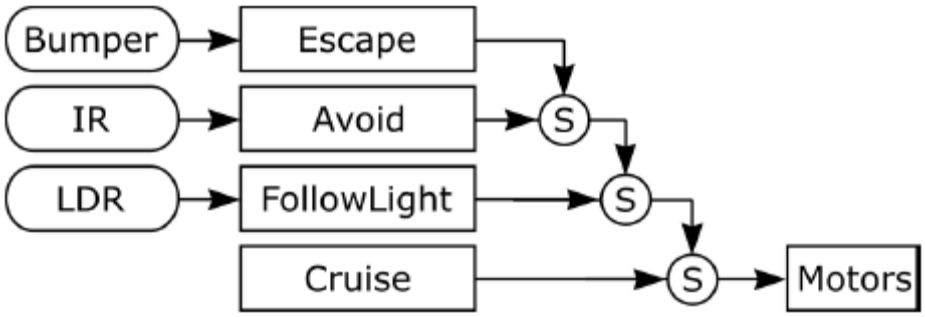
Tento program bude vysílat hlášení na sériové rozhraní.
V tomto ukázkovém programu se robot nepohybuje.

Tento ukázkový program demonstruje, jak se používají dva světelné snímače. Pro indikaci snímače, na který dopadá více světla (nebo zda je osvětlení obou snímačů stejné) používáme stavové LED. Vedle toho bude program také neustále hlásit tyto informace na sériové rozhraní.

Příklad 14: Řízené chování robotu 3
Adresář: <RP6Examples>\RP6BaseExamples\Example_07_LightFollowing\
Soubor: RP6Base_LightFollowing.c

POZOR: Robot se bude v tomto programu pohybovat!

Snímače osvětlení se mohou samozřejmě použít k rozšíření návrhu robotu implementací chování, které nazveme “Sleduj světlo”:



Priorita “Sledování světla” je pod Únikem a Vyhýbáním, ale nad Projížďkou, která zahrnuje chování “Projížďka”, které v tomto příkladu nebude nikdy aktivní, pokud je místnost velmi tmavá (jakmile úroveň obou LDR klesne pod 100, zruší se okamžitě chování “Sleduj světlo”).

Chování “Sleduj světlo” se pokouší, pomocí světelně závislých rezistorů, sledovat jasné světelné zdroje nebo vyhledat nejjasnější svítící bod v místnosti. Jednoduchý algoritmus a jednoduché uspořádání snímačů může samozřejmě ve zvláštních podmínkách selhat – tj. pokud je robot konfrontován s velkým počtem stejných světelných zdrojů. Tento systém však může být velmi užitečný ve tmavé místnosti, pokud se snaží najít velmi intenzivní světlo baterky.

Program používá LED v duálním režimu – pokud ACS nehlásí překážku, indikují LED, který světelný snímač detekuje větší intenzitu osvětlení a jakmile ACS ohlásí překážku, signalizuje to také LED.

Tento program dokončuje přehled ukázek řízeného chování robotu simulací jednoduchého chování hmyzu. Robot se při hledání a sledování světelných zdrojů chová stejně, jako můra a zároveň se vyhýbá překážkám.

Robot se může dále rozšiřovat pomocí dalších snímačů na rozšiřujících modulech a programováním vlastního chování a zdokonalováním stávajících funkcí. Tyto možnosti velmi závisí na vaší kreativitě a programátorských schopnostech.

Příklad 15: Dálkové ovládání pomocí univerzálního IR dálkového ovladače RC5
Adresář: <RP6Examples>\RP6BaseExamples\Example_08_TV_REMOTE\
Soubor: RP6Base_TV_REMOTE.c

POZOR: Robot se bude v tomto programu pohybovat!

Tento ukázkový program vám dovolí ovládat robot podobně jako dálkově ovládané autíčko pomocí běžného televizního dálkového ovladače s kódováním RC5. Ve srovnání s běžnou funkčností většiny RC autíček, přidáme několik zvláštních pohybů.

Vozidlo se samozřejmě může pohybovat dopředu a dozadu, ale je také schopné na místě zatáčet doleva a doprava. Dále může robot projíždět levou i pravou zátočinu směrem dopředu i dozadu. Může také spustit jediný motor směrem dopředu nebo dozadu.

Pro zajištění flexibility programu, můžeme zpřístupnit všechny pohybové povely na libovolné kódy tlačítek konkrétního dálkového ovladače. Pohyby se spustí, když se přijme konkrétní kód tlačítka a rychlost motoru se pomalu zvyšuje. Když uvolníme tlačítko, rychlost motoru zase pomalu klesá. To znamená, že robot pomalu zrychluje nebo zpomalujete (a nemusí vás to znepokojovat). Stisknutím speciálního tlačítka je také možné robot okamžitě zastavit na jednom nebo dvou centimetrech (záleží to na zatížení a rychlosti robotu).

Příjem kódu RC5 se může používat pro další různé funkce – tj. spouštění různých programů, nastavení parametrů chování, modifikaci parametrů řízení v regulaci rychlosti atd...

S některými univerzálními dálkovými ovladači je možné řídit několik robotů současně pomocí speciálních funkčních tlačítek, které vybírají různá zařízení – tato funkční tlačítka se mohou naprogramovat na vysílání signálů RC5 s adresami různých zařízení – to dovoluje současné řízení několika robotů z jediného dálkového ovladače.

Příklad 16: Rozhraní sběrnice I^2C – režim Master
Adresář: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_01\
Soubor: RP6Base_I2C_MASTER_01.c

Tento program demonstruje, jak se používá režim Master sběrnice I^2C. Před spuštěním programu musíte mít samozřejmě na sběrnici I^2C připojené vhodné zařízení typu slave.

Tento program implementuje pomocí 8 LED jednoduché běžící světlo “Knight Rider”. LED jsou připojené ke standardnímu 8-bitovému PORT expandéru PCF8574 na sběrnici I^2C. Obvod PCF8574 můžeme vložit a zapájet na experimentální rozšiřující modul (nebo zpočátku otestovat sestavení obvodu na nepájivém kontaktním poli).To již poskytuje systém s 8 volnými I/O porty pro odzkoušení číslicových snímačů nebo alternativně ovládání malých zátěží jako jsou LED. Velká zátěž samozřejmě vyžaduje samostatné externí tranzistory nebo raději spínací obvody.

Tento obvod je velmi užitečné zařízení a na stejné sběrnici můžete použít několik obvodů. Jediné co musíte udělat, je vybrat, pomocí tří adresových vývodů, správnou adresu jednotlivých obvodů. Pokud chcete použít více jak 8 obvodů, musíte použít 8 normálních obvodů PCF8574 a dalších osm PCF8574A, které mají jinou základní adresu. To vám umožňuje adresovat 8 obvodů každého typu pro řízení celkem 16*8 = 128 I/O vývodů portu přes jedinou sběrnici I^2C.

```
Příklad 17: Rozhraní sběrnice I2C – režim Master
Adresář: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_02\
Soubor: RP6Base_I2C_MASTER_02.c

Tento program demonstruje, jak se používá režim Master sběrnice I²C. Před spuštěním
programu musíte mít samozřejmě na sběrnici I²C připojené vhodné zařízení typu slave.
```

V tomto ukázkovém programu přidáme rutiny pro PCF8591. Obvod PCF8591 obshuje 8-bitový analogově/číslicový převodník (ADC) se čtyřmi kanály a číslicově/analogový převodník (DAC) pro generování analogových napětí. To znamená, že se obvody PCF8574 a PCF8591 perfektně doplňují.

Obvod PCF8591 umožňuje sledování čtyř různých napětí – náš příklad byl navržen pro vyhodnocení dalších čtyř LDR (světelně závislých rezistorů zapojených jako děliče napětí). Současné zapojení obvodů je skutečně zanedbatelné – můžeme použít čtyři infračervené snímače vzdálenosti Sharp GP2D120, nějaké snímače teploty nebo další podobná zařízení.

Hlavní nevýhodou používání těchto obvodů je, že příjem naměřených údajů přes sběrnici I²C, zabírá více času v porovnání s integrovaným ADC nebo I/O portem. Toto omezení obou obvodů se zjednoduší použitím v časově nenáročných aplikacích. Kdykoliv potřebujete rychlou odezvu a řízení systému, můžete zvážit použití dalšího mikroprocesoru. Druhý volně programovatelný mikroprocesor obvykle zkomplikuje systém, ale vytvoří také větší flexibilitu.

Na CD jsme vložili katalogové listy obvodů PCF8574 a PCF8591.

```
Příklad 18: Rozhraní sběrnice I2C – režim Master
Adresář: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_03\
Soubor: RP6Base_I2C_MASTER_03.c

Tento program demonstruje, jak se používá režim Master sběrnice I²C. Před spuštěním
programu musíte mít samozřejmě na sběrnici I²C připojené vhodné zařízení typu slave.
```

Tento jednoduchý program demonstruje jak se přes sběrnici I²C řídí ultrazvukový snímač vzdálenosti Devantech SRF08 nebo SRF10. Program se může samozřejmě přizpůsobit na používání podobných snímačů od jiných výrobců.

```
Příklad 19: Rozhraní sběrnice I2C – režim Mode
Adresář: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_04\
Soubor: RP6Base_I2C_MASTER_03.c

Tento program demonstruje, jak se používá režim Master sběrnice I²C. Před spuštěním
programu musíte mít samozřejmě na sběrnici I²C připojené vhodné zařízení typu slave.
```

V tomto ukázkovém programu se řídí čtyři I²C zařízení typu slave: dva SRF08/SRF10, jeden PCF8574 a jeden PCF8591. Program využívá kód ze třech předchozích příkladů.

Další programové příklady periferních obvodů I²C budou dodány společně s příslušnými rozšiřujícími moduly.

Příklad 20: Rozhraní sběrnice I2C – režim Slave
Adresář: <RP6Examples>\RP6BaseExamples\Example_I2C_SLAVE\
Soubor: RP6Base_I2C_SLAVE.c

Na začátku program nebude předvádět žádnou viditelný výstup. K robotu musíte přidat rozšiřující modul, který dokáže ovládat robot a chová se jako I^2C master

Systém vytváří senzorický systém robotu pomocí několika přídavných mikroprocesorů. Ve většině případů je to dobrým řešením ponechat jeden přídavný mikroprocesor na řízení celého robotu. Nemusí to být větší a rychlejší mikroprocesor, ale stačí další MEGA32. Mikroprocesor na hlavní desce již provádí několik úkolů jako ACS, regulace pohonu atd. a to jistě spotřebuje určitý výpočetní čas (řadu událostí přerušení). Externí mikroprocesor bude mít mnohem více nevyužitého času.

Sestavení programu se řídí myšlenkou: master řídí horní úroveň veškeré funkcionality robotu a přes sběrnici I^2C posílá příkazy do jednotky slave, která obsluhuje nízkou úroveň řízení. Všechny známé funkce jako automatická regulace pohybu se nyní jeví jako skutečná výhoda – master pouze přenáší krátké povely typu “jeď dopředu na <parametry rychlosti>” a vše zbývající se provede automaticky. Některé běžící úlohy se mohou samozřejmě zhroutit, například když snímače detekují překážky.

Zdrojový text programu obsahuje seznam všech platných příkazů a příslušných parametrů. Dále zdrojový text programu obsahuje ostatní detaily potřebné pro regulaci.

Pokud je třeba, mikroprocesor ve slave automaticky inicializuje signál přerušení na první linku externího přerušení (INT1). To se může Master využít k rychlé a snadné detekci stavu snímačů nebo pohonu.

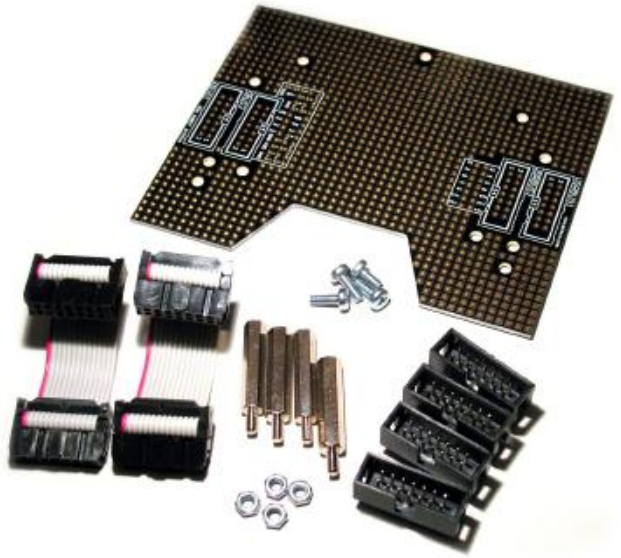
Hodnoty snímačů se mohou číst z řady registrů. Pokud se spustí událost přerušení, může master začít číst některé stavové registry a kontrolovat, co způsobilo přerušení. Později může číst další související registr.

Alternativně samozřejmě můžete kdykoliv jednoduše číst VŠECHNY registry snímačů najednou.

Podrobné informace o této problematice a samozřejmě podrobnou specifikaci registrů lze najít ve zdrojovém textu programu.

Další ukázkové programy řízení robotu budou publikovány současně s rozšiřujícími moduly. Několik takových programů a příkladů pro jednotku RP6 CONTROL M32 je již přidáno na CD-ROM.

5. Experimentální deska



Pro sestavování vašich vlastních obvodů již existuje rozšiřující experimentální modul, který se dodává společně s robotem. Modul se dodává jako stavebnice. Musíte si SPRÁVNĚ zapájet propojovací konektory. Zvláštní pozornost musíte věnovat správné polaritě konektorů – zkontrolujte bílý potisk desky plošných spojů.

Sestavování vlastních obvodů z vývodových součástek vyžaduje určitou zkušenost s pájením a elementární znalosti o elektronice. Samozřejmě musíte mít základní nápad, jak se to má udělat...

Nyní se podívejme, jaké obvody lze sestavit na experimentální desce.

V předchozí kapitole jsme již poskytli dva příklady! Pomocí popsaných expandérů portu a A/D převodníků můžete systém rozšířit o další I/O porty nebo A/D kanály. Tyto hardwarové prostředky se mohou použít jako rozhraní se snímači osvětlení, infračervenými snímači, dotykovými snímači LED a dalšími periferiemi. Systém sběrnice I^2C vám dále umožňuje vytvářet rozhraní s mnohem složitějšími senzorovými moduly, jako jsou například ultrazvukové snímače vzdálenosti, moduly elektronických kompasů, gyroskopy nebo akcelerometry. Další velmi zajímavá senzorika jsou snímače tlaku, teploty a vlhkosti, které se dají zabudovat do malé mobilní meteorologické stanice.

Systém vám v podstatě umožňuje připevnit na robot libovolný počet rozšiřujících desek. Nemusíte se omezovat na jedinou desku. Kdykoliv uvažujete o konstrukci složitějšího systému, měli byste počítat s instalací dalšího mikroprocesoru. Vhodným zařízením může být deska RP6 Control M32, která vašemu systému poskytuje zvláštní mikroprocesor, 14 volných I/O linek (včetně 6 kanálů A/D převodníku). I/O linky jsou dostupné na 10 pólových konektorech. Snadno si můžete sestavit několik plochých kabelů k propojení modulu a rozšiřujících desek. Brzy bude k dispozici rozšiřující modul "C-Control", který bude velmi vhodný pro realizaci složitých projektů.

Základní deska robotu samozřejmě také nabízí 6 rozšiřujících, které mohou být zajímavé pro osazení senzory, které potřebují být co nejníže nad podlahou (např. další IR čidla). Nicméně vám doporučujeme raději začít práci s rozšiřujícími projekty na experimentálním modulu před pájení dílů na základní desku. Experimentální deska umožňuje snadnější odstranění součástek, když se něco pokazí ...

6. Závěrečné slovo

Na tomto místě tento malý úvod do světa robotiky končí. Doufáme, že jste si naplno užili všechny aktivity s robotem RP6!

Všechny začátky jsou těžké, obzvláště v této složité oblasti, ale jakmile jste zvládli první překážky, zažijete s robotikou spoustu legrace a snadno zvládnete řadu nových věcí.

Pokud jste se spřátelili s programováním v jazyku C, můžete začít sestavovat své vlastní zajímavé projekty. Již jsme vám předložili nějaké nápady na projekty v oblasti robotiky, ale vaše vlastní představy mohou být mnohem zajímavější!

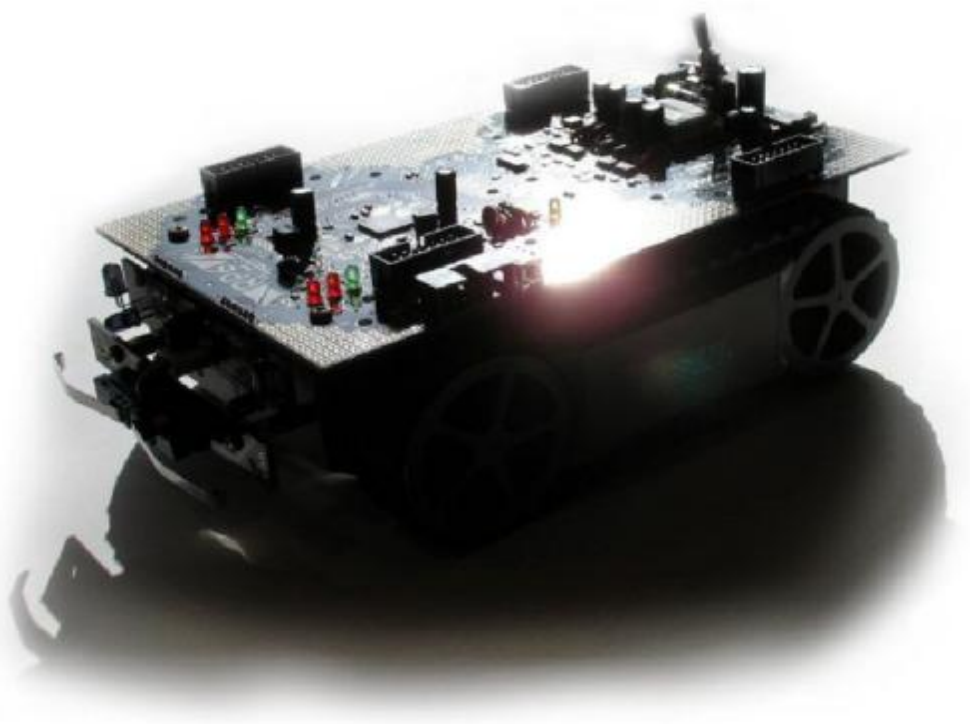
Zveme vás k diskuzi o robotice a elektronice v několika specializovaných internetových fórech. Obvykle zde bývá mnohem více zajímavých informací než v jakékoli příručce. I poměrně zdlouhavá a namáhavá práce neumožňuje, projít všechna relevantní témata a pominutí velkého množství detailů.

Samozřejmě plánujeme vývoj nějakých dalších rozšiřujících modulů pro robotický systém RP6. Již jsme uvolnili modul s experimentální propojovací deskou a řídící jednotku RP6 CONTROL M32, která poskytuje další mikroprocesor ATmega32 (včetně mikrofonního čidla, akustického piezo měniče, obrovskou externí paměti 32kB EEPROM, tlačítek, LED a LCD portu). Připravujeme také rozšiřující modul pro novější systém Conrad Electronic C-Control Modules (např. pro C-Control PRO MEGA128).

Budeme vyvíjet moduly s dalšími snímači, ale raději nebudeme prozrazovat technické podrobnosti...

Jakmile se uvolní nový moduly, oznámíme to na webových stránkách AREXX a na našem diskusním fóru!

Do té doby jistě budete zaneprázdněni samotným systémem RP6 a stávajícími možnostmi rozšiřování!



PŘÍLOHY

A – Vyhledávání a odstraňování problémů

V této příloze je souhrn řady problémů a možných příčin, řešení a návrhů, jak se těmto problémům vyhnout. Seznam může být postupně čas od času rozšiřován a aktualizován!

1. Robot nelze zapnout - žádná LED nesvítí a robot nereaguje na tlačítko Start/Stop!

- USB rozhraní je zapojeno do robotu, ale ne do PC! Toto spojení bude držet mikroprocesor MEGA32 v režimu reset, který mikroprocesoru znemožňuje bliknutí libovolné LED diody. USB rozhraní vždy nejdříve zapojte k PC a pak teprve do RP6! Stejný problém může nastat, pokud je osobní počítač vypnutý.
- Rozšiřující modul drží všechny mikroprocesory ve stavu RESET - což se může snadno stát ve vašich vlastních konstrukcích (může se to stát také v předem sestavených projektech, ale většinou se to omezuje na vady nebo softwarové chyby). Tento stav můžete vyzkoušet odstraněním všech rozšiřujících modulů!
- Mohli jste vyjmout baterii nebo používáte vybité či vadné baterie.
- Hlavní pojistka je přerušena. Pomocí ohmmetru (nebo multimetru) zkontrolujte pojistku. Docela často se může spálená pojistka snadno zkontrolovat pohledem na tenký drát uvnitř skleněného pouzdra. Pojistka se obvykle přepálí uprostřed tenkého drátu, ale někdy nelze přerušený drátek rozpoznat vizuální kontrolou a musí být se zkontrolovat multimetrem!

POZOR: Pokud se pojistka spálí, musíte zkontrolovat neregulérní podmínky v systému! Velký proud může způsobit nějaký zkrat obvodu (nezapomněli jste odstranit některé kovové nástroje nebo jiné předměty z těla robotu?) nebo jsou některé části namáhány nadměrným proudem. Zkontrolujte, prosím desku plošných spojů a všechny součásti! Možná jste upravil nějaký obvod robotu nebo jste použili špatné součástky. Dodrželi jste správnou polaritu baterie? Dávali jste pozor na zmáčknutí kabelů mezi základní deskou a rámem? Dávali jste pozor na správné připojení rozšiřujících modulů? Pokud ano, budete muset odpojit všechny rozšiřující moduly před opakovaným zahájením zkoušek! Pokud jsou všechny kontroly v pořádku a všechny rozšiřující moduly byly odstraněny, můžete vložit správnou, rychlou pojistku 2,5A a znovu zkusit funkčnost.

2. Robot odmítá spustit programy!

- Stiskli jste po nahrání programu tlačítko Start/Stop?
- Nahráli jste program do RP6 správně? Vybrali jste správný program?

3. Během pohybu robot opakovaně generuje RESET a zastavuje program!

- Baterie neposkytuje dostatečnou energii pro napájení systému!
- Baterie selhává (nekvalitní nebo příliš stará baterie) a během provozu způsobuje poklesy napětí, které automaticky vedou k vyvolání resetu. V takovém případě použijte v robotu čerstvé baterie a zkuste ho spustit znovu!
- Baterie nejsou pevně vloženy v držáku nebo je špatný kontakt v kabeláži napájení.
- Přetížení baterie může způsobovat rozšiřující modul!

4. Program RP6Loader se nedokáže připojit k robotu!

- USB propojovací kabel nebo 10-ti žilový plochý kabel mohou mít špatný kontakt (opatrně zahýbejte USB konektorem!)
- V programu RP6Loader jste zvolili nesprávný sériový port.
- Sériový port je používán jinou aplikací, proto se neukázal v seznamu portů. Zavřete všechny ostatní aplikace, které mohou používat USB porty nebo převodníky USB na sériovou linku. Následně obnovte seznam portů a znovu spusťte RP6Loader.
- V dialogu předvoleb můžete mít aktivovánu funkci “Inverze vývodu RESET” – zrušte tuto volbu!
- Robot byl vypnutý nebo je vybitá (téměř vybitá) baterie.
- Poškozený kabel nebo konektor – což je poněkud neobvyklé, ale může se to stát vlivem velkého mechanického namáhání kabelu.

5. Robot při pohybu vydává podivný hluk!

- Nezvyklý hluk může mít několik příčin. Budete muset otevřít robot, abyste zkontrolovat převodovky a motory. Nedostaly se do systému převodovky napájecí kabely? Malý hluk je zcela normální a nezpůsobí žádné problémy! Při vysokých rychlostech bude robot určitě vydávat větší hluk (ale hluk určitě nedosáhne extrémní hladinu starého předchůdce – systému CCRP5).
- Možná se uvolnil některý pojistný kroužek (montážní šroubek) v převodovce, na předních kolech nebo cokoliv jiného? Všechna kola se musí otáčet volně (pečlivě to otestujete opatrným otáčením zadními koly!), ale převodovky to neumožňují a mohou se obrousit ozubená kola!
- Hluk může snížit také několik kapek jemného oleje aplikovaných na obě kuličková ložiska každého motoru – na předním a zadním víku, kde vidíte hřídel motoru (**POZOR:** Striktně používejte, prosím, pouze speciální olej pro kuličková ložiska např. Conrad skladové číslo 223441-62 "Olej na volně běžící ložiska Team Orion #41701"). Během a po tomto postupu mazání se musí kuličková ložiska a hřídel motoru otáčet, aby se olej dostatečně rozložil po celé ploše ložiska (motor se několik sekund drží ve svislé poloze, aby olejový film natekl do ložiska)! Po této proceduře se setřou všechny zbytky oleje! Ložiska by měla být namazána při výrobě v továrně, ale mazací postup se musí v určitých intervalech obnovit!

Nemažte, prosím, tímto olejovým způsobem převodovky! To může zničit systém snímače otáčení! Přestože ropa nemůže poškodit elektronické součástky, může kapalina poškodit nalepená snímací kola, na které se nesmí dostat žádná tekutinu, protože výrazně ovlivňuje odrazivost! Navíc může otáčení ozubených kol rozstříknout tyto tekutiny a maziva kamkoliv uvnitř interiéru robota! Použití mazacího tuku (ne tekutiny!) je omezen pouze na hřídele (!) obou převodových ústrojí! Mazací tuk obvykle vůbec nezlepší hlučnost ...

6. V každém programu během chvilky rozběhnou motory až do vysoké rychlosti a pak okamžitě zastaví. Následně začnou blikat čtyři červené LED!

- Nejjednodušší možnost je: Nahrajte do robotu vlastní nebo upravený ukázkový programu a před spuštěním motorů zapomeňte zavolat funkci "PowerOn ();"!
- Provedli jste nějakou modifikaci software (zejména knihovny)? Zkuste spustit nějaké další programy a program s autonomním testem.
- Pokud se ostatní programy a autonomní test chovají podobným způsobem: zkontrolujte výstup na sériovém rozhraní! Při blikajících červených LED systém navíc vypíše chybové hlášení, které můžete sledovat na terminálu. S největší pravděpodobností může takovou poruchu způsobit systém snímače polohy (enkodéru). Někdy se může oddělit stahovací kroužek, který drží kolo snímače, což způsobí, že se ozubená kola se odsunou od snímačů. V takovém případě přestane snímací systém dodávat signály zpětné vazby! Mimochodem: snímače jsme vybavili malými potenciometry (rezistory s proměnným odporem) pro nastavení snímačů (pomocí malého šroubováku s šířkou 1.8mm šroubovák nebo vhodného křížového šroubováku! BUĎTE VŠAK VELMI OPATRNÍ: Nastavení obvykle vyžaduje použití osciloskopu! Bez něho může být kalibrace velmi obtížná. Autonomní testovací program ukáže, zda snímače fungují správně. Poskytujeme program se speciálním režimem (c – test střídy) na zevrubnou kontrolu střídy průběhu pravoúhlého signálu. Optimální je 50% střída. Přesto se za přijatelné považují odchylka a kolísání od 25 do 75% (mimochodem program může hlásit chyby, i když je vše docela normální: příčina těchto chyb bude spíše v programu, nikoliv v systému snímače polohy). Většina hodnot střídy by měla být v rozmezí 30 až 70% a test střídy by měl po většinu času hlásit stav "OK". Během zkoušek testovací program vypínal regulaci motoru a spouštěl motory na pevnou hodnotu PWM. Hodnoty měření rychlosti se zobrazí, jakmile snímač čítá impulsy po dobu 200ms.
- Možná se poškodila kola snímače polohy? Například při mazání olejem nebo mazacím tukem, jak je popsáno výše?
- Zkontrolujte, zda nejsou poškozeny kabely odměřování, např. pomocí multimetru (vypne se napájení robota a u každého jednotlivého vodiče se kontroluje spojení obou konců a zkrat na své sousední vodiče!).
- Podívejte se, zda nejsou znečištění malé citlivé plochy reflexních IR snímačů.
- LED také začne blikat, jakmile přestane fungovat je jeden motor (nebo oba motory) – proto zkontrolujte zapojení a desky plošných spojů, zejména v blízkosti budičů motoru.

7. Robot se buď nepohybuje, nebo se pohybuje pomalu a okamžitě po spuštění programu začnou blikat čtyři červené LED!

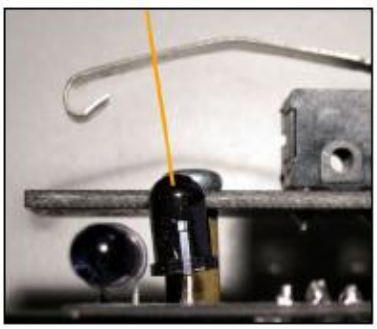
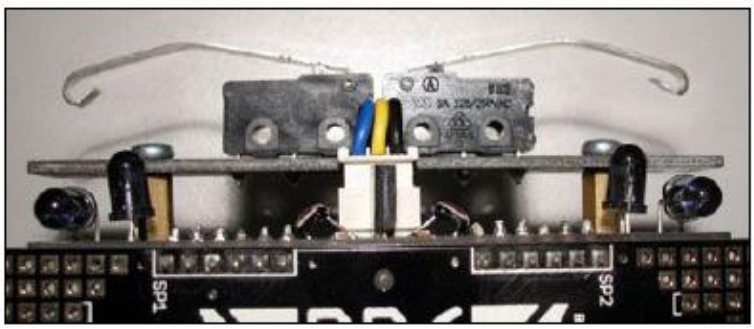
- Čtveřice LED začne blikat také při nízkém napětí baterie! Jen připojte robot k PC a podívejte se, zda se při kliknutí na “Conect“ v programu RP6Loader neobjeví výstraha nízkého napětí! Když stisknete resetovací tlačítko robota a bootloader začne čekat na startovací signál, může nízké napětí baterie způsobit blikání čtyř LED! V takové situaci je stále možné spustit program, ale robot se brzy opět zastaví a začnou blikat LED...
- Pokud je baterie skutečně úplně nabitá a stavové pole programu RP6Loader tedy hlásí, že napětí baterie překračuje 6V, zkontrolujte další chybová hlášení na terminálu. Může být zablokovaná jedna nebo obě převodovky tj. příliš dotažen upevňovací kroužek, do převodovky se dostaly napájecí kabely nebo kola převodovek drhnou o jiné mechanické části. Odstraňte překážky, opatrně uvolněte upevňovací kroužky a zajistěte je.
- V nejhorším případě mohlo dojít k poškození motorů nebo elektronických obvodů, které vyžaduje výměnu poškozených součástek... Spusťte, prosím, autonomní testovací program #8 (Během testu NEDOVOLTE, aby se robot dotýkal podlahy – pásy se musí pohybovat volně! Neblokujte a nebrzděte pásy rukou! Blokování bude mít okamžitě za následek selhání testu!). Autonomní testovací program zkontroluje funkci motorů a elektronických řídících obvodů včetně proudových snímačů. Udělal autonomní test nějaká chybová hlášení? (Doporučujeme vám okopírovat celý obsah terminálu do textového souboru, např. pomocí menu "Nastavení -> Uložit terminál").

8. Moje nabíječka nedokáže nabít baterie uvnitř robotu!

- Máte nabíječku připojenu správně? Zkontrolujte polaritu kabelů a konektorů.
- Máte správně vložené baterie? Zkontrolujte poškození nebo uvolnění kontaktů. Zkontrolujte polaritu kabelů, konektorů a baterií.
- Konektor nabíjení a desce plošných spojů možná nemá vnitřní kolík stejný jako zástrčka na kabelu nabíječky? (Existují různé verze těchto konektorů, které mohou být snadno zaměněny. Standardní nabíječky jsou dodávány s velkým souborem různých konektorů).

9. ACS nedokáže spolehlivě rozpoznávat překážky!

- Vyzkoušeli jste již různé rozsahy nastavení výkonu?
- Jsou IR LED nebo IR přijímač pokřivené? Zkontrolujte vyrovnání a seřízení IR LED a IR přijímače. IR LED musí být nastaveny tak, aby směřovaly téměř přímo ze senzorové desky, ale měla by být vykloněny ven přibližně o úhel 5 - 10°. Infračervený přijímač musí být osazen přesně 90° směrem nahoru (viz fotografie na této stránce a v kapitole 2).


- ACS bude často špatně rozpoznávat černé nebo velmi tmavě překážky. Černé objekty mají více či méně tendenci zcela absorbovat IR světlo. Toto je normální fyzikální chování! Pro spolehlivou detekci objektů můžete navíc použít jiné typy snímačů (například ultrazvukový měřič vzdálenosti...)!
- Robot se provozuje v prostředí s jasným osvětlením, např. s přímým slunečním světlem, zářivkami nebo na podsvícením plochých obrazovek. Tyto světelné zdroje mohou rušit IR-komunikaci a ACS.

10. IRCOMM není schopen přijímat signály z dálkového ovladače! Robot nereaguje na tyto signály!

- Vysílá váš dálkový ovladač signály skutečně ve formátu RC5? Pokud to nevíte jistě, tak pravděpodobně ne. Software rozpozná pouze formát RC5. Dálkový ovladač budete muset přepnout do jiného systému kódování (většina univerzálních dálkových ovladačů umožňuje nastavit formát RC5) nebo použít jiný dálkový ovladač.
- Přiřazení kláves se liší typ od typu a také podle výrobce. Vždy začněte přiřazením tlačítek pro váš program. Komentáře ve zdrojovém kódu ukázkového programu RC5 vám poskytnou pokyny, jak přiřadit tlačítka jednotlivým funkcím. Zkontrolujte výstupní kódy na terminálu, který bude zobrazovat kódy jednotlivých tlačítek pro ukázkový program RC5!

11. Robot se nedokáže otočit o zadaný úhel.

- To je normální chování a v příručce již byly vysvětleny příčiny těchto odchylek! Jednotka pásového pohonu vždy způsobí odchylky směru díky proklouzávání a posouvání. Občas se musí také kalibrovat snímače polohy (enkodéry), aby se zajistila správná funkce podle představ. Podrobnosti viz příloha B.

12. Proč i extrémně krátký program zabírá 7 kB paměti?

- Každý program vždy obsahuje knihovnu RP6Library, která bude zabírat více než 6,5 kB paměti, takže je nutné uvažovat za standardní velikost programu paměťový prostor 7 kB. Brzy zjistíte, že se velikost programu nezvětší, i když se přidá velký kus zdrojového kódu. Nebojte se velikosti paměťového prostoru. Paměť poskytuje dostatek místa pro běžné programy. Pokud velikost vašeho programu přesahuje hranice paměťového prostoru, můžete zvážit použití rozšiřujícího modulu s dalším mikroprocesorem.

13. Nelze kompilovat moje programy – kompilátor skončí překlad s nějakými chybovými zprávami!

- Zkontrolujte chybovou zprávu a pokuste se pochopit, co chybu způsobilo. Pokud máte kontaktovat technickou podporu, nejprve zkopírovat celé hlášení kompilátoru do textového souboru. Pak zašlete hlášení kompilátoru a zdrojový soubor do střediska technické podpory! Následující přehled uvádí běžné programátorské chyby:
    - Zapomněli jste do projektu vložit knihovnu RP6Library nebo je nesprávný název adresáře, v souboru makefile. Pokud začínáte tvořit nový projekt, nezapomeňte

změnit názvy adresářů v souboru makefile! Jinak se kompilátoru nepodaří najít tyto soubory!

- Zapomněli jste někde v programu středník?
- Chybí v programu něco důležitého nebo je nadbytečné?
- Dodržovali jste správnou syntaxi jazyka C? Kromě komentářů a obvyklých formátovacích pravidel pro mezery a tabulátory, psaní různých variant symbolů je pro kompilátor jazyka C vše ostatní relevantní. Pokud tAto příručKa obsahuje chybY, nemusí čtenář správně pochopit význam textu! Kompilátory nepovolují žádné chyby a mohou generovat obrovské množství chyb při vzniku malá chybičky. Bohužel kompilátor nemá automatickou opravu chyb, srovnatelnou s námi lidmi ...

14. Moje programy stále nefungují a robot neposlouchá moje povely - co je špatně?

- Nemám ponětí! ;-) Při požadavku na podporu musíte být konkrétnější v popisu svých problémů. Často dostáváme dotazy typu "Proč tento program nefunguje?". Co s tím. Samozřejmě potřebujeme trochu podrobnější popis toho, co program má dělat a co na něm nefunguje! V opačném případě se rychle ukáže, že se otázka změnila na kvíz ...
- Obvyklé chyby začátečníků jsou:
  - Přidání středníku na špatné místo – např. uzavření smyčky nebo výrazu if. Často umožníte vložení symbolu středník tam, kde nebude fungovat tak, jak jste si představovali!
  - Vytváření konstrukce if/else s vyhodnocením na nesprávných místech – to se může snadno stát, pokud je špatná struktura odsazování zdrojového textu.
  - Použití nesprávného datového typu pro proměnné – např. datový typ uint8_t může akceptovat hodnoty v rozsahu od 0 do 255, ale nelze jej použít pro počítání až do hodnoty 1500! Pro počítání tak vysoko budete muset použít datový typ uint16_t! Datový typ uint8_t také nebude akceptovat záporné hodnoty. Pro práci se zápornými hodnotami budete potřebovat znaménkový datový typ např. int8_t! Zkontrolujte přehled všech datových typů na začátku krátkého intenzivního kurzu jazyka C!
  - Zapomenutí na nekonečnou smyčku na konci programu – pokud tato nekonečná smyčka chybí, může váš program vytvářet podivné výsledky.
  - Používáte neblokovaný režim pro funkce "přesun" nebo "otáčení", ale zřídka voláte funkci "task_motionControl" nebo "task_RP6System"! Možná váš program obsahuje dlouhé přestávky generované funkcí mSleep. Při použití neblokovaného režimu funkcí "přesun" a "otáčení" nebo ACS musíte pro všechna zpoždění delší než přibližně 10 milisekund použít stopky! Funkce mSleep a další blokující funkce by neměla být používána v kombinaci s neblokujícími funkcemi! Přečtěte si, prosím, znovu kapitolu o knihovně RP6Library pro upřesnění podrobnosti a prostudujte ukázkové programy!
  - Před novou kompilací vždy pamatujte na uložení změněných zdrojových textů programu! V opačném případě bude kompilátor překládat předchozí nezměněné verze uložené na vašem pevném disku! V případě pochybností můžete provést příkaz MAKE CLEAN a znovu provést kompilaci projektu!

15. Potýkáte se s jinými problémy?

- Zkuste znovu přečíst důležité části příručky! To může často pomoci, ale někdy to nic nepřinese ...
- Máte již staženu nejnovější verzi software a poslední verze manuálu z http://www.arexx.com/ nebo z domovských webových stránek RP6 http://www.arexx.com/rp6?
- Používáte vyhledávací funkci na našem fóru --> http://www.arexx.com/forum/
- Navštívili jste webové stránky AVRFreaks věnované programovacímu jazyku C, systému AVR nebo problematice mikroprocesorů? --> http://www.avrfreaks.net/.
- Věděli jste, že můžete běžně navštěvovat obě fóra? (Nezačínejte, prosím tím, že okamžitě přidáte nový příspěvek – začněte hledáním svého problému pomocí různých vyhledávacích dotazů!). Můžete také zkusit použít obecný webový vyhledávač.

B – Kalibrace enkodérů

Účinné rozlišení snímače polohy závisí na reálném průměru kol a gumových pásů. Housenky gumových pásů se zatlačují do povrchu, nebo se deformují dovnitř. Výrobní tolerance navíc způsobují změny velikosti průměru. Pro kompenzaci těchto odchylek budeme muset provést několik měření a kalibrovat rozlišení snímačů (enkoderů).

Rozlišení je vzdálenost ujetá s každým krokem snímače. Teoretická hodnota pro rozlišení v tomto robotu je 0,25 mm/krok ekodéru, ale z praktických zkušeností jsme odvodili hodnoty rozlišení v rozmezí od 0,23 až do 0,24 mm.

Při kalibraci rozlišení enkodéru necháme robot poháněný na předem definovanou, dlouhou a rovnou vzdálenost (např. jeden metr nebo více), která se následně musí být přesně měřit s krejčovským nebo svinovacím metrem. Aby bylo možné zobrazit počet kroků kodéru, je robot během tohoto pohybu, připojen k počítači PC. USB kabel a plochý kabel muset být veden volně přes robota – netahejte ani nedržte kabel! Deska plošných spojů nárazníku na přední straně může být nastavena přesně na začátek měřicí pásky. Nastavte cestu robota přesně na přímce rovnoběžné s měřicí páskou.

Jako nácvik doporučujeme napsat program pro pohon robota přesně na vzdálenost jednoho metru. Alternativně můžete také zvolit 2 metry nebo jinou vzdálenost (nejdůležitější je, aby byl kabel dostatečně dlouhý). Samozřejmě můžete znovu sestavit program pro různé vzdálenosti a upravený program opakovaně nahrávat do robota. Výstupy z každé verze programu se uspořádají do seznamu ujetých vzdáleností v krocích kodéru. (Pokud jste příliš líní - existuje také menu v programu autonomního testu, které to může udělat za vás ;-))

Vzdálenost jeden metr odpovídá při rozlišení 0,25 mm přesně 4000 krokům enkodéru. Nyní, když test spustí pohyb robotu na vzdálenost pouhých 96,5 cm = 965 mm a čítač hlásí celkový počet 4020 kroků enkodéru, můžeme vypočítat jednoduchým dělením 965mm ku 4020 rozlišení přibližně 0,24 mm. Zapište, prosím tyto hodnoty do tabulky. Opakujte kalibrační postup několikrát a hodnoty zaznamenávejte v tabulce. Nyní vypočítáte průměrnou hodnotu a zadáte ji do pole parametrů ENCODER_RESOLUTION v souboru RP6Lib/RP6Base/RP6Config.h (relativní cesta z adresáře hlavního ukázkového programu – nezapomeňte soubor uložit!), znovu kompilujte program a nahrajte jej do robota. Opakujte tento test třikrát. Každý test by měl zlepšit výsledky v jízdě na přesnou vzdálenost jednoho metru. Pokud nedojde ke zlepšení, opakujte test znovu a zadejte novou hodnotu do konfiguračního souboru. Nikdy se však nepodaří provést přesnou kalibraci na 100% – k tomu budete potřebovat spoustu dalších senzorů.

Otáčení robotu na místě se dokonce zhorší kalibraci. Tyto rotace způsobí, že pásy prokluzují po podlaze, což způsobí výsledek, že se reálně ujede mnohem kratší vzdálenosti s ohledem na naměřené hodnoty. Výsledky budou velmi záviset na podmínkách povrchu. Kluzké parkety nebo koberec mít velmi odlišné kalibrační parametry pásů. Proto budete muset uvažovat kalibrační tolerance až 10° (při otáčení). Kromě toho některé povrchy mohou způsobit, že robot uklouzne stranou. Pokud chcete zahrnout tyto nežádoucí boční účinky do kalibrace, budete muset udělat nějaké další zkoušky.

Můžete se pokusit během otáčení zvednout robota za přední nárazník tak, že se podlahy dotýkají pouze zadní kola. Budete překvapeni, jak se robot nyní otáčí mnohem rychleji – to vám dává představu o tom, jak velký rozdíl je mezi reálnou a změřenou hodnotou.

Dokonalejší metody měření vzdálenosti a polohy

Měření vzdálenosti pomocí encodéru nikdy nedostane 100% přesnost. Pokud potřebujete lepší navigaci, budete muset použít další senzory.

Například na Michiganské Universitě v USA navrhli malý přívěs, který nese ekodérové snímače polohy pro jeden ze svých velkých pásových robotů. Robot táhne přívěs, který byl připojen pomocí otočné kovové tyče. Kola encodéru na nápravě přívěsu poskytují, po provedení několika výpočtů, přesné údaje o poloze. Podrobnosti lze studovat na:

http://www-personal.engin.umich.edu/%7Ejohannb/Papers/umbmark.pdf

text začíná na str. 20 (resp. na str. 24, v PDF verzi) nebo:

http://www.eecs.umich.edu/~johannb/paper53.pdf

Podobnou konstrukci bychom mohli navrhnout také pro RP6 (i když to nebude snadné) – případně můžete začít zvažovat nějaké pokusy s optickou počítačovou myší připevněnou na zadní nebo přední části robotu. Dále můžete použít elektronický kompas nebo podobné zařízení, které bude namontováno na nejvyšším místě robotu (aby se omezilo rušení od motorů a další elektroniky) a umožňuje přesné rotační pohyby.

Samozřejmě také Gyro může být dobrý alternativní snímač polohy ...

My jsme netestovali všechny tyto nápady, které musí být považovány za podnětné impulsy pro další zlepšování a studium ... samozřejmě některé uživatele nemusí zajímat přesné pohybování a navíc přidání senzoru myši také zadržuje vozidle v offroad podmínkách.

Přesné polohování principielně vyžaduje použití značek, které mají být umístěny v přesně definovaných místech. Tyto poziční body musí být samozřejmě dobře známé a snadno rozpoznatelné robotem, např. infračervené naváděcí majáky vysílají naváděcí signál pro robot. Robot dokáže detekovat směr majáků.

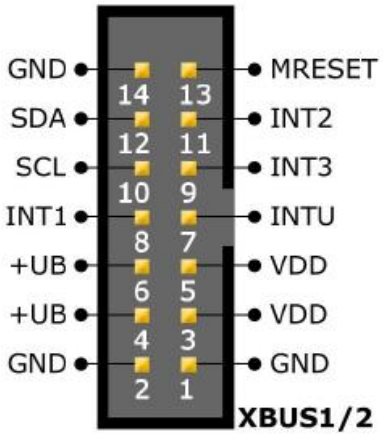
Nebo se rozhodnete implementovat snímače pro sledování čáry na podlaze.

Ve skutečnosti můžeme vymyslet mnohem složitější metody určování polohy ... ale omezíme se pouze na tento přehled několika nápadů, které vyžadují minimální množství výpočetní síly CPU. Mohli bychom instalovat kameru na strop místnosti a dálkově ovládat robota pomocí speciálního software pro rozpoznávání vzorů a bezdrátový přenos příkazových dat prostřednictvím vysokofrekvenčních nebo infračervených signálů. Některým uživatelům se již podařilo vytvořit podobný systém dálkového ovládání pro malého robota ASURO ...

Můžete si všimnout, že je snadné robota navigovat relativně přesně pomocí dálkového ovládání. Obvyklá přesnost ovládání je široce podporována vizuální zpětnou vazbou k uživateli, která přímo sleduje směr, překážky a cíle! Robot není schopen vidět tyto detaily bez kamer a výkonného zpracování obrazu. Kamera umístěná na stropě a barevná značka na horní straně robota nám společně s vnějším čidlem poskytují přesnější vizuální zpětnou vazbu ...

C – Rozmístění kontaktů na konektorech

Tento přehled vám poskytuje nejdůležitější rozložení kontaktů na konektorech základní desky. Seznam jsme dále rozšířili o řadu detailů pro používání.



Pro úplnost opět začneme definicí kontaktů konektoru pro rozšiřující moduly (viz kapitola 2):

Na hlavní desce je vývod 1 umístěn na straně bílého popisu s nápisy XBUS1 a XBUS2, respektive na značce "1" vedle konektoru.

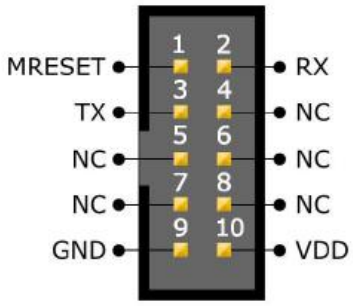
Kontakt +UB je napětí přímo z baterie, VDD je napájecí napětí +5 V, GND je "záporná" svorka (GND = zem), MRESET je master resetovací signál, INTx jsou linky externího přerušení, SCL je hodinová a SDA datová linka sběrnice I^2C.

<u>Důležitá informace:</u> prosíme, omezte maximální zatížení jednotlivých větví rozšiřujícího konektoru pro napájecí napětí VDD a +UB tak, aby jimi protékal <u>maximální proud 1A</u> (to platí pro oba kolíky SOUČASNĚ, takže: vývody 4 + 6 (+UB) a 3 + 5 (VDD))!

Veškeré další potřebné signálové linky mohou být připájeny na pájecí plošky konektoru USRBUS, které jsou propojeny 1:1 s ploškami na desce, to znamená, vývod konektoru PIN1 je připojen k Y1, PIN2 na Y2,, atd...

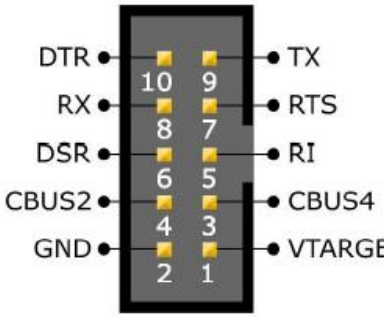
Rozmístění kontaktů na 10-vývodovém konektoru rozhraní USB je odlišné od všech ostatních konektorů:

Hlavní deska:



Pro umožnění komunikace je samozřejmě nutné prohodit signály RX a TX. Kromě toho, že se musela zrcadlit orientace konektorů tak, aby se zajistilo správné propojení, je správná orientace zajištěna tvarem konektorů plochého kabelu.

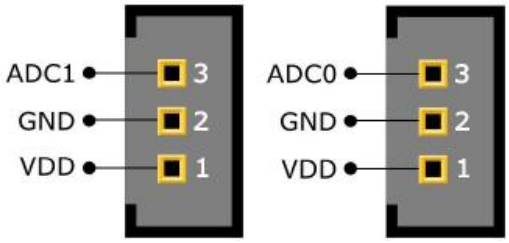
Rozhraní USB:



Pokud není komunikační konektor základní desky používán pro rozhraní USB, můžeme do něj připojit zařízení do jiných systémů – např. pro dálkové ovládání robotu přes UART z jiných mikroprocesorů.

Konektory AD převodníku:

Obrázek ukazuje uspořádání kontaktů konektoru obou volných kanálů AD převodníku.



Tyto konektory jsme neosadili a můžete je využívat pomocí konektorů se třemi kontakty s roztečí 2,54 mm. Při pájení buďte opatrní, abyste nezničili základní desku! Pokud nemáte zkušenosti s pájením, vynechejte prosím, pájení na desce spojů a raději začněte pokusy pomocí rozšiřovací desky!

Na tyto volné ADC kanály můžete volně připojit dva analogové nebo digitální snímače. Výstupní napětí snímače je povolené v rozsahu od 0 do 5V a konektory poskytují napájecí napětí 5V pro senzory. Je rozumné připájet na základní desku velké elektrolytické kondenzátory – s hodnotou od 220 do 470µF (nepřekračujte tuto hodnotu!) Pro většinu aplikací bude dostatečné provozní napětí kondenzátoru větší nebo rovno 16V!

Při práci se snímači, které vyžadují obrovské proudy, budete pravděpodobně opravdu potřebovat velké elektrolytické kondenzátory – např. populární Sharp IR snímače vzdálenosti. Blokovací kondenzátory (100nF) na základní desce jsou vhodné pouze pro krátké spoje – pro dlouhé přívody musí být připájeny přímo ke snímači (doporučujeme připojit tyto kondenzátory přímo na čidlo, kde je dostatek místa i pro krátké vývody!).

Ostatní spoje jsou jasně označeny na základní desce. Je vhodné se podívat na příslušná schémata na CD!

D – Recyklace a bezpečnostní pokyny

Recyklace

Likvidace RP6 v domovním odpadu není povoleno! Při likvidaci musí být robot doručen do místního recyklačního centra nebo jiných recyklačních center pro elektroniku!

Podrobnosti zjistíte u místního prodejce elektroniky.

Bezpečnostní pokyny pro baterie

Baterie (akumulátory a alkalické články) musí být uchováván mimo dosah dětí! Nedovolte, aby baterie ležely na místech, která jsou přístupná pro každého, jejich části mohou spolknout děti nebo zvířata. V případě, že došlo kpolknutí některé části robotu, je nutné okamžitě vyhledat lékaře!

Kontakt s vytékajícími nebo jinak poškozenými bateriemi mohou způsobit poleptání kůže. V případě, že musíte manipulovat s takovými předměty, je nutné používat vhodné ochranné rukavice.

Nezkratujte baterie a nevyhazujte baterie do ohně. Není dovoleno nabíjet alkalické články! Dobíjení běžných článků může způsobit jejich explozi! Omezit se na používání pouze speciálních nabíjecích akumulátorových baterií (např. akumulátory NiMH) a vhodné nabíjecí zařízení kompatibilní s těmito články!

Recyklační pokyny pro baterie

Analogicky jako u RP6 není povolena likvidace baterií (akumulátorů a alkalických článků) do domácího odpadu! Koncoví uživatelé musí recyklovat všechny vadné a použité baterie. Proto, prosím, vraťte své vadné, vybité a použité baterie ke svému prodejci nebo do místního recyklačního centra pro baterie! Akumulátorové baterie a alkalické články můžete vrátit v každém obchodě, který prodává také toto zboží.

Tímto způsobem jsou splněny zákonné povinnosti a zároveň se přispívá k ochraně životního prostředí!