Začínáme s

Následující text byl poprvé zveřejněn v dnes již neexistujícím robotickém časopise Robot Revue (www.robotrevue.cz) v roce 2010 jako seriál článků. Seriál, který si vzal za cíl představit na příkladech mikrokontroléry PICAXE a jejich programování. Tento text v jiné grafické podobě má stejný obsah, tedy je neaktualizovaný s ohledem na změny v nabídce mikrokontroléru i novějších příkazů. Uvedené příklady volené s ohledem na zaměření časopisu a předpokládanou úroveň znalostí čtenářů nemusí být optimálním řešením daných úloh. O to vůbec nejde, cílem bylo názorně představit novým zájemcům možnosti mikrokontrolérů PICAXE na příkladu typu 20M a provést zájemce úvodem při psaní vlastních programů a aplikací těchto mikrokontrolérů.

Ing. Michal Černý

Začínáme

Každý správný robot by měl něco dělat, nějak reagovat na své okolí. K tomu potřebuje několik věcí. První z nich je zdroj energie, protože bez energie to zkrátka nejde. Budeme předpokládat, že půjde o energii elektrickou z baterií nebo akumulátorů, stavět parní roboty by bylo jistě zajímavé, ale v dnešní době poněkud nepraktické, a atomový reaktor se do domácího robota asi těžko vejde. Další jsou čidla, jenž poskytnou robotovi informace z okolí, jeho oči, uši a hmat. Kromě toho bude potřebovat něco, co mu poskytne možnost se hýbat, případně svítit nebo vydávat zvuky, tedy vykonat to, co je jeho reakcí na podnět. Zbývá to nejdůležitější, mozek, bez něhož je všechno předchozí jen bezduchou hromádkou „železa“. Ta část, která dokáže přijmout informace od čidel, rozhodnou, co se má udělat, a vydat příkaz pohonům

a světlům, aby to vykonaly. Mozkem robota bývá počítač, i když většinou vypadá úplně jinak, než ten, který stojí doma na stole, spojuje nás přes internet se světem a dají se na něm hrát hry. Chování robota určuje program v jeho řídícím počítači, nezbytnou součástí stavby je tedy i programování, dokonce často částí nejzajímavější.

Nejjednodušší řídící počítače, které k ovládání malého robota úplně stačí, jsou velké jako nehet, často ještě mnohem menší, a tvoří je jediný integrovaný obvod, „brouk“ s několika „nožičkami“ a pár dalších součástek. Postavit „mozek“ pro robota s takovým obvodem je poměrně jednoduché nebo se dá použít některý z prodávaných elektronických modulů, jenže co s tím programováním? Naučit se programovat jednočipové procesory, jak se těm nejmenším počítačům říká, není jednoduché, a udělat program opravdu dobře, to už vůbec ne.

Učit se systematicky programování asi není naším cílem, ale v případě stavby malého robota prostředkem, který potřebujeme, a který je třeba co nejvíc zjednodušit a usnadnit. Právě tento přístup měli na mysli tvůrci procesorů PICAXE z Velké Británie. Vyšli ze známých a často používaných procesorů PIC vyráběných firmou Microchip, do nichž vložili speciální zaváděcí program a předprogramovali řadu užitečných funkcí, jenž se hodí právě v malé robotice. Zaváděcí program dovoluje přeprogramování uživatelského, tedy toho našeho programu z PC bez nutnosti použít nákladný programátor, místo něj nám stačí pouhý kabel. Obvody PICAXE jsou celkem levné a lze je přeprogramovat zhruba 100000x, takže i kdybychom každý den vyzkoušeli 100 verzí svého programu, vydrží nám několik let. Můžeme zkrátka zkoušet podle chuti. Zavaděč nesmíme z procesoru smazat, v tom okamžiku by se z něj stal „obyčejný“ PIC a byl by pro naše účely ztracen. Program do PC, který budeme používat k psaní a zkoušení svých programů, takové smazání zavaděče ani neumožňuje.

Program se dá zapsat příkazy, tedy slovy, nebo sestavit graficky z bloků na obrazovce. To je mnohem názornější, ale u velkých programů nepřehledné. Tím, že máme připravené právě ty funkce, které jsou potřeba, vychází programy pro PICAXE mnohem kratší a jednodušší, než kdybychom museli programovat obyčejný procesor PIC nebo jiný. Ukážeme si to na příkladu. K pohybu malých robotů se většinou používají modelářská serva. Jsou to malé krabičky, uvnitř kterých je elektromotor, převody a také elektronika. Nemusíme nutně vědět, jak to přesně funguje, prostě vyšleme servu povel a ono vykoná odpovídající pohyb, natočí podle toho svoji páku nebo kotouč. Zatímco v assembleru, jazyce, který asi nejlépe odpovídá přirozenému programování jednočipových procesorů, by povely sloužící k mávnutí pákou do krajní polohy a zase zpátky zabraly nejméně 20 řádků, v jazyce procesorů PICAXE nám stačí tři příkazy, tento zlomek programu:

```
servo 1,200
pause 1000
servo 1,150
```

Není třeba složitého vysvětlování, slovům se dá celkem dobře porozumět. Příkaz servo se asi bude týkat našeho serva, jednička za ním určuje, které servo to bude, druhé číslo zadává polohu, jak se má natočit. 150 odpovídá prostřední poloze, číslo 200 jedné krajní výchylce (případně číslo 100 druhé). Nebudeme zatím zkoumat, proč to tak je. Mezi povely s významem „dojeď na kraj“ a „vrať se doprostřed“ je ještě jeden, slovo „pause“ znamená čekání, číslo za ním jak dlouho, v našem případě 1 vteřinu.

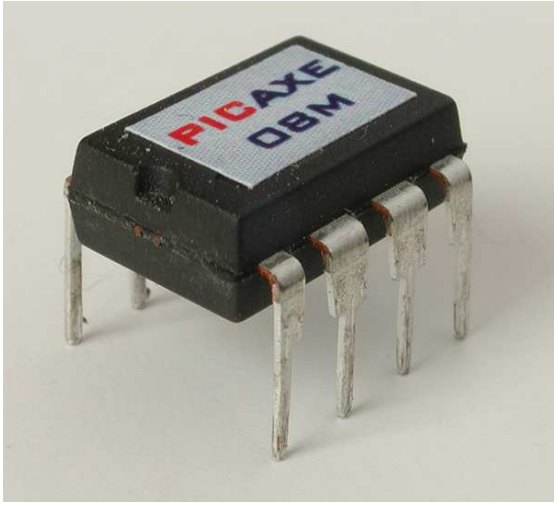
Ukážeme si ještě jeden příklad, už kompletní funkční program, který bude přerušovaně pípat a k tomu před každým pípnutím rozsvítí nebo naopak zhasne svítivou diodu. Na obrázku se můžete podívat, jak podobný program vypadá, když je vytvořen graficky.

```
start:
sound 2,(100,30)
pause 400
toggle 0
goto start
```

Program v PC, v němž budeme své vlastní programy sestavovat a zkoušet, se dá volně stáhnout z internetu. Dovoluje funkci námi zapsaného programu dokonce simulovat, takže s pokusy můžeme začít i bez zakoupeného procesoru a stavby jednoduchých zapojení, jen na monitoru PC. Simulace je zajímavá a dá se na ní hodně poznat, cílem je ale skutečné řízení skutečného robota. V následujících dílech našeho seriálu se krok za krokem seznámíme s jednotlivými stavební kameny, z nichž se pak dá složit program pro robota, a budeme si je průběžně zkoušet v praxi. Pro ty, kteří nechtějí nebo nemohou shánět potřebné díly a součástky sami, připravíme zkompletované sady na pokusy. Příště tedy můžeme začít.

Rodina procesorů PICAXE

Minule jsme si řekli, co vlastně procesory PICAXE jsou, k čemu jsou určeny a proč je výhodné je používat, nyní se podíváme na vybrané typy procesorů z této skupiny. Rozdíly najdeme především v počtu vstupů a výstupů, jenž můžeme použít pro své konstrukce, ve vybavení (například vnitřními převodníky), rozsahu paměti na program a rychlosti.

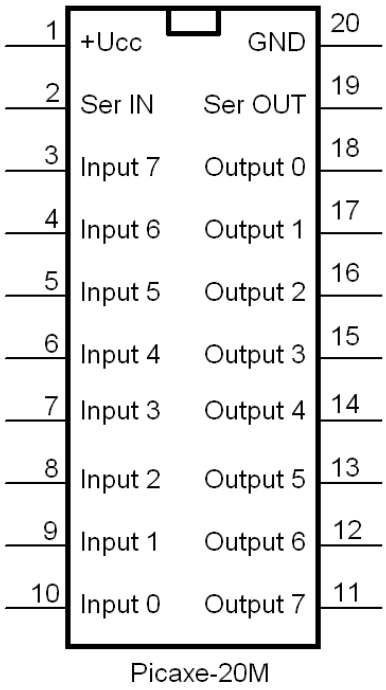


Jako první si představíme nejmenší procesor PICAXE-08M, ten má jen 8 vývodů. Vždy dva vývody slouží k napájení, jeden pro programování (přenos dat do procesoru) a jeden jako sériový výstup, ten může být případně využit i k jiným účelům. Tři z osmi vývodů jsou tedy obsazeny úplně, jeden částečně. Zbývající čtyři (případně pět) vývodů můžeme využít jako vstupy nebo výstupy. U tohoto typu procesoru se to dá zvolit a dokonce měnit i za běhu programu. Do paměti PICAXE-08M se vejde přibližně 80 řádků programu, základní frekvenci (rychlost) 4 MHz lze zvýšit na 8 MHz, ovšem za cenu odlišné funkce některých příkazů.

PICAXE-14M je větší, má celkem 14 vývodů, podobně jako v předchozím případě tři z nich mají přesně danou funkci a jeden částečně. Tentokrát ovšem nelze volit, který z vývodů bude vstup a který výstup, procesor má pevně daných 5 vstupů a 6 výstupů. Rozsah programu i frekvence jsou stejné.

PICAXE-20M má, jak už označení napovídá, 20 vývodů, čtyři mají předurčenou funkci a zbývajících 16 se dělí na 8 vstupů (Input) a 8 výstupů (Output). Tento procesor založený na PIC 16F677 od firmy Microchip budeme v našem seriálu a cvičných konstrukcích používat. Paměť má stejnou velikost jako u předchozích typů, tedy přibližně na 80 řádků programu, frekvence je také stejná, na čtyřech vstupech máme k dispozici převodníky napětí.

Představitelem výkonnějších procesorů PICAXE je typ 28X1, ten podobně jako 08M dovoluje některé vývody použít volitelně buď jako vstup nebo jako výstup a nabízí celkem 0 - 12 vstupů a 8 - 16 výstupů. Jeho paměť je podstatně rozsáhlejší, pojme až 1000 řádků programu, umožňuje využít 4 vstupy jako převodníky napětí a lze jej provozovat až na frekvenci 20 MHz. Nejvýkonnější procesor PICAXE-40X2 pak dovoluje uložit až 4000 řádek programu ve čtyřech oddílech a běží až na 40 MHz.



Picaxe-20M

Je vidět, že rozpětí parametrů procesorů PICAXE je poměrně široké a dostaneme-li se do situace, kdy už daný procesor úloze nestačí počtem vstupů a výstupů, pamětí nebo rychlostí, lze přejít na vyšší typ, přitom převážná část programu bude využitelná s minimálními změnami. Co se tedy naučíme na poměrně malém a levném PICAXE-20M, můžeme pak využít i při řešení mnohem složitějších úloh.

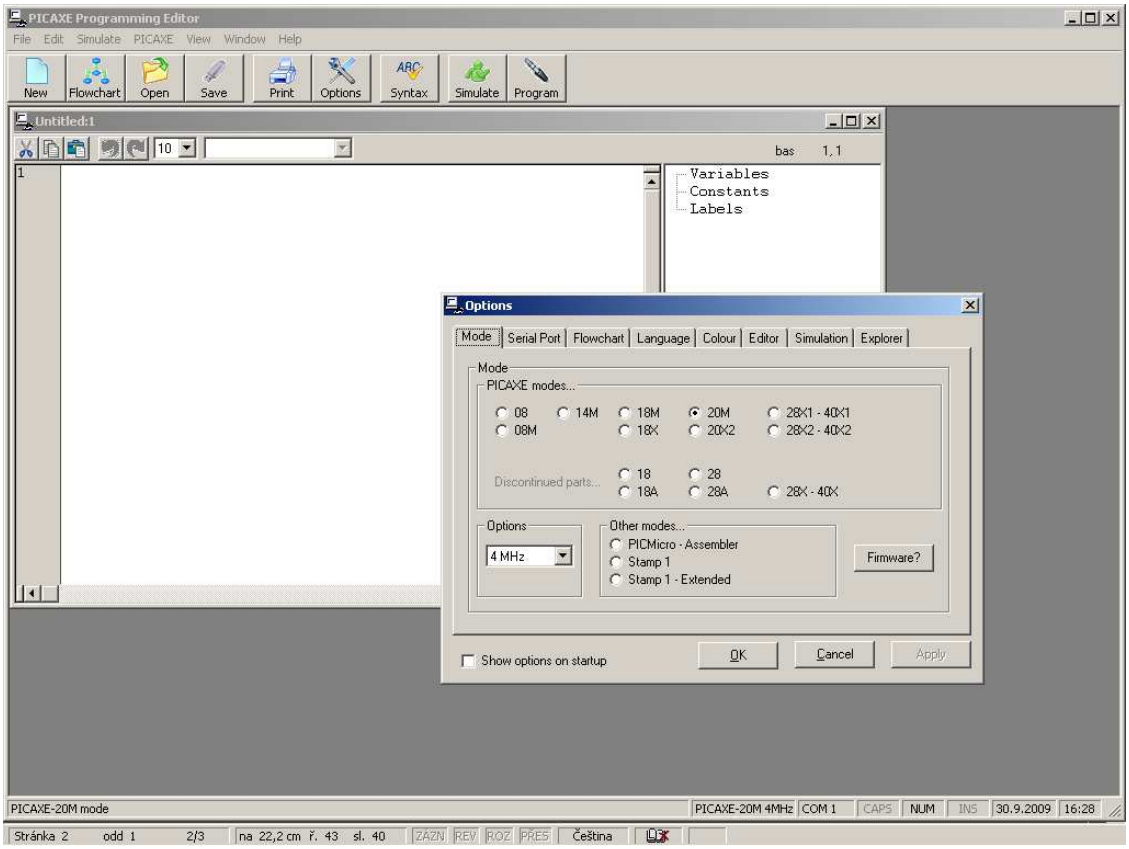
Je výhodné si zapamatovat rozložení „nožiček“ procesoru a jejich význam, bohužel, používají se dva různé způsoby značení. Obecně se vývody integrovaného obvodu počítají tak, že si jej položíme značkou (výřezem, prohlubní) na jeho konci (kratší straně) nahoru a postupujeme proti směru hodinových ručiček kolem dokola. U vývodu číslo 1 bývá navíc na pouzdře vylisovaná tečka, ale nemusí tam být. Náš PICAXE-20M tedy má v dané poloze vývod 1 vlevo nahoře, vývod 10 vlevo dole, vývod 11 vpravo dole a poslední vývod 20 vpravo nahoře.

Druhý způsob označení se už týká konkrétního procesoru a budeme jej využívat mnohem častěji, vychází totiž z významu (funkce) vývodů. Budeme-li například hledat vstup 0 (Input 0), najdeme jej na vývodu (nožičce) 10, budeme-li hledat výstup 0 (Output 0), je na vývodu 18. Do doby, než se nám poloha a funkce vývodů vryje do paměti, se vyplatí okopírovat si nebo obkreslit nákres s rozložením na papír a mít jej stále při ruce.

Instalace obslužného programu

Než se pustíme do prvních pokusů s programováním, musíme si stáhnout do osobního počítače obslužný program, který spojuje editor, v němž budeme své programy psát, a programátor, který náš program přeloží a přenese do procesoru. Obslužný program najdeme na internetových stránkách projektu PICAXE *www.rev-ed.co.uk/PICAXE* nebo na stránkách našeho časopisu *www.robotrevue.cz* , instalační soubor má 38 MB. V celém seriálu budeme používat výhradně verzi programu 5.2.7, to proto, aby se prostředí, v němž budeme pracovat, shodovalo s popisem. Kromě toho si stáhneme PDF soubor se stručnou českou příručkou

programátora, v níž jsou vysvětleny nejpoužívanější příkazy. Podrobná dokumentace v angličtině se nainstaluje spolu s programem, my však budeme vycházet především z české verze, i když je stručnější a nepostihuje všechny možnosti. Budeme-li se odvolávat na konkrétní místo české příručky, bude v textu v závorce číslo strany uvedené hvězdičkou**,** například (*7) bude znamenat sedmou stranu české příručky programátora.



Program nainstalujeme do PC běžným způsobem, vyžaduje systém Windows (verze 95 nebo novější) a asi 66 MB místa na pevném disku. Lze bez problémů využít i starší počítač. Je výhodou, pokud má sériový port (COM), zjednoduší se tím programování. Nainstalovaný program zanechá na pracovní ploše zástupce (ikonu), spustíme jej a jako první přes Menu - Help - About zkontrolujeme, že máme správnou verzi. Potom si vyvoláme nabídku parametrů Menu - View - Options a zvolíme mód procesoru 20M, dostupný sériový port (třeba COM1) a jazyk. Vzhledem k tomu, že čeština zatím v nabídce není, budeme zřejmě vycházet z angličtiny. Pak nastavení parametrů opustíme.

Základní příkazy

Pokusíme se napsat jednoduché programy, první bude jen běhat v kruhu bez dalších projevů, druhým rozblikáme LED na výstupu 7, i když zatím jen v simulovaně. K tomu budeme potřebovat několik základních příkazů. V jazyce Basic se jednotlivé řádky programu číslují, my se ale o toto číslování nemusíme starat, to zajistí editor sám.

REM

bývá nejčastěji používaným příkazem. Nedělá nic, jen sděluje editoru, že to, co bude následovat až do konce řádku, má být považováno za komentář, tedy vysvětlující poznámky programátora.

Stejný příkaz můžeme také zapsat stručněji středníkem (;) nebo apostrofem (‘). Je rozumné si vždy na začátek poznamenat, k čemu je program určen, případně jaká je to verze programu číslem nebo datem. Používáme-li více druhů procesorů, doplníme vždy i typ procesoru, na němž byl program odladěn. V průběhu zápisu programu komentáře využíváme k vysvětlení, co ta která část programu dělá. Není nutné věnovat komentáři celý řádek, můžeme jej přidat i za výkonný příkaz (*1). Nezměníme-li si barvy individuálním nastavením, vypisuje editor komentáře pro názornost zeleně.

NÁVĚŠTÍ

je název, jímž označíme místo v programu, na něž se chceme později odvolat a skákat na něj. Návěští musí začínat písmenem a končit dvojtečkou (*1).

Programy neběží jen „shora dolů“ tak jak je píšeme, někdy je nutné skočit na určité místo programu. Abychom nemuseli k označení využívat čísla řádků, která nám editor stejně průběžně mění při každé úpravě, označíme si dané místo v programu srozumitelným názvem, to je právě návěští. Návěští se v programu vypisuje černě jak tam, kde je definováno (s dvojtečkou na konci na místě kam je třeba skákat), tak jako součást příkazů skoku (už bez dvojtečky).

PAUSE

Často je potřeba určitou přesnou dobu počkat, k tomu slouží příkaz Pause, za nímž je udáno, kolik tisícin sekundy se má čekat. PAUSE 1500 bude tedy čekat 1,5 s. Podrobněji viz (*17).

Doba čekání v tisícinách sekundy odpovídá jen pro hodinový kmitočet procesoru 4 MHz, pokud bychom příkazem zvýšili frekvenci na 8 MHz, čekání se odpovídajícím způsobem zkrátí (Pause 1 odpovídá 0,5 ms). Jako parametr můžeme zadat čísla 0 - 65535, takže nejdelší doba, kterou takto můžeme zadat, je přibližně minuta, přesně 65,5 s.

HIGH, LOW

Tyto dva příkazy slouží k ovládání výstupů, jejichž číslo následuje. Například HIGH 1 nastaví výstup 1 (Output 1) na hodnotu H (napětí blízké kladnému napájení), LOW 1 nastaví výstup 1 na hodnotu L, tedy napětí blízké zemi. Podrobněji viz (*10) a (*15).

Po startu programu (zapnutí procesoru) jsou všechny výstupy ve stavu L, tedy bez napětí. Pokud pošleme příkaz HIGH (LOW) na výstup, který už v příslušném stavu je, nic se neděje, příkaz se vykoná (zabere určitý čas), ale stav výstupu se nezmění ani nijak nezakmitne. Většina programování jednoduchých zařízení je právě „o tom“, kdy který výstup zapnout a vypnout, HIGH a LOW patří mezi nejpoužívanější příkazy.

GOTO

Parametrem příkazu GOTO je námi zadané návěští, kam program bez dalších podmínek ihned skočí a pokračuje ve vykonávání příkazů (*9).

Skok pomocí GOTO svádí při častém používání k vytváření nepřehledné struktury programu, v níž se pak jen velmi obtížně hledají chyby. Jeho použití proto omezujeme na jednoduché a zřejmé případy a dáváme pokud možno přednost jiným způsobům, zatím jej ale budeme využívat jako nejjednodušší možnost.

První program

Zapíšeme do editoru náš první program, řádky se očíslují automaticky. Na prvním je zeleně vypsaný komentář s názvem nebo spíš vysvětlením, o jaký program jde, na druhém řádku je návěští, sem tedy budeme skákat. Na třetím řádku je příkaz k čekání, aby bylo možné dobře sledovat, co program dělá, PAUSE 1000 odpovídá čekání 1000 / 1000 = 1 s. Poslední, čtvrtý řádek, obsahuje příkaz skoku na návěští START. Celý program nebude dělat nic jiného, než stále dokola čekat jednu sekundu. To je dost málo, ale ukážeme si na něm, jak takový program zkontrolovat, simulovaně spustit a sledovat jeho chod.

```
1       REM Tohle je první pokusný program
2       start:
3         pause 1000
4         goto start
```

Nad zápisem programu je několik tlačítek, najdeme to s nápisem Syntax a klikneme na jej. Slovo „syntaxe“ znamená česky skladba, týká se formální správnosti zápisu programu, toho, zda je program zaznamenaný podle pravidel, počítač (procesor) mu bude schopen porozumět a něco podle něj vykonat. Kontrola syntaxe pouze prověří, zda jsou jednotlivé příkazy zapsané správně, to ale v žádném případě ještě neznamená, že program je správný jako celek a bude dělat to, co jsme od něj chtěli. Pokud jsme program zapsali přesně jak je uvedeno, kontrola proběhne, ukáže nám znaménko odškrtnutí v zeleném poli a navíc sdělí, že náš program zabere z paměti procesoru 8 byte z celkem 256 dostupných. Tomuto sdělení musíme věnovat pozornost při psaní delších programů, říká nám, kolik je v procesoru ještě místa a kdy narazíme na meze jeho možností.

Vpravo vedle plochy, do níž zapisujeme program, se nám zobrazuje seznam použitých proměnných (Variables), konstant (Constants) a návěští (Labels), takže máme i ve složitějším programu přehled, která návěští jsme použili a případně se na ně můžeme ťuknutím do přehledu rychle přemístit v zápisu programu. Konstanty a proměnné poznáme později.

Po úspěšné kontrole programu si rovnou vyzkoušíme reakci na chybu. V posledním řádku změníme slovo START na STARTUJ a znovu spustíme kontrolu syntaxe. Nyní kontrola skončí jinak, s obrázkem brouka (štěnice nebo anglicky „bug“ se často používá jako označení chyby v programu) a vypíše se příčina, je-li ji program schopen rozpoznat. V našem případě „Label not defined - startuj“ znamená, že návěští STARTUJ není definované (neoznačuje žádné místo). Chybu opravíme a pokusíme se program rozběhnout.

Tlačítko „Simulate“ otevře tři nová okna, v jednom se ukazuje to, co posílá procesor po sériové lince ven z počítače, to se nás zatím netýká, v druhém je zobrazen obsah paměti, to také zatím není podstatné. V největším z nově otevřených okem je v levé polovině schématický nákres procesoru PICAXE-20M a u každého z vývodů je znázorněno, v jakém je stavu. U výstupů tmavě zelená barva značí stav L, světle zelená barva stav H, vstupy jsou zobrazeny jako tlačítka s kontrolkou, tmavá kontrolka znamená, že vstup je ve stavu L, výrazně žlutá ukazuje stav H. Stav vstupů lze kliknutím měnit. Protože ale zatím se vstupy a výstupy nepracujeme, soustředíme pozornost jen na zápis programu. Během simulace se v něm po většinu doby, přesněji řečeno vždy 1 sekundu, ukazuje zvýrazněný příkaz PAUSE, pak krátce problikne příkaz GOTO, bliknutí návěští START ani není vidět. Program běhá v kruhu a nic kromě čekání nedělá, tak jak jsme předpokládali. Novým stiskem tlačítka Simulace, tentokrát s červeným obrázkem stop, simulaci ukončíme a okna ukazující stav v procesoru se sama zavřou.

```
1       REM Program pro PICAXE-20M
2       REM Blikajici LED
3       start:
4         high 7
5         pause 1000
6         low 7
7         pause 1000
8         goto start
```

Do programu přidáme příkazy, které budou měnit stav výstupu 7. Ve smyčce ohraničené na začátku návěštím START a příkazem GOTO jsou nyní čtyři příkazy, které procesoru říkají: Nastav výstup 7 na H (logickou jedničku, napětí blízké napájecímu), počkej 1 s, nastav výstup 7 na L (logickou nulu, napětí blízké nule), počkej 1 s. Zkontrolujeme syntaxi a spustíme simulaci, nyní se již ve schématu procesoru rozbliká výstup 7 (vývod 11 obvodu). Tím máme hotový první jednoduchý funkční program, který viditelně něco dělá.

Pokusíme se pozměnit příkazy HIGH a LOW tak, aby místo výstupu 7 blikal jiný výstup, třeba 0, pak dvakrát zrychlíme blikání (snížíme parametr PAUSE) nebo zkusíme udělat krátké záblesky s dlouhou mezerou (první příkaz PAUSE kratší, třeba 100, druhý delší, třeba 1900) nebo naopak dlouhý svit k krátkými mezerami. I takto jednoduchý program je možné zkrátit, a to docela podstatně. Seznámíme se s dalším příkazem.

TOGGLE

nastavuje zadaný výstup podobně jako HIGH nebo LOW, ale s tím rozdílem, že vždy změní stav výstupu. Pokud tedy byl výstup ve stavu logické nuly (L), nastaví logickou jedničku (H), jestliže byl v H, nastaví L.

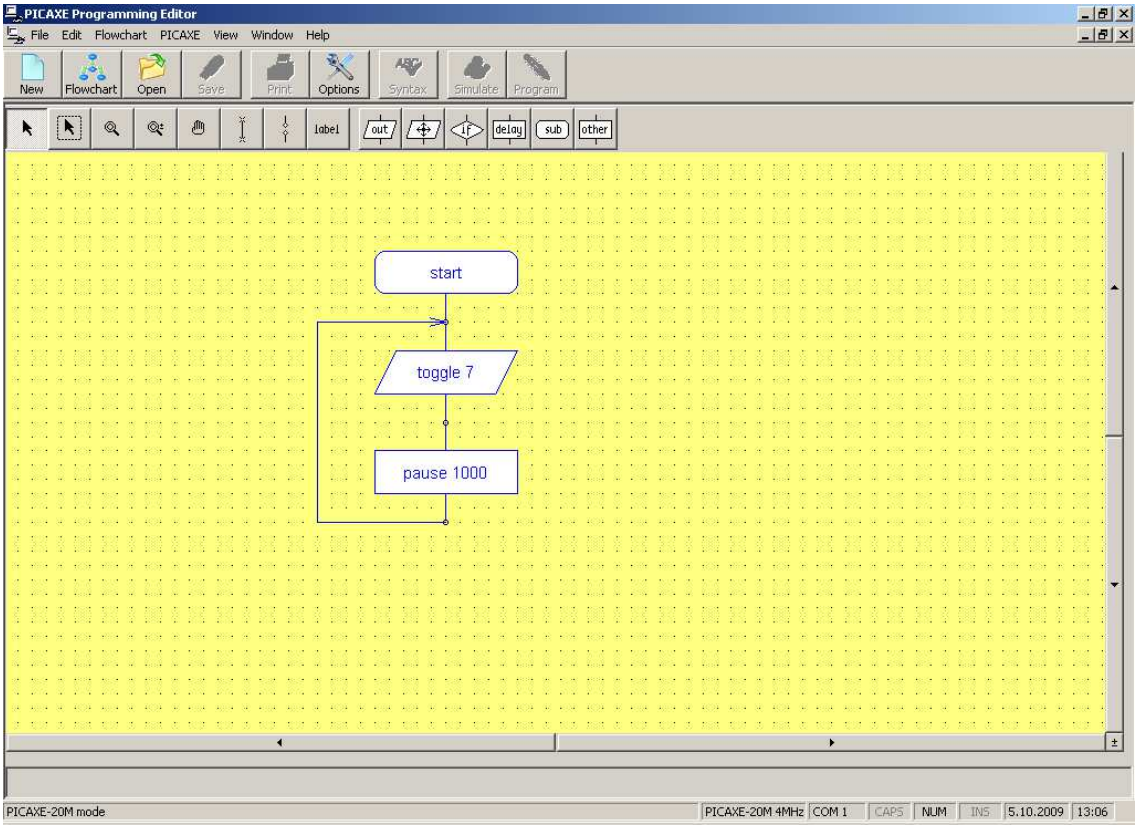
Náš původní program pracující s výstupem 7 tímto příkazem zkrátíme o dva funkční řádky, ovšem za tu cenu, že již nebudeme moci měnit samostatně dobu zapnutí a vypnutí, blikání bude mít vždy stejnou střídu a do příkazu PAUSE zadáme polovinu doby odpovídající periodě. Na druhou stranu budeme-li chtít změnit frekvenci blikání, stačí upravit jen jeden parametr, podobně k přesměrování na jiný výstup stačí změnit jen jeden příkaz. Doba potřebná na vykonání ostatních příkazů je zanedbatelná, program tráví naprostou většinu času čekáním.

```
1       REM Program pro PICAXE-20M
2       REM Blikajici LED 2
3       start:
4         toggle 7
5         pause 1000
6         goto start
```

Grafický zápis programu

Pro zápis programu nemusíme používat jen slovní příkazy, ale také skládání z grafických bloků. Stiskem tlačítka Flowchart se dostaneme do jiné části editoru, v níž máme připravený vstupní bod programu „Start“. Nad žlutou plochou se pod jednotlivými tlačítky skrývají

skupiny příkazů, z nichž si vybíráme a sestavujeme program jako domino. Začneme příkazem TOGGLE který najdeme ve skupině Out, ťukneme do něj a usadíme grafický symbol tak, aby se značka vstupního bodu (malého kroužku) kryla s výstupním bodem Start. Hodnotu parametru zvolíme z výběru nabízených hodnot v levém dolním rohu okna editoru. Pak podobně ze skupiny Delay vezmeme a usadíme příkaz pause, parametr 1000 je předvolený, ten ani měnit nemusíme. Zbývá udělat smyčku, aby program běhal stále dokola. V tomto případě nepoužijeme příkaz, ale čáru (Draw lines), kterou natáhneme od výstupu PAUSE vlevo, pak nahoru a vpravo do bodu mezi START a TOGGLE. Tím je program hotový.



Graficky zadaný program můžeme přenést do skutečného procesoru, ten ale zatím nemáme, kontrolovat syntaxe a simulovat přímo nejde. V Menu - PICAXE můžeme zkonvertovat grafický záznam na příkazy Basicu a pak už lze simulovat, dokonce v grafu se za běhu programu zvýrazňuje právě prováděný příkaz. Vyzkoušíme si to, dostaneme následující zápis. Editor sám označil odkud program pochází a kdy byl konvertován, přidal počáteční návěští „main“, zvolil si sám návěští smyčky „label_1B“ a jinak zachoval příkazy s parametry, které jsme zadali.

```
'BASIC converted from flowchart:
'

'Converted on 10.5.2009 at 13:09:24


main:
label_1B:     toggle 7
```

```
pause 1000

goto label_1B
```

Grafický záznam programů je jistě přehlednější než zápis příkazy, sám dokonce připomíná a nabízí použitelné příkazy, vyžaduje ale podstatně více prostoru a rozsáhlejší struktury už přehlednost ztrácí. Bohužel, nelze volně přecházet z jedné formy záznamu do druhé, proto si už jen jednou ukážeme tento alternativní způsob, ale jinak budeme pracovat se slovními příkazy.

Světelný had

Pokusíme se s tím, co jsme dosud poznali, naprogramovat efekt „světelného hada“, tedy běžící světlo. Vyžaduje to, abychom postupně na krátkou dobu rozsvítili (uvedli do stavu H) výstupy 0 až 7 procesoru. Program zapíšeme, zkontrolujeme syntaxi (měl by mít 44 byte, tedy zabírat zhruba šestinu paměti procesoru), případně opravíme chyby a spustíme simulaci. V grafickém zápisu stejně fungující program je na snímku.

```
1   REM Program pro PICAXE-20M
2   REM Světelný had - 1
3   start:
4     high 0 pause 200 low 0
5     high 1 pause 200 low 1
6     high 2 pause 200 low 2
7     high 3 pause 200 low 3
8     high 4 pause 200 low 4
9     high 5 pause 200 low 5
10    high 6 pause 200 low 6
11    high 7 pause 200 low 7
12    goto start
```

Konstanty, symboly, operátory

Chceme-li v předchozím programu běh „hada“ zrychlit nebo zpomalit, musíme zasáhnout do všech osmi příkazů PAUSE a jejich parametr zmenšit nebo zvětšit. Ukážeme si další možnosti, jimiž tento program zjednodušíme a zkrátíme.

KONSTANTY

jsme již použili v prvním programu, číslo 1000 je v programu pevně zadané, tedy konstanta. Zatím stačí uvědomit si konstantu jako pojem (*1).

Běžná čísla v desítkové soustavě, píšeme přímo. Pokud bychom potřebovali zapsat číslo v šestnáctkové (hexadecimální) soustavě, uvedeme jej znakem $, například $C = 12, $1A = (1*16)+(10*1) = 26. Při práci se vstupy a výstupy se častěji používá zápis ve dvojkové (binární) soustavě, který je uveden znakem %, například %00001001 = 9. V čísle pozice vpravo patří nejméně významnému bitu (má váhu 1), pozice vlevo nejvíce významnému bitu (váha 128). Binární zápis má tu výhodu, že v něm přesně pozice jedniček odpovídá

(vstupům) výstupům procesoru ve stavu H, ostatně i při simulaci posledního příkladu bylo vidět, že v pravé části okna vedle schématu procesoru na řádku pins(out) běhala jednička v binárním vyjádření výstupů zprava doleva. Konstantou nemusí být vždy jen číslo, ale také znak nebo celá skupina znaků uzavřená do uvozovek, takže můžeme pracovat i s krátkým textem.

SYMBOLY

respektive symbolické názvy jsme již také poznali, zvláštním případem jsou návěští, což není nic jiného, než symbolické pojmenování místa v programu. Pojmenovat můžeme ale také třeba výše uvedené konstanty a pak je v programu použít opakovaně pod srozumitelným názvem, obdobně můžeme symbolickým názvem označit proměnnou (viz dále) (*2).

Při definování začneme zápis slovem SYMBOL, pak následuje námi zvolený název, rovnítko a hodnota, kterou chceme názvu přiřadit. Zápis SYMBOL cekani = 200 tedy znamená, že slovo „cekani“ bude v programu totéž co zápis 200. Výhoda je jednak v tom, že použitým názvem vyjádříme účel konstanty, jednak to, že symbolicky pojmenovanou konstantu můžeme v programu použít mnohokrát, ale chceme-li změnit její hodnotu, stačí upravit jen jediné místo, to, na němž je konstanta definována.

OPERÁTORY

známe z matematiky, nejčastěji používáme základní čtyři operátory pro sčítání, odčítání, násobení a dělení (+ - * /). Kompletní seznam a konkrétní způsob zápisu najdeme v (*3).

Z dalších operátorů se hodí nejvíce MAX a MIN, které omezují výsledek operace na zadanou největší (nejmenší) hodnotu. Z logických operátorů používaných v podmínkách, jimiž se pak běh programu řídí, nachází nejčastější využití AND a OR. První z nich znamená, že obě části podmínky, mezi nimiž stojí, musí být současně splněny, aby se příkaz provedl, druhý znamená, že stačí, aby byla splněna třeba jen jedna z částí podmínky (nebo obě, to nevadí) a příkaz se provede. Výrazy se vyhodnocují vždy zleva doprava, závorky se nepoužívají a také se nebere ohled na přednost operací, například přednost násobení a dělení před sčítáním a odečítáním, na to je potřeba dávat pozor.

Nyní se pokusíme program upravit s využitím symbolicky pojmenované konstanty „cekani“. Z hlediska délky zápisu programu jsme si nepomohli, ta se dokonce prodloužila, délka programu přenášeného do procesoru zůstala stejná (44 byte), nesrovnatelně se však usnadnila možnost vyzkoušení různé rychlosti běhu. Změnami hodnoty konstanty na začátku programu si toto vyzkoušíme.

```
1 REM Program pro PICAXE-20M
2 REM Světelný had - 2
3 symbol cekani = 200
4 start:
5 high 0 pause cekani low 0
6 high 1 pause cekani low 1
7 high 2 pause cekani low 2
8 high 3 pause cekani low 3
9 high 4 pause cekani low 4
```

```
10 high 5 pause cekani low 5
11 high 6 pause cekani low 6
12 high 7 pause cekani low 7
13 goto start
```

Poznámka: Programy zde uvedené lze stáhnout z internetových stránek *www.hobbyrobot.cz* a načíst do obslužného programu procesorů PICAXE, nicméně jejich samostatné přepsání pomůže zapamatování si syntaxe příkazů i pochopení funkce.

Proměnné a přiřazení

PROMĚNNÁ

je místo v paměti, kam můžeme uložit nějakou hodnotu a používat ji. Po zapnutí napájení procesoru mají všechny proměnné hodnotu 0. V PICAXE-20M máme k dispozici celkem 14 proměnných jednobytových (pro hodnoty 0 až 255) pojmenovaných B0 až B13, samozřejmě si je můžeme v programu označit vlastním symbolickým jménem. Je-li potřeba uložit vyšší hodnoty v rozsahu 0 až 65535, můžeme stejný prostor v paměti využít jako 7 proměnných dvoubytových (typ word) značených W0 až W6, přičemž proměnné B0 a B1 tvoří společně proměnnou W0, W1 vzniká z proměnných B3 a B4 atd. Můžeme používat vedle sebe v programu jak jedno tak dvoubytové proměnné, musíme se ale vyhnout tomu, abychom se nechtěně na stejné místo v paměti procesoru odvolávali jednou jako na proměnnou jednobytovou (typu byte) a jindy jako na dvoubytovou (typu word), takže pokud potřebujeme třeba W5, už bychom neměli v programu používat B10 a B11 (a naopak). Podrobnější údaje jsou v (*2).

Jednoduchou pomůckou, jež nám umožní udržet si přehled o použitých pamětech a jejich určení, je následující tabulka. Tu si nakreslíme ve větším formátu a namnožíme do zásoby. Při psaní programu vždy, když potřebujeme nějakou proměnnou, zaznamenáme to do tabulky včetně významu. Které proměnné jsou volné v daném programu je pak vidět na první pohled. Kromě toho formou komentáře poznamenáme význam použité proměnné i přímo do programu.

B0	W0
B1	
B2	W1
B3	

...

B8	W4
B9	
B10	W5
B11	
B12	W6
B13	

PŘIŘAZENÍ

Zadání hodnoty do proměnné neboli přiřazení je jednoduché, například LET W6 = 123 uloží do proměnné W6 hodnotu 123. Z toho plyne, že bychom už neměli používat proměnné B12 a B13, které obsazují stejný prostor. Slovo LET, které uvádí přiřazení, se ani nemusí v naprosté většině případů používat. Součástí přiřazení může být i výraz s konstantami, symbolicky pojmenovanými konstantami, jinými proměnnými (případně i symbolicky pojmenovanými proměnnými) a samozřejmě s operátory.

INC, DEC

Velmi často je v programu třeba přičíst k nějaké proměnné číslo 1 nebo naopak 1 odečíst, to vedlo k tomu, že tyto operace byly zjednodušeny až vznikly samostatné příkazy INC a DEC, které používáme podobně jako operátory. Za nimi je uvedena proměnná, jejíž obsah se má zvýšit (snížit) o 1.

Například INC W0 je totéž jako napsat LET W0 = W0 + 1, tedy významově: Vezmi obsah proměnné W0, zvyš jej o 1 a výsledek ulož zpět do proměnné W0.

Je vidět na první pohled, že řádky programu světelného hada jsou vzájemně podobné až na měnící se nasměrování výstupu (příkazy HIGH a LOW). Zkrátíme program za pomoci proměnné a využijeme k tomu i příkaz INC a operátor zbytek po dělení (*3). Zkrácení zápisu je velmi výrazné, kdybychom počítali jeden příkaz na jeden řádek, vyšlo by z původních 29 řádků omezení na 10 a podobně je to i s délkou programu nahrávaného do procesoru, ta se z 44 byte „smrskla“ na 16. V programu jsou už použity komentáře k vysvětlení významu každého řádku, takto často psané komentáře nejsou v praxi nutné, ale přinejmenším význam proměnných a malých úseků bychom komentářem opatřit vždy měli. Program vyzkoušíme simulací, měl by z hlediska výstupů fungovat stejně, ale na zápisu programu je vidět, že zatímco předtím jeden cyklus programu pokryl celé přejetí světla od kraje do kraje, nyní probíhá cyklus mnohem rychleji, pro každý výstup zvlášť.

```
1     REM Program pro PICAXE-20M
2     REM Světelný had - 3
3     symbol cekani = 200        ;doba aktivace každého výstupu
4     start:                     ;začátek smyčky programu
5       high B0                  ;výstup podle B0 do stavu H
6       pause cekani             ;počkej podle konstanty cekani
7       low B0                   ;výstup podle B0 do stavu L
8       inc B0                   ;zvyš hodnotu proměnné B0 o 1
9       B0 = B0 // 8             ;do B0 dej zbytek po dělení 8 (čísla 0 až 7)
10      goto start               ;zpět na smyčku programu
```

Při simulaci máme v pravé polovině okna výpis stavu všech proměnných a přepínačem si můžeme zvolit, zda se na ně budeme dívat jako na 14 proměnných typu byte nebo 7 proměnných typu word a zda požadujeme výpis v hexadecimální soustavě. Sledování stavu proměnných za běhu programu je jedním z nejefektivnějších nástrojů při hledání chyb. Mezi nákresem procesoru a tabulkou hodnot proměnných můžeme najít tři tlačítka pro řízení simulace, zdola je to spuštění / zastavení, pozastavení (pauza) a krokování po jednom příkazu.

Cykly v programu

CYKLUS FOR ... NEXT

Má-li se část programu vícekrát zopakovat a známe předem počet potřebných opakování, je to příležitost pro FOR cyklus (*8). K počítání průchodů se používá“řídící“ proměnná, podle počtu musí být typu byte (0 až 255) nebo word (0 až 65535). Základní tvar vypadá takto:

```
for b0 = 1 to 50
 .. ; něco
 next b0
```

B0 je v tomto případě proměnná cyklu, 1 je spodní mez pro dosazení v cyklu, 50 je horní mez. Protože používáme hodnoty mezí jen z rozsahu 0 až 255, může být proměnná typu byte. To „něco“, co se má provést, se vykoná celkem 50x, přičemž při prvním chodu je hodnota v proměnné B0 rovna 1, při druhém 2 atd. V cyklu můžeme využívat i hodnotu proměnné cyklu, ale nikdy nesmíme tuto hodnotu měnit. Cyklus končí příkazem NEXT, jenž má v parametru příslušnou proměnnou cyklu, zde NEXT B0. Je-li třeba, aby se průchody cyklem nepočítaly po jedné nahoru, ale jinak, můžeme zadat krok (step).

```
for b0 = 1 to 50 step 2
 .. ; něco
 next b0
```

Tento kousek programu projde „něco“ postupně pro hodnoty B0 rovny 1; 3; 5; 7 atd., skončí, když B0 bude větší než 50. Krok lze zadat i záporný, docílíme tak počítání směrem dolů, v tom případě ale pochopitelně musíme zadat první mez větší než druhou.

```
for b0 = 50 to 1 step -2
 .. ; něco
 next b0
```

Je dobré si uvedené úryvky programu samostatně vyzkoušet, za „něco“ můžeme dosadit třeba bliknutí stavem jednoho z výstupů používané v předešlých programech. V režimu simulace se v rozšířeném okně za běhu ukazují i hodnoty všech proměnných, tam můžeme sledovat, jak se mění B0, a také si všimnout, že dva předchozí příklady se neliší jen tím, že v prvním proměnná cyklu roste a v druhém klesá, ona totiž jednou nabývá lichých hodnot (vychází od 1 a přičítá číslo 2) a podruhé sudých (vychází z 50 a odečítá 2).

```
1       REM Program pro PICAXE-20M
2       REM Světelný had - 4
3       symbol cekani = 200             ;doba aktivace každého výstupu
4       start:                          ;začátek smyčky programu
```

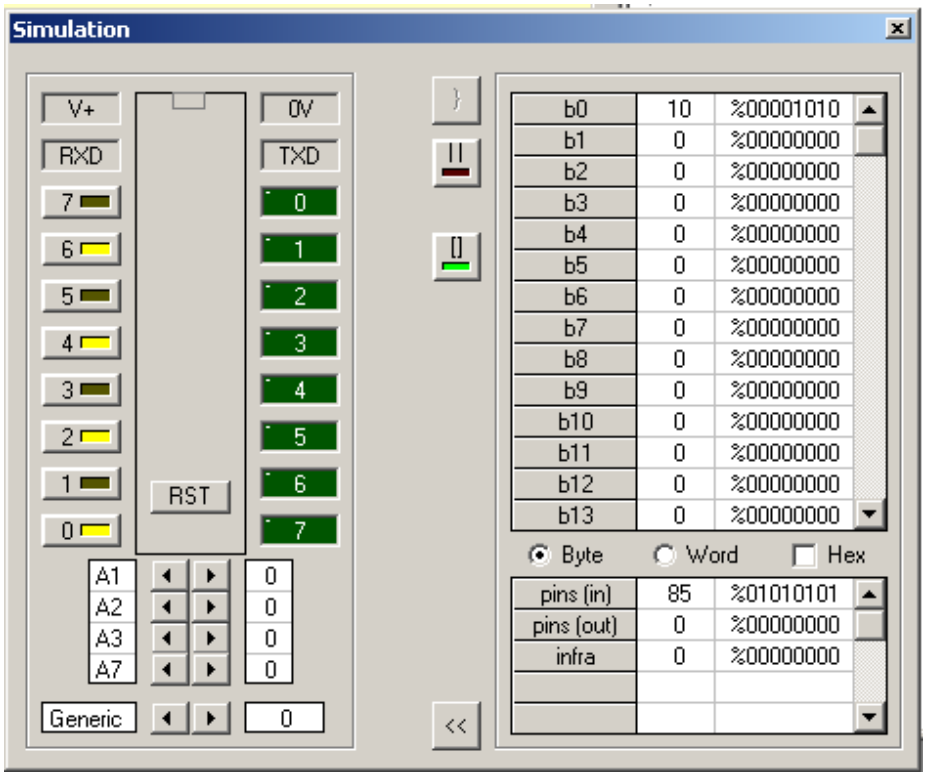
```
5       for b0 = 0 to 7                 ;cyklus pro b0 od 0 do 7
6       high b0                         ;nastav výstup podle b0 do H
7       pause cekani                    ;počkej podle konstanty cekani
8       low b0                          ;nastav výstup podle b0 do L
9       next b0                         ;konec cyklu
10      goto start                      ;zpět na nekonečkou smyčku programu
```

Čtvrtá verze programu pro světelného hada využívá právě cyklus FOR a je určitě přehlednější než předchozí verze s přičítáním jedničky do proměnné a směrováním výstupu pomocí dělení modulo 8 (zbytku po dělení 8). FOR cykly se používají velmi často a mohou být vnořeny i vícenásobně do sebe, každý z cyklů ovšem musí mít svou vlastní řídící proměnnou (zde B0) a vnitřní cyklus nesmí měnit hodnotu proměnné vnějšího cyklu.

V tomto programu jsme také poprvé použili odsazování začátku řádku v zápise, které přispívá k přehlednosti a orientaci v programu. Na stejnou úroveň by měly být odsazené příkazy, které k sobě patří a vykonávají se postupně (zde příkazy cyklu). Při programování se často pro přehlednost nechávají návěští vypsaná od levého kraje a všechny ostatní příkazy se výrazně odsazují (například o 6 znaků) a dál pak člení dalším odsazením, tuto zvyklost ale nebudeme dodržovat kvůli tomu, že příliš rozšiřuje zápis programu a v tisku musíme vyjít s podstatně užším místem, než jaké máme k dispozici při programování v editoru.

Čtení vstupů

Ještě chvíli setrváme u simulace. Zatím jsme pracovali jen s výstupy, dokázali jsme uvést výstup do stavu H nebo L, což bylo při simulaci vidět na zeleném poli u příslušného výstupu světlou nebo tmavou barvou. Vstupy jsou ve schématu obvodu vyznačeny jako tlačítka ze žlutou kontrolkou, můžeme na ně kliknout myší a změnit stav vstupu. Na obrázku jsou všechny sudé vstupy aktivní (stav H), liché neaktivní (stav L).



Pokud si chceme změnu vstupů vyzkoušet, musíme napsat nějaký program, aby nás obslužný program vůbec do simulace pustil. Může posloužit nekonečná smyčka, program, který jsme napsali úplně první. Jak hodnotu ze vstupu použijeme? Všechny vstupy a výstupy procesoru PICAXE-20M můžeme v programu popsat jako pin0 až pin7 a použít v přiřazovacím příkazu. Podle toho jestli bude „pin“ na jeho pravé nebo levé straně, půjde o vstup nebo výstup hodnoty a akce bude také podle toho směrována na jiný vývod procesoru. Například budeme-li číst pin3, pak budeme číst stav vývodu 7, zapíšeme-li hodnotu (nastavíme stav) pin3, projeví se to na vývodu 15.

Konkrétně příkaz **`b5 = pin1`** uloží do proměnné B5 hodnotu 0, pokud je vstup pin1 ve stavu L nebo hodnotu 1, pokud je ve stavu H. Zápis **`pin1 = 1`** je stejný jako příkaz **`high 1`**, zápis **`pin1 = 0`** je totéž jako **`low 1`** . Příkaz **`pin1 = b5`** vezme nejméně významný bit z proměnné B5 a podle něj pošle na výstup pin1 hodnotu 0 nebo 1 (stav L nebo H). Jinak řečeno, pro všechna sudá čísla v proměnné B5 bude výstup v L, pro všechna lichá v H. Lze dokonce zkopírovat přímo stav vstupu na výstup bez účasti proměnné, třeba **`pin4 = pin3`** přečte stav vývodu 7 a přenese jej na vývod 14. Znovu si všimneme, jak se stav použité proměnné zobrazuje v pravé části okna simulace a stav všech vstupů a výstupů vyjádřený jediným bytem také v jeho dolní části.

Vyjádření všech osmi vstupů (0 až 7) nebo všech osmi výstupů (0 až 7) jako bitů jednoho bytu nazvaného „pins“ předznamenává také jinou možnost, jak se s vývody dá zacházet. Výhodou je, že tímto způsobem lze přečíst všech osm vstupů v jediném okamžiku, což jinak nejde. I dva po sobě následující příkazy, které přečtou dva konkrétní vstupy, vyžadují určitý čas, a program nezaznamená stav u obou přesně současně. Podobně to platí i o výstupech. Zatímco při postupném zápisu výstup po výstupu je mezi sousedními prodleva kolem 0,3 ms, při společném zápisu se výstupy aktivují současně. 0,3 ms jsou 3/1000 sekundy, obvykle pro nás tak krátká doba nic neznamená, ale je řada programů, v nichž to už vadí. (Ve skutečnosti ani v tomto případě nejdou výstupy zcela přesně stejně, ale stejně jen ve skupinách 01-234-567 a posunutí mezi skupinami nepřevyšuje 7,3 mikrosekundy.)

Při zpomalené simulaci je postupné zapínání jednotlivých výstupů nebo jejich současné zapnutí zcela jasně vidět. K předvedení použijeme následující programy. Kromě zjednodušení se využitím „pins“ program i zkrátí (z 32 byte na 14) a zrychlí, chceme-li vidět zkrácení názorněji, stačí v prvním případě psát na jeden řádek jen jeden příkaz a pak porovnat.

```
1       REM Program pro PICAXE-20M
2       REM Blikání všemi výstupy 1
3       start:
4         high 0 high 1 high 2 high 3       ;zapnutí všech
5         high 4 high 5 high 6 high 7
6         pause 200                         ;čekání 0,2 s
7         low 0 low 1 low 2 low 3           ;vypnutí všech
8         low 4 low 5 low 6 low 7
9         pause 200                         ;čekání 0,2 s
10        goto start
```

Podmínky, větvení

Velmi často je potřeba program rozvětvit, to znamená podle hodnoty nějaké proměnné nebo stavu vstupu pokračovat buď jednou nebo druhou cestou. K tomu slouží následující příkaz.

IF … THEN … ELSE … ENDIF

Za IF následuje podmínka, většinou proměnná, s kterou budeme pracovat, pak relační operátor (nejčastěji =, <, > nebo <> (nerovná se)), konstanta nebo druhá proměnná, jejíž hodnotu porovnáváme s první (*11). Za slovem THEN bude v nejjednodušším případě symbolická adresa, kam má program skočit podobně jako by tam byl příkaz GOTO. Není-li podmínka splněna, pokračuje program následujícím příkazem v řadě. Například „if w6 >150 then zapni“ porovná hodnotu proměnné W6 s konstantou 150; je-li větší, skočí na návěští „zapni“, je-li menší nebo rovna, nedělá nic a pokračuje dalším příkazem.

Příkaz IF může mít i jinou, širší podobu. Za THEN nemusí následovat jen adresa, ale také přímo jeden nebo více příkazů, které se provádí, je-li podmínka splněna, pak může následovat slovo ELSE a za ním opět jeden nebo více příkazů, jenž se provádí při nesplnění podmínky. Příkaz IF končí slovem ENDIF. Jde tedy zapsat

if w6>150 then high 4 else low4 endif

což znamená: Je-li hodnota W6 větší než 150, nastav Pin 4 na H, v opačném případě nastav Pin 4 na L.

Předchozí příkazy si vyzkoušíme simulovat na příkladu světelného hada ovládaného vstupem pin0, bude-li vstup v H, světlo poběží, bude-li v L, budou jen na krajních výstupech krátké záblesky.

```
1       REM Program pro PICAXE-20M
2       REM Světelný had - 5
3       symbol svit = 100               ;doba svitu dílku hada
4       symbol zablesky = 10            ;délka záblesků
5       start:                          ;začátek smyčky programu
6         if pin0=1 then                ;testování pin 0 -> je-li v H
7           for b0 = 0 to 7             ;pro všech osm výstupů
8             high b0                   ;nastav výstup podle b0 do H
9             pause svit                ;počkej podle konstanty svit
10            low b0                    ;nastav výstup podle b0 do L
11            next b0                   ;konec cyklu
12          else                        ;... je-li vstup 0 v L
13          pins = %10000001            ;rozsviť 0 a 7
14          pause zablesky              ;doba svitu při záblesku
15          pins = 0                    ;zhasni vše
16          endif                       ;konec větvení
17        pause 500                     ;čekání po průběhu cyklu
18        goto start                    ;zpět na smyčku programu
```

Napájení procesoru PICAXE-20M

Přiblížil se čas práce se skutečným procesorem. Význam jednotlivých vývodů byl uveden v minulém dílu seriálu. K napájení procesoru slouží vývod 1 (+Ucc), na něj přivedeme kladné napětí, a vývod 20 (GND), což je zem (ground). Vůči zemi budeme měřit všechna napětí v zapojení, na ní zapojíme záporný vývod zdroje. I když procesor je většinou schopen pracovat ve větším rozpětí napětí, jeho napájení by nemělo klesnout pod 4,5 V a nikdy by nemělo přesáhnout 5,5 V. Při napětí kolem 6 V dochází k tomu, že se smaže celý obsah paměti procesoru včetně toho programu, který nahrál do paměti výrobce a který dělá z „obyčejného“ procesoru PIC příslušného typu procesor PICAXE. Pokud by se nám „povedlo“ tento program smazat, je procesor pro naši práci už nepoužitelný a budeme muset koupit nový. Běžným používáním procesoru se to nestane, ale zvýšení napětí je právě jedna z mála možností, jak to dokázat.

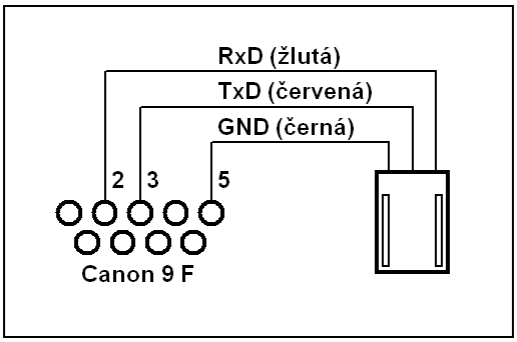
Je třeba si uvědomit, že skutečné napětí se často může dost lišit od jmenovitého napětí, které se uvádí na bateriích. Použijeme-li k napájení běžné (primární) baterie, ty mají jmenovité napětí 1,5 V na jeden článek, po zatížení však jejich napětí rychle klesá i pod 1 V. Takové baterie nejsou pro napájení procesoru vhodné, čtyři články v klidu dávají 6 V (4x1,5), což je více než povolené napětí, a při zatížení často méně než 4 V. Pokud to jde, budeme procesor napájet ze stabilizovaného napětí 5 V, univerzálnější ale pro naše pokusy bude pouzdro se čtyřmi tužkovými (AA) nebo mikrotužkovými (AAA) akumulátory NiCd nebo NiMH. Akumulátory mají jmenovité napětí 4,8 V (1,2 V na článek) a skutečné se pohybuje mezi 5,4 V (chvíli po nabití) a 4,5 V, to už ale jsou téměř úplně vybité. Napětí na procesoru nikdy nesmíme obrátit (přepólovat), zničil by se.

Přenos programu do procesoru

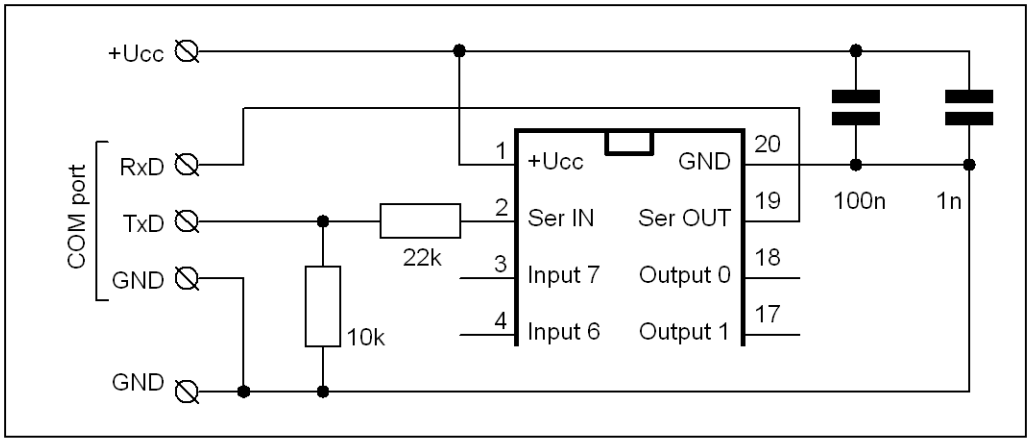
Procesory PICAXE nepotřebují k programování speciální drahé programátory ani přípravky, stačí jednoduchý sériový kabel do PC, z něhož dokonce využijeme jen tři vodiče. S výhodou lze k práci s PICAXE použít starší počítač, nové počítače a notebooky už dokonce ani klasické sériové porty (COM) nemívají, v takovém případě musíme koupit k počítači převodník rozhraní USB na COM. I pokud bychom měli k dispozici speciální programátor na procesory PIC (a PICAXE přece je založen na procesoru PIC), nikdy jej nepoužíváme! Takový programátor by zacházel s naším procesorem jako s „holou“ součástkou a smazal z paměti PICAXE program od jeho výrobce, čímž by z našeho hlediska procesor znehodnotil.

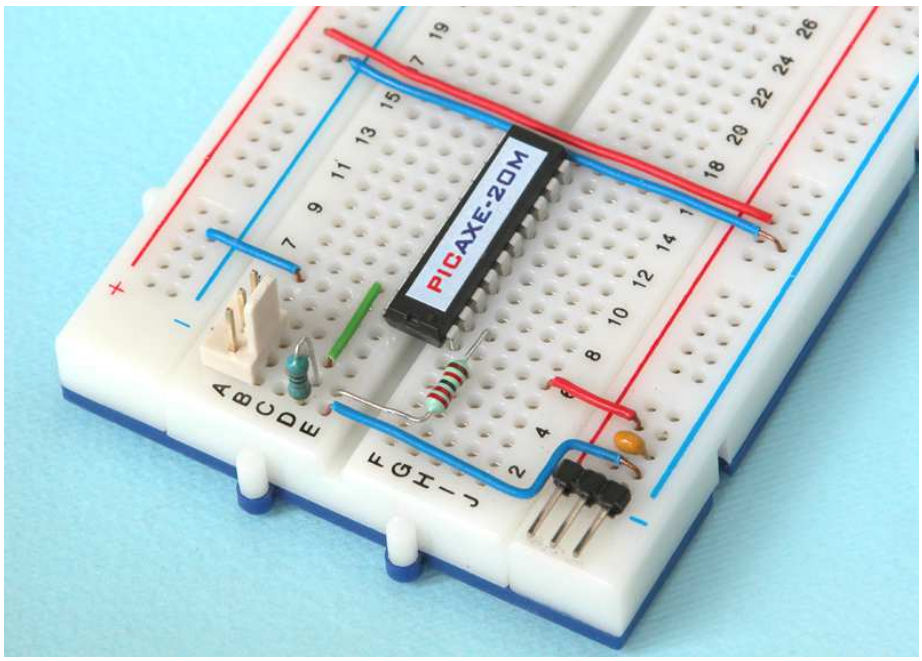
PICAXE lze programovat přímo v našem pokusném zapojení, pokud v něm vyvedeme na nějaký konektor zem (GND), vývod 2 (Serial IN, vstup sériových dat) a vývod 19 (Serial OUT, výstup sériových dat). Základní zapojení obvodu je ve schématu. Jakmile se na vstupu Serial IN objeví úroveň H, procesor přeruší svou práci a okamžitě se připraví na příjem nového (uživatelského) programu, který přemaže ten starý. Hned jak je nový program přijat, spustí se a začne vykonávat. Nemusíme mít obavy z častého přeprogramování procesoru (respektive uživatelského programu), průměrně by paměť měla vydržet asi 100000 přepisů, takže pokud bychom každý den vyměnili náš program 100x, vydržel by to procesor přibližně tři roky.

Pokud není procesor připojený na programovací kabel, musí být zajištěno, že na Serial IN bude napětí blízké zemi, o to se stará dvojice rezistorů ve schématu. Výstup sériových dat je nutný jednak v průběhu zavádění nového uživatelského programu, jednak dovoluje za běhu programu přenášet do PC data potřebná třeba k hledání chyb. Ještě je třeba zajistit, aby napájecí napětí procesoru bylo blokované kondenzátory, které pohltí špičky napětí a vykryjí nepatrné nárazové odběry proudu, čímž se omezí možnost rušení jak od našeho zařízení do okolí (třeba rádia, televize, ..), tak od jiných okolních zařízení do toho našeho. Na napájení blízko k procesoru zapojíme běžný keramický kondenzátor 100n a pokud možno k němu ještě druhý typu NPO s kapacitou 1n právě kvůli odrušení na vyšších kmitočtech.



Programovací kabel můžeme zakoupit (*www.snailshop.cz*) nebo vyrobit sami, zapojení je na schématu. Stačí obstarat devítikolíkový konektor Canon (s dutinkami, ne s kolíky, protikus sériového konektoru v PC), tenký třížilový kabel dlouhý jeden až dva metry (například servokabel) a konektor PFH02-03P s kontakty.





Základní zapojení procesoru osadíme na kontaktním poli přibližně tak, jak ukazuje fotografie. Konektoru sériové linky nejprve opatrně narovnáme ohnuté vývody a potom jej zasuneme do pole. Typ, který by měl rovné kontakty, nejde použít, vývody jsou krátké. Konektor již nevyndáváme, vývody jsou silnější a rozehnou kontakty v poli, takže na stejném místě by už slabší vodiče špatně držely. Procesor můžeme osadit do precizní objímky, vzhledem k tomu, že se s ním prakticky nehýbe, to však není nutné. Pro napájení použijeme tříkolíkový konektor ze zahnuté propojkové lišty, kladné napájení je uprostřed, zem na jednom kraji, druhý krajní kontakt odštípneme. Takto nelze otočením orientace konektoru napájení přepólovat.

K napájení zkušebního zapojení použijeme pravděpodobně čtyři tužkové akumulátory NiMH, jaké se v domácnosti běžně používají spolu s jednoduchými síťovými nabíječkami. Plastové bateriové pouzdro s vývodem zakončeným servokonektorem, tedy řadovým konektorem s třemi kontaktními dutinkami a roztečí 2,51 mm, se prodává v modelářských prodejnách, případně si můžeme pouzdro s drátovými vývody opatřit konektorem sami. Kontakty jsou lisovací a protože nelze předpokládat, že bychom měli k dispozici vhodné lisovací kleště, raději nastavíme vývod servokabelem zakoupeným v modelářské prodejně. Kontakty není vhodné pájet. Konektor napájení má kladný vývod na prostřední dutince (červený vodič), a záporný na jednom kraji (černý nebo hnědý vodič), druhá krajní dutinka je nezapojená (bílý nebo žlutý vodič odstraníme).

Pro napájení zkušebního zapojení s procesorem je nejlepší stabilizovaný zdroj 5,0 V, proud stačí do 0,5 A. Výběrem a konstrukcí takového zdroje se budeme zabývat v samostatném článku, zatím postačí baterie. Pokud bychom nechtěli použít akumulátory (jmenovité napětí 1,2 V / článek) ale ZnC nebo alkalické články (jmenovité napětí 1,5 V / článek), zapojíme pro jistotu přímo do napájecího kabelu jednu křemíkovou usměrňovací diodu dimenzovanou na 0,7 A. Na ní se vytvoří úbytek kolem 0,5 V, takže se napětí na procesoru o něco sníží.Na kontaktním poli máme základní zapojení procesoru, připojený programovací kabel a napájení. Vrátíme se o dva díly zpět k prvnímu pokusnému programu, nahrajeme jej nebo zapíšeme znovu, můžeme vyzkoušet simulaci, ale tentokrát tlačítkem „program“ přeneseme program do procesoru. Je-li vše zapojené správně a také v nastaveních je zvolen odpovídající typ procesoru, proběhne „modrá linka“, vypíše se délka programu v bytech a celková velikost

dostupné paměti. Jestliže programování nemůže proběhnout, zobrazí se připomínka nejčastějších chyb: nepřipojení programovacího kabelu, odpojené napájení procesoru, použití prázdného procesoru PIC (tedy nikoli PICAXE), nutnost provést reset procesoru nebo nezapojená propojka programování (platí pro firemní výukové přípravky). Vznik některé z chyb si záměrně vyzkoušíme. Program je uvnitř a děje se přesně to co má, totiž nic.

PICAXE 20M	1 ks
Konektor 3pin programovací	1 ks
Kabel programovací CABAXE025	1 ks
Konektor napájecí 2 pin	1 ks
Bateriové pouzdro s konektorem	1 ks
R 22k	1 ks
R 10k	10 ks
R 1k5	5 ks
R 330	10 ks
R 820	2 ks
C 100n keramický	1 ks
C 220M elektrolytický	1 ks
D BAT43	1 ks
D 1N4148	4 ks
Sokl DIL 20	1 ks
LED rudá rozptylná průměr 3	8 ks
LED zelená rozptylná průměr 3	1 ks
LED žlutá rozptylná průměr 3	1 ks
Piezoreproduktor PT-1540T	1 ks
LED IR	1 ks
Fototranzistor IR	1 ks
QRB1134 (odrazový infrasenzor)	1 ks
SN754410 (řízení motorů)	1 ks
Kontaktní pole dlouhé (64 řad)	1 ks
Motor s převodovkou	2 ks
Tlačítko mini nízké	2 ks
DIL switch 2x	1 ks
Sedmisegmet LED	1 ks
Termistor NTC 640-12k	1 ks
Pinová lista 2,54 mm 10 pin	1 ks
Drát 0,5 mm	10 m

Sada součástek

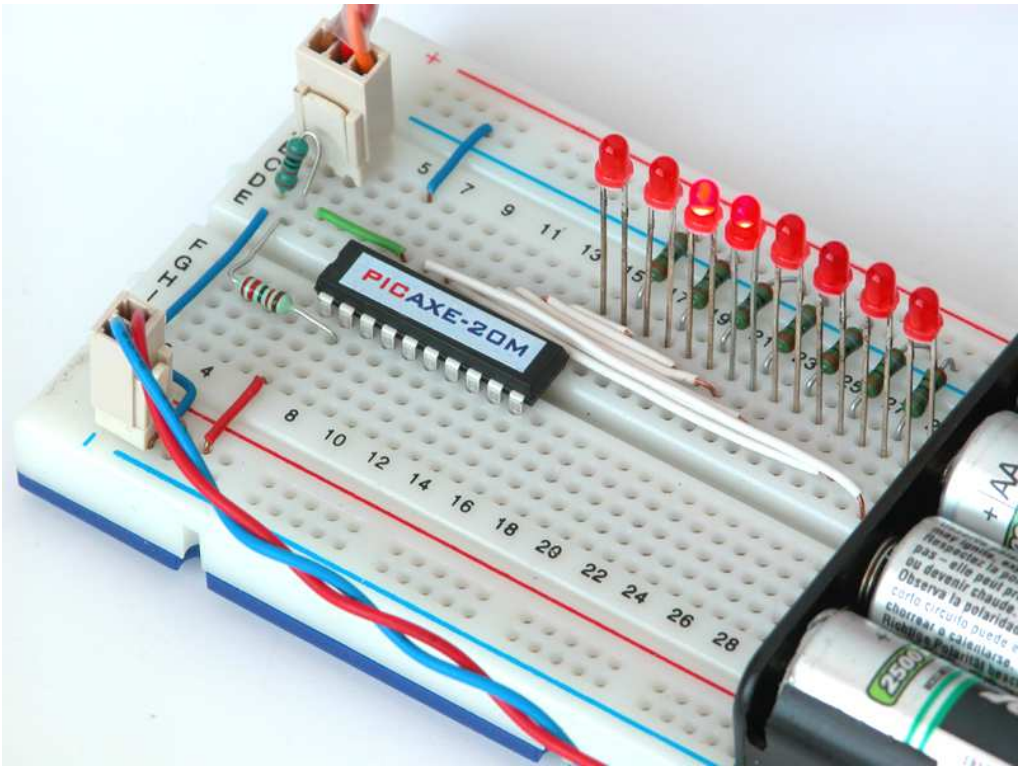
K vyzkoušení příkladů, jenž postupně probíráme, budou potřeba součástky podle připojené tabulky. Kdo chce, může si je sehnat jednotlivě, protože je však stěží půjde koupit na jednom místě, zkompletovala firma Hobbyrobot ucelenou sadu, kterou si můžete objednat na internetové adrese www.snailshop.cz za 580 Kč včetně DPH. Příklady představí obsluhu jednotlivých čidel na vstupu i výkonných prvků na výstupu a typické algoritmy, později na ně naváže pokračování, které jednotlivé pokusy propojí a přímo z naší kontaktní desky se zapojením nechá vzniknout jednoduchého pohyblivého víceúčelového robotka, jemuž budeme říkat „Robrouk“.

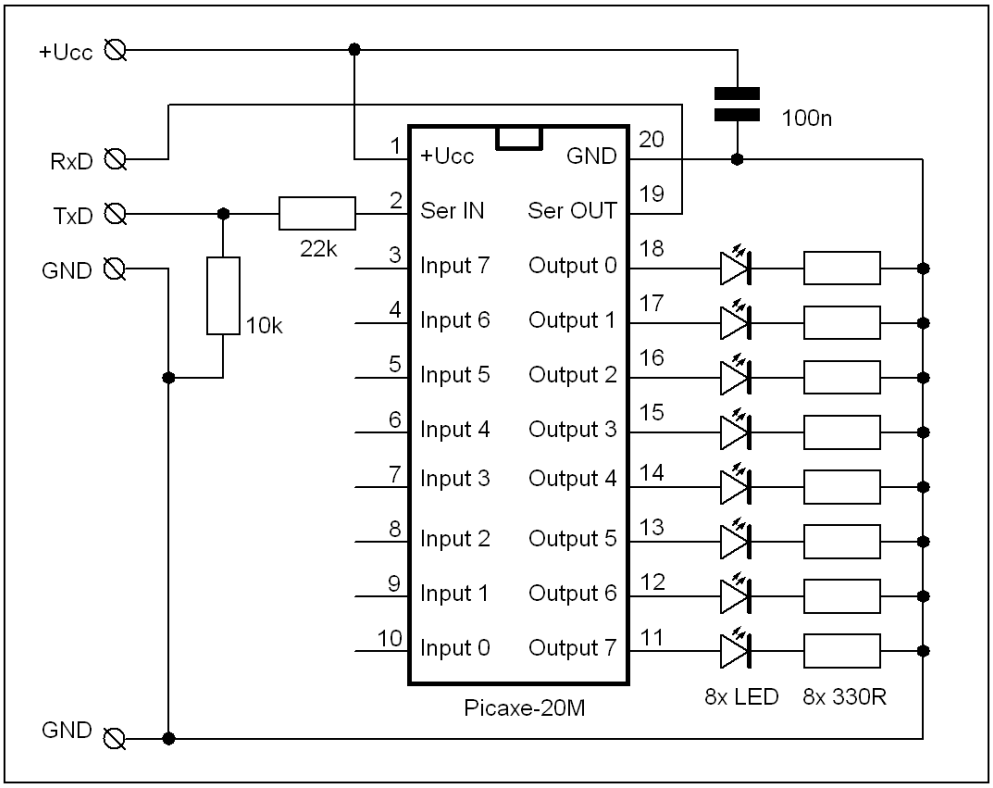
Pokusy, které už známe

Připravíme a vyzkoušíme druhý program „Blikající LED“ z předminulého dílu. Na pin 7 připojíme LED delším vývodem, od kratšího vedeme k rozvodu země rezistor 330Ω a program přeneseme. LED by se měla rozblikat, což je první viditelný projev našeho procesoru. Otestujeme i funkci programu „Blikající LED 2“

Rozšířením použitého zapojení na 8 LED podle schématu si připravíme půdu pro běh programů Světelný had 1 až Světelný had 4 a blikání všemi výstupy. Abychom nezůstali jen u již uvedených programů, napíšeme si jeden nový, v pořadí šestou verzi „hada“. Úkolem je zařídit, aby světlo běhalo střídavě z jedné strany na druhou, sem a zpátky. V podstatě můžeme vyjít z kteréhokoli předchozího programu, blok příkazů zařadit dvakrát za sebe

a v druhé polovině změnit smysl běhu světla, upravíme-li Světelného hada 4 a trochu zrychlíme, pak třeba takto:





```
1       REM Program pro PICAXE-20M
2       REM Světelný had - 6
3       symbol cekani = 100             ;doba aktivace každého výstupu
4       start:                          ;začátek smyčky programu
5         for b0 = 0 to 7               ;cyklus pro b0 od 0 do 7
6           high b0                     ;nastav výstup podle b0 do H
7           pause cekani                ;počkej podle konstanty cekani
8           low b0                      ;nastav výstup podle b0 do L
9           next b0                     ;konec cyklu
10        for b0 = 7 to 0 step -1       ;cyklus pro b0 od 7 do 0
11          high b0                     ;nastav výstup podle b0 do H
12          pause cekani                ;počkej podle konstanty cekani
13          low b0                      ;nastav výstup podle b0 do L
14          next b0                     ;konec cyklu
15        goto start                    ;zpět na nekonečkou smyčku programu
```

Program takto funguje, ale ukážeme si na něm zcela jiný přístup k problému, zápisem i výsledným kódem delší, jímž však můžeme vytvořit prakticky libovolný světelný efekt. Využijeme příkazu pins pro zápis stavu všech osmi výstupů naráz. Program „Světelný had 7“ zbavíme veškeré chytrosti, bude jen brát jeden předem zapsaný stav po druhém a posílat jej na výstup. Zatím napodobíme jen hada běhajícího v jednom směru.

```
1       REM Program pro PICAXE-20M
2       REM Světelný had - 7
3       symbol cekani = 200             ;doba aktivace jednoho stavu
4       start:                          ;začátek smyčky programu
5         pins = %00000001 pause cekani
6         pins = %00000010 pause cekani
7         pins = %00000100 pause cekani
8         pins = %00001000 pause cekani
9         pins = %00010000 pause cekani
10        pins = %00100000 pause cekani
11        pins = %01000000 pause cekani
12        pins = %10000000 pause cekani
13        goto start                    ; ... a znovu
```

Využili jsme zápis čísla ve dvojkové soustavě, protože nejnázorněji pozicí jedniček ukazuje, které LED svítí. Navrhneme a vyzkoušíme si další efekty, třeba světlo běžící od krajů ke středu nebo opačně, postupné rozsvícení všech LED a pak jejich zhasnutí ve stejném pořadí nebo podpoříme dojem posunování světla tím, že vždy svítí dvojice LED vedle sebe a tyto dvojice se posunují. Blok příkazů „pins“ pro poslední příklad (Světelný had 8) by mohl vypadat takto:

```
pins = %00000001 pause cekani
pins = %00000011 pause cekani
pins = %00000110 pause cekani
pins = %00001100 pause cekani
pins = %00011000 pause cekani
```

```
pins = %00110000 pause cekani
pins = %01100000 pause cekani
pins = %11000000 pause cekani
pins = %10000000 pause cekani
```

Pokročilá obsluha LED

Zatím jsme vždy LED jen rozsvítili do plného jasu nebo zhasli, ostatně nic jiného s jedním výstupem procesoru ani dělat nejde. Přesto můžeme jas LED řídit, ovšem za tu cenu, že obětujeme naprostou většinu času našeho procesoru a ten už zřejmě nestihne dělat nic jiného. Princip je jednoduchý, LED rozblikáme tak rychle, aby už naše oko blikání nevnímalo. Poměr doby, kdy LED svítí, a kdy je zhasnutá, pak určuje intenzitu jasu. Zkusíme rozsvítit pro začátek jednu LED na pinu 7 nejprve do částečného a pak plného jasu. Má-li být rozdíl jasu zřetelně viditelný, musí být doba svitu podstatně kratší, často 10 - 20% z celku, takže záblesk vytvoříme příkazy pro rozsvícení a zhasnutí řazenými těsně za sebe a k tomu najdeme vhodnou dobu vypnutí.

```
1       REM Program pro PICAXE-20M
2       REM Obsluha LED - polojas
3       start:                      ;začátek smyčky programu
4         pins = 0                  ;zhasnout všechny LED
5         pause 1000                ;počkat 1s
6         for w0 = 1 to 200         ;200x krátce bliknout LED 7
7           high 7
8           low 7
9           pause 3
10          next w0
11        high 7                    ;rozsvítit LED 7
12        pause 1000                ;počkat 1s
13        goto start                ; ... a znovu
```

Složitějším případem je kombinace plného a částečného svitu třeba ve světelném hadu. Musíme zajistit, aby se v době, kdy jednoduchá verze jen čekala, střídaly dva „obrazy“ přenášené na LED, jeden je uložen v proměnné B0, druhý v B1. Dohromady vytvoří dojem „doznívajícího světla“. Budeme potřebovat další příkaz.

GOSUB ... RETURN

Gosub je příkaz skoku podobně jako Goto, ale na rozdíl od něj si pamatuje, odkud byl skok proveden, a když program narazí na povel Return, vrátí se zpátky za příkaz, z něhož k odskoku došlo. Tu část programu, jež se má na zavolání Gosub vykonat, nazýváme podprogramem, a musíme samozřejmě její začátek označit návěštím. Počet použití příkazů Gosub je omezen, podrobněji viz (*9).

```
1       REM Program pro PICAXE-20M
2       REM Obsluha LED - polojas 2
3       start:                                  ;začátek programu
4         b0=%00000001 b1=%00000001 gosub zobraz
```

```
5       b0=%00000011 b1=%00000010 gosub zobraz
6       b0=%00000111 b1=%00000100 gosub zobraz
7       b0=%00001110 b1=%00001000 gosub zobraz
8       b0=%00011100 b1=%00010000 gosub zobraz
9       b0=%00111000 b1=%00100000 gosub zobraz
10      b0=%01110000 b1=%01000000 gosub zobraz
11      b0=%11100000 b1=%10000000 gosub zobraz
12      b0=%11000000 b1=%00000000 gosub zobraz
13      b0=%10000000 b1=%00000000 gosub zobraz
14      goto start                              ; ... a znovu
15    zobraz:                                   ;podprogram zobrazení
16      for w1 = 1 to 10                        ;10x vystřídat zobrazení
17        pins = b0
18        pins = b1
19        pause 10
20        next w1
21      return                                  ; ... a znovu
```

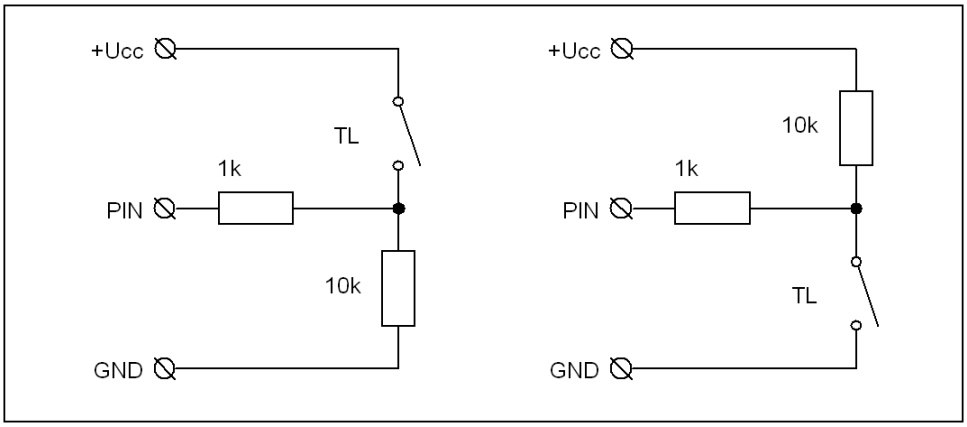
Co se týče připojení LED k procesoru, můžeme je zapojit jak mezi výstup a zem (rozsvítí se ve stavu H), tak mezi výstup a +5V (rozsvítí se ve stavu L), v obou případech samozřejmě s rezistorem omezujícím proud. V případě, že by bylo nutné osadit dvoubarevnou LED se dvěma vývody, zapojíme ji i s rezistorem mezi dva výstupy procesoru. Při rozdílném stavu na výstupech pak svítí jedna z LED (základní barvy), při stejném stavu jsou obě zhasnuty a rychlým přepínáním dosáhneme složené barvy.

Obsluha tlačítek a spínačů

Mechanické spínací kontakty patří mezi to nejčastější, co se připojuje ke vstupům procesoru. Nezapojené vstupy zpravidla vykazují stav L, ale obecně jsou ve stavu vysoké impedance, což znamená, že stačí i nepatrný podnět, aby svůj stav změnily. Můžeme si to názorně vyzkoušet následujícím programem, který čte stav všech vstupů a výsledek bezprostředně kopíruje na výstupy osazené LED. V klidu budou pravděpodobně LED zhasnuté (na vstupech je L), jakmile se však dotkneme nezapojených vstupů prstem nebo vodivým předmětem, přenese se na vstup rušení a LED se rozsvítí.

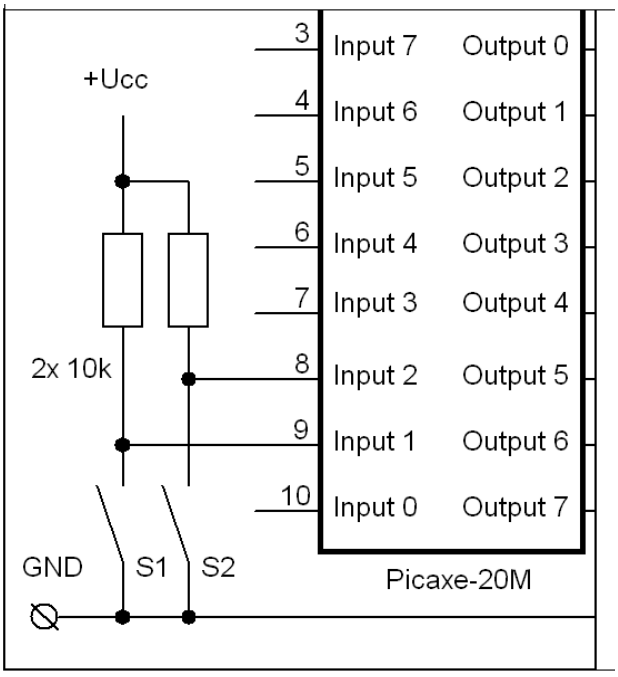
```
start: b0=pins pins=b0 goto start
```

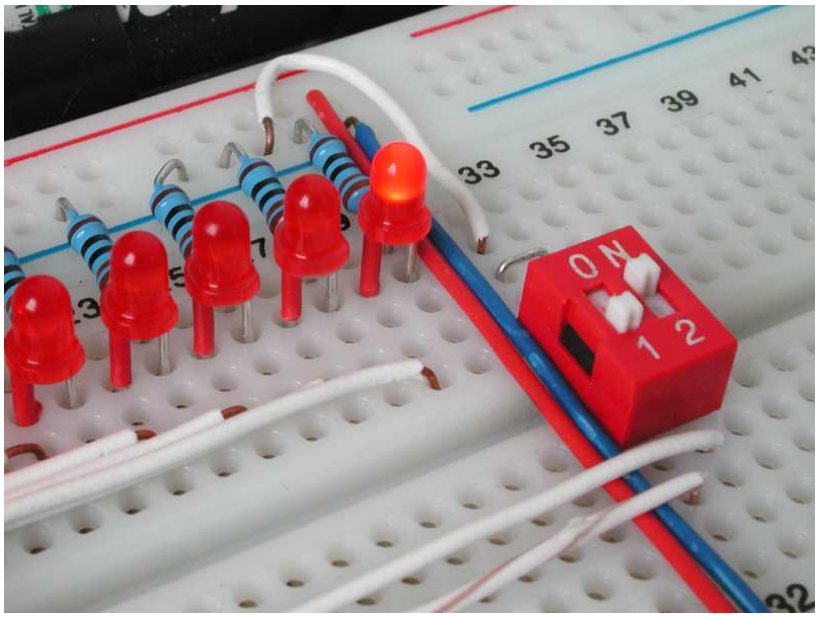
Aby se jasně definovala úroveň na vstupu procesoru, připojíme jej k zemi nebo +5V přes rezistor 4,7k až 10k. Rezistor 1k sériově zapojený ke vstupu zlepšuje funkci a omezuje proudové špičky při nabíjení nebo vybíjení kapacity vstupu, ale nutný není a ve většině pokusů jej nebudeme používat.



Jeden program využívající spínač jsme již zkoušeli, i když jen simulací. Byl to „Světelný had 5“. Nahrajeme jej, osadíme mezi vstup 7 a zem tlačítko a vstup „přitáhneme“ ke kladnému napájení rezistorem. Program by měl v klidu ukázat běžící světlo, při stisknutí se rychle rozblikají krajní LED.

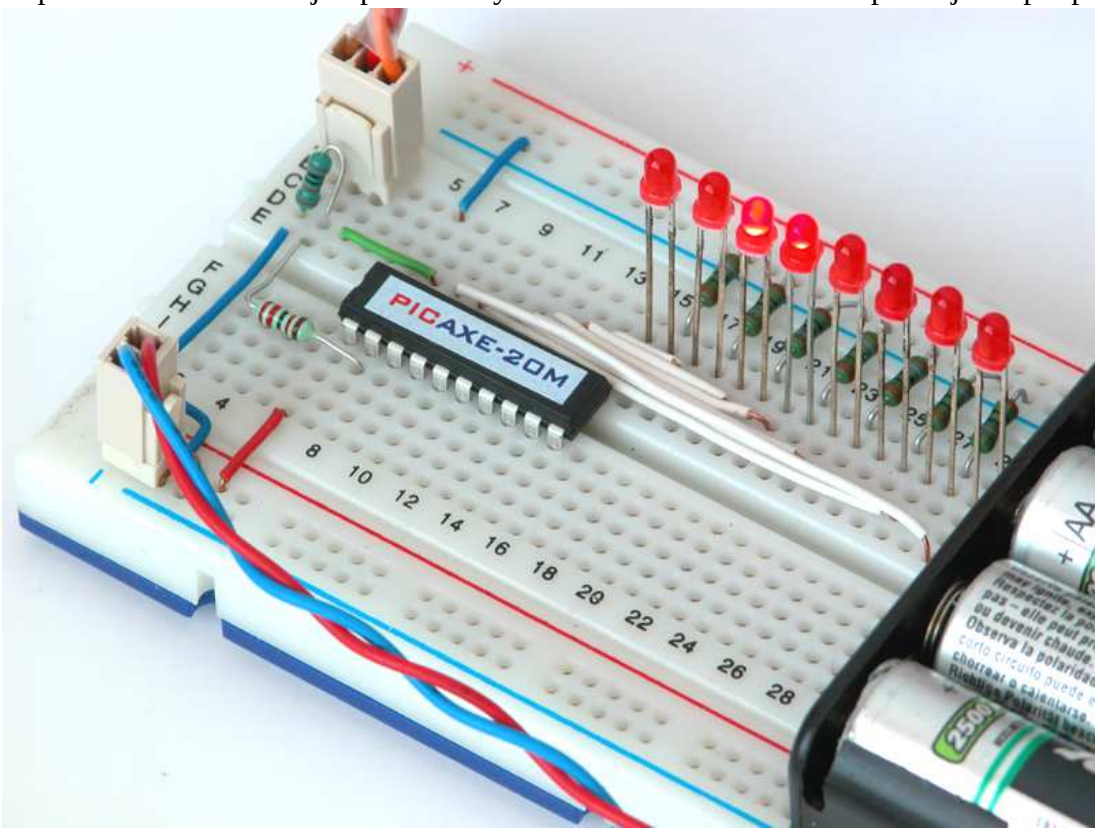
Je-li potřeba nastavit spínači více parametrů běhu programu, používáme zpravidla řadové DIL spínače, jeden dvojitý je i součástí naší sady součástek. Zapojíme jej tak, aby spínač 1 spojoval vstup 1 se zemí, analogicky spínač 2 spojuje vstup 2 se zemí, na obou vstupech při rozpojených spínačích zajišťují spolehlivý stav H rezistory 10k. Program upravíme, spínač 1 bude ovládat rychlost běhu světla (hodnota proměnné „svit“), spínač 2 směr pohybu. Režim lze změnit kdykoli za chodu, na běžící světlo už ale vliv nemá, změna se projeví až od následující periody.






```
1       REM Program pro PICAXE-20M: Světelný had - 9
2       REM Input 1 (vývod 9) rychlost běhu
3       REM Input 2 (vývod 8) smysl běhu
4       symbol svit = b1                        ;doba svitu dílku hada
5       start:                                  ;začátek smyčky programu
6         if pin1=0 then let svit=25 else svit=75 endif
7         if pin2=0 then                        ;testování pin 2 - směr
8           for b0=0 to 7                       ;pro všech osm výstupů
9             high b0                           ;nastav výstup podle b0 do H
10            pause svit                        ;počkej podle konstanty svit
11            low b0                            ;nastav výstup podle b0 do L
12            next b0                           ;konec cyklu
13          else                                ;... nebo druhý směr
14          for b0=7 to 0 step -1               ;pro všech osm výstupů
15            high b0                           ;nastav výstup podle b0 do H
16            pause svit                        ;počkej podle konstanty svit
17            low b0                            ;nastav výstup podle b0 do L
18            next b0                           ;konec cyklu
19          endif                               ;konec větvení
20        pause 1000                            ;čekání po průběhu cyklu
21        goto start                            ;zpět na smyčku programu
```

Pro čtení tlačítka včetně odstranění zákmitů při stisku a simulaci opakovaného stisku (autorepeat) je určen příkaz Button se sedmi parametry. Česká příručka jen odkazuje na anglický originál, podle popisu v něm však příkaz nefunguje, dokonce lze říci, že žádnou smysluplnou činnost se z něj nepodařilo vyloudit. Tato zkušenost koresponduje i s příspěvky



zahraničních uživatelů PICAXE v diskusních fórech. Naštěstí příkaz prakticky není třeba a tlačítka lze velmi jednoduše obsloužit prostým čtením stavu na vstupu a případně čekáním na změnu (puštění) nebo zpomalením funkce, které překryje náhodné zákmity mechanických kontaktů.

LED se dodávají s dlouhými vývody, které při použití kontaktního pole spíš překáží, proto je zkrátíme asi na 10 mm. Delší vývod LED připojujeme na kladné napájení, kratší na záporné, tuto informaci o polaritě bychom neměli zkrácením ztratit, takže předem označíme kladný pól navléknutím asi 5 mm červené izolační bužírky stažené z drátu.

Práce s napětím

Procesor PICAXE 20M je vybaven AD převodníkem, takže může bez dalších součástek číst napětí na některých vstupech, převést jej na číslo a s ním dále pracovat. Převod může fungovat jako osmibitový s ukládáním výsledku do jednobytové proměnné (hodnoty 0 – 255) nebo jako desetibitový (0 - 1023) s ukládáním výsledku do proměnné typu word. Vzhledem k tomu, že převod pracuje v mezích tvořených zemí a napájecím napětím procesoru (nikoli s přesnou referencí) a protože není příliš přesný, používá se desetibitový převod spíše výjimečně a když už, tak především pro jemnější rozlišení změn. Osmibitový převod poskytuje uspokojivé výsledky, vyhneme-li se oblastem těsně kolem nuly a plného napájecího napětí. Procesor PICAXE 20M má možnost pracovat s napětím na čtyřech vstupech a to 1, 2, 3 a 7 (vývody 9, 8, 7 a 3). K přečtení napětí ze vstupu budeme potřebovat následující příkazy.

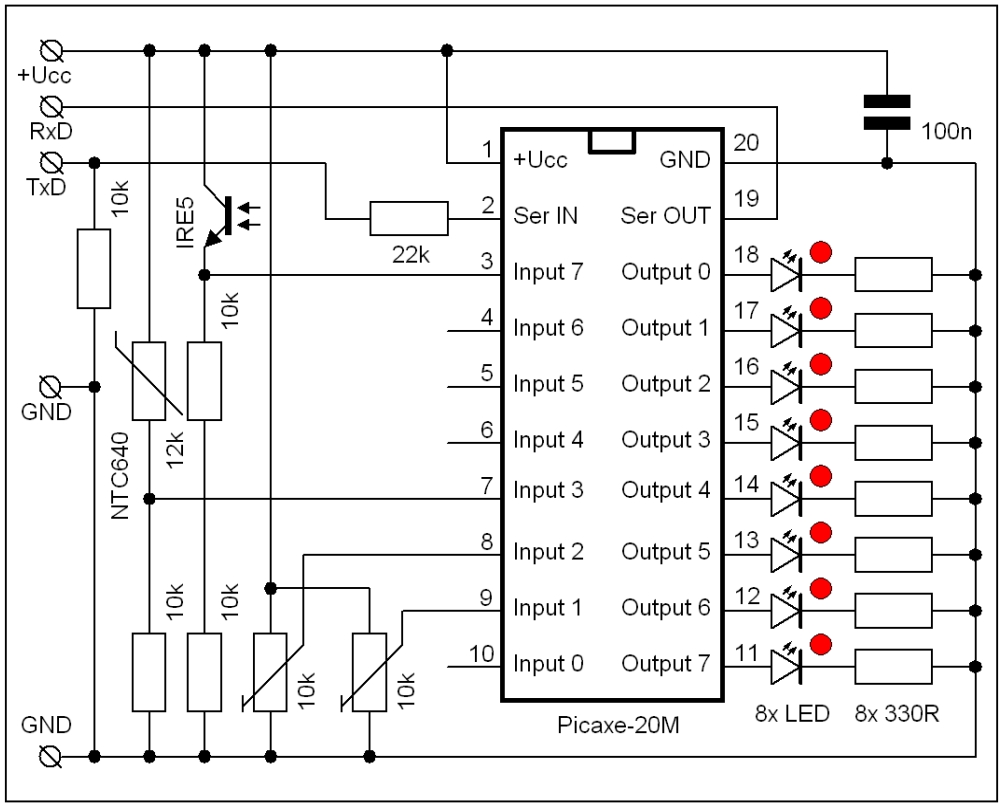
READADC

má dva parametry, prvním je pin, na němž se bude napětí sledovat, druhým je proměnná, do níž se výsledek převodu uloží. Readadc 1,b0 tedy přečte napětí z Pin1 (vývod 9 procesoru), převede na osmibitové číslo (0 - 255)v poměru k napájecímu napětí a výsledek uloží do proměnné b0. (*20)

READADC10

Pracuje obdobně jako Readadc, jen proměnná musí být typu word. Poskytuje hodnoty 0 až 1023. (*20)

Ze zapojení odstraníme spínače, řadu LED ponecháme, na vstupy 1 a 2 zapojíme odporové trimry, na vstup 3 termistor a na 7 IR fototranzistor podle schématu. Trimry mají širší vývody než je možné zasunout do pole, takže na ně buď připájíme krátké nástavce z tenčího drátu nebo opilujeme vývody ze stran asi na polovinu, což je řešení pracnější, ale mechanicky pevnější a celkově lepší. Rozpoznání fototranzistoru a IR LED je jednoduché, LED má čiré pouzdro, tranzistor tmavé. K prvnímu vyzkoušení využijeme trimr na vstupu 1, jezdec nastavíme přibližně do středu dráhy. Po spuštění následujícího programu by se mělo ukázat okno vyvolané příkazem Debug s výpisem stavu proměnných aktualizované zhruba po půl sekundě, hodnota b0 by měla být někde kolem 128. Šroubovákem pomalu otáčíme jezdcem a můžeme sledovat, jak se stav mění. Současně můžeme sledovat změny na řadě LED, kde se načtená hodnota napětí zobrazuje v binárním kódu. V jedné krajní poloze trimru by měly být všechny LED zhasnuté (napětí blízké 0V), v druhé všechny rozsvícené (napětí blízké napájecímu).

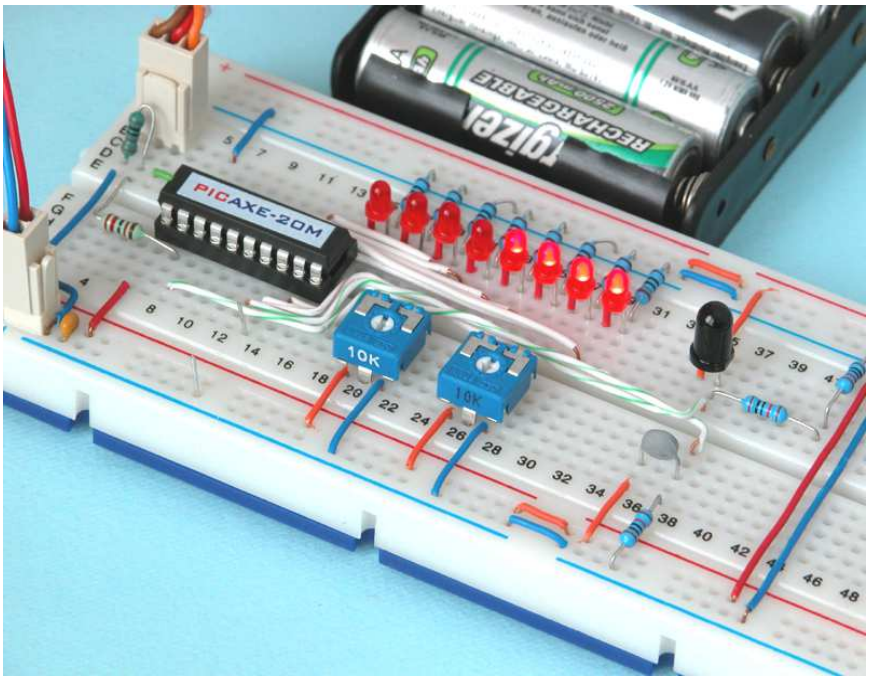


```
1       REM Program pro PICAXE-20M: Analog 1
2       start:                      ;začátek smyčky programu
3         readadc 1,b0              ;načíst napětí z Input 1
4         pins=b0                   ;hodnotu poslat na výstupy
5         debug                     ;přenos všech pamětí do PC
6         goto start                ;zpět na smyčku programu
```

DEBUG

se netýká přímo převodu, je univerzálním prostředkem pro ladění programů, někdy slouží i k přenosu výsledných hodnot do počítače. Formálně má jeden parametr označující proměnnou, ale protože tento parametr je nepovinný a bez praktického významu, nebudeme jej používat. Příkaz po připojeném programovacím kabelu vyšle do PC hodnoty všech proměnných a pinů a ty se zobrazí v přehledné tabulce. Sériový přenos poměrně velkého množství dat výrazně zpomalí běh programu, takže se tento příkaz zpravidla nehodí pro odlaďování programů vyžadujících synchronizaci s vnějšími událostmi. (*6).

V dalším kroku program modifikujeme pro desetibitový převod, místo Readadc použijme Readadc10 a místo bytové proměnné b0 dvoubytovou proměnnou w0. Ve výpisu proměnných ověříme, že vše pracuje tak jak má, nicméně na LED se zobrazí jen spodní byte proměnné w0, takže se při pomalém najíždění napětí počítání „protočí“ mezi minimem a maximem čtyřikrát. Vrátíme osmibitový převod i proměnnou na své místo a pokusíme se naprogramovat jednoduchý voltmetr, který by vracel do PC změřenou hodnotu napětí v desetinách V. Budeme předpokládat, že napájecí napětí procesoru je 5 V. Čtyřčlánek Nixx, který pravděpodobně k pokusům používáme, má zřejmě napětí menší, což omezuje přesnost funkce. Pokud by stejný procesor byl napájen stabilizovaným napětím 5,0 V, bude měření odpovídat velmi dobře. Při převodu potřebujeme vydělit číslo získané z AD převodníku konstantou 5,1 (255/5), pomůžeme si nejdřív vynásobením 10 a pak vydělením číslem 51. Samozřejmě musíme pracovat s proměnnou, do níž se vynásobené číslo vejde. V tomto programu nemá smysl používat výstup na LED.



```
1       REM Program pro PICAXE-20M: Analog 2
2       REM Jednoduchý voltmetr
3       start:                      ;začátek smyčky programu
4         readadc 1,w0              ;převod AD z Input 1
5         let w0=w0*10              ;dělení konstantou 5,1
6         let w0=w0/51
7         debug                     ;přenos všech pamětí do PC
8         goto start                ;zpět na smyčku programu
```

V další verzi upravíme zobrazení LED na lineární „páskový indikátor“, v němž bude svítit řada LED, jejíž délka je úměrná vstupnímu napětí. Příkaz Debug už nepoužijeme, aby se chod programu zbytečně nezpomaloval, a zobrazení rychle sledovalo otáčení jezdcem trimru. U tohoto příkladu se zastavíme déle a ukážeme si více způsobů jeho řešení a také jejich výhody a nevýhody.

```
1       REM Program pro PICAXE-20M: Analog 3
2       REM Bodový indikátor
3       start:                      ;začátek smyčky programu
4         readadc 1,b0              ;načíst napětí z Input 1
5         b0=b0/32                  ;dělit 32 (2^8)
6         pins=0                    ;zhasnout všechny LED
7         for b1=0 to b0
8           high b1 next b1         ;dle b0 rozsvítit počet LED
9         goto start                ;zpět na smyčku programu
```

Program je jednoduchý a krátký (21 byte), nicméně první LED svítí vždy a upravit tento program na jiný (nelineární) průběh zobrazení už bude trochu obtížnější, protože aritmetika není silnou stránkou procesorů PICAXE. Zkusíme jinou metodu, „otrocké“ definování mezí přímo v programu. Bylo by určitě elegantnější definovat meze jako konstanty, ale zápis programu by se tím výrazně prodloužil a na jádru věci to nic nemění. Nový program je delší zápisem i délkou potřebné paměti v procesoru (59 byte, tedy více než 2x), ale budeme-li chtít změnit zobrazení na nelineární, je to velmi jednoduché, stačí změnit parametry.

```
1       REM Program pro PICAXE-20M: Analog 4
2       REM Bodový indikátor páskový
3       start:                          ;začátek smyčky programu
4         readadc 1,b0                  ;načíst napětí z Input 1
5         pins=0                        ;zhasnout všechny LED
6         if b0>16 then high 0 endif    ;obsluha LED
7         if b0>48 then high 1 endif
8         if b0>80 then high 2 endif
9         if b0>112 then high 3 endif
10        if b0>144 then high 4 endif
11        if b0>176 then high 5 endif
12        if b0>208 then high 6 endif
13        if b0>240 then high 7 endif
14        goto start                    ;zpět na smyčku programu
```

Program funguje dobře, ale při rozsvícení všech LED si lze všimnout, že jejich jas není stejný, nejvíc svítí ta „dolní“, která se rozsvěcí při nejnižším napětí, nejméně ta na druhém konci řady. Důvod je jednoduchý, běh programu trvá nějakou dobu a ty LED, které jsou po zhasnutí všech obslouženy později, svítí vlastně kratší dobu, tedy méně intenzivně. Dá se domu zabránit několika způsoby, například za obsluhu LED (před Goto) vložíme čekání (řekněme 200 ms), které rozdíly mezi svitem jednotlivých LED relativně zmenší za cenu zpomalení reakce na změnu, nebo třeba nebudeme pásek jedním povelem zhasínat a pak postupně LED rozsvěcet, ale v každém příkazu obsloužíme rozsvícení nebo zhasnutí (viz následující program Analog5), ovšem za cenu prodloužení na 84 byte, tedy čtyřnásobek původní verze.

```
1     REM Program pro PICAXE-20M: Analog 5
2     REM Bodový indikátor páskový
3     start:                                  ;začátek smyčky programu
4       readadc 1,b0                          ;načíst napětí z Input 1
5       if b0>16 then high 0 else low 0 endif
6       if b0>48 then high 1 else low 1 endif
7       if b0>80 then high 2 else low 2 endif
8       if b0>112 then high 3 else low 3 endif
9       if b0>144 then high 4 else low 4 endif
10      if b0>176 then high 5 else low 5 endif
11      if b0>208 then high 6 else low 6 endif
12      if b0>240 then high 7 else low 7 endif
13      goto start                            ;zpět na smyčku programu
```

Můžeme zkusit změnit program Analog4 na bodové zobrazení, v němž vždy svítí jen jedna LED. Uvedený způsob vede k dlouhému kódu (112 bytů, téměř polovina celkové kapacity paměti), ale na druhou stranu umožňuje snadné změny mezí a také jejich překrývání, čímž je možné jemněji vyjádřit hodnotu vstupního napětí. Příklad může sloužit i jako ilustrace toho, že programování jednočipových procesorů je často hledáním kompromisu mezi optimální funkcí, možnou délkou kódu v paměti a obvodovou jednoduchostí, případně cenou. Jak se dá podobný problém někdy zjednodušit použitím dalších integrovaných obvodů si ukážeme později.

```
1     REM Program pro PICAXE-20M: Analog 6
2     REM Bodový indikátor bodový
3     start:                                  ;začátek smyčky programu
4       readadc 1,b0                          ;načíst napětí z Input 1
5       if b0<=48 then high 0 else low 0 endif
6       if b0>48 and b0<=80 then high 1 else low 1 endif
7       if b0>80 and b0<=112 then high 2 else low 2 endif
8       if b0>112 and b0<=144 then high 3 else low 3 endif
9       if b0>144 and b0<=176 then high 4 else low 4 endif
10      if b0>176 and b0<=208 then high 5 else low 5 endif
11      if b0>208 and b0<=240 then high 6 else low 6 endif
12      if b0>240 then high 7 else low 7 endif
13      goto start                            ;zpět na smyčku programu
```

Analogové vstupy často nacházejí uplatnění ve spojení s jednoduchými čidly teploty, světla nebo polohy (natočení), dvě z nich máme na vyzkoušení v sadě součástek. Prvním je termistor zapojený do děliče napětí. Při pokojové teplotě kolem 25°C je jeho odpor mezi 10 a 12 kΩ a s rostoucí teplotou klesá, napětí na vstupu procesoru se pohybuje kolem 2,5 V. Jako teploměr můžeme použít například následující program, s omezeným rozsahem zobrazení, přesné meze je nutné doladit podle konkrétního čidla. Ve výchozím stavu by při pokojové teplotě měla svítit 4. LED, po zahřátí dotykem ruky nebo pod žárovkou se světlo rychle vyšplhá až na mez.

```
1       REM Program pro PICAXE-20M: Teploměr
2       start: readadc 3,b0
3         pins=0
4         if b0<=110 then high 0 endif
5         if b0>110 and b0<=115 then high 1 endif
6         if b0>115 and b0<=120 then high 2 endif
7         if b0>120 and b0<=125 then high 3 endif
8         if b0>125 and b0<=130 then high 4 endif
9         if b0>130 and b0<=135 then high 5 endif
10        if b0>135 and b0<=140 then high 6 endif
11        if b0>140 then high 7 endif
12        goto start
```

V reálných zapojeních se termistor jako čidlo použije asi jen tehdy, bude-li záležet na nízké ceně, ale ne na přesnosti a délce programu, může třeba hlídat přehřátí motoru. Procesory PICAXE dovedou přímo spolupracovat s digitálními čidly DS18B20 měřícími teplotu v rozsahu –55 až +123°C s přesností na jeden stupeň, k měření slouží jediný příkaz Readtemp (*21) a připojení nevyžaduje vstup s AD převodníkem.

Fototranzistor IRE5 na vstupu 7 je zapojen podobně, jen potřebuje větší odpor do děliče, ten získáme sériovým spojením dvou až tří rezistorů 10k. K otestování poslouží krátký program pro bodové zobrazení, který ještě zvyšuje citlivost. V tomto případě nebude podstatný program, ale vyzkoušet si vlastnosti čidla. IRE5 reaguje téměř výhradně na infračervenou (tepelnou) část spektra, takže i bez další elektroniky registruje spolehlivě dálkové ovládání od televize na vzdálenost 10 až 30 cm, je citlivý i na světlo žárovky, ale ani silné ozáření LED svítilnou s bílým světlem se nijak neprojeví. Čidlo je výrazně směrové, katalog udává záběr 20° (maximální odchylka 10° od osy), takže se uplatní jednak jako součást světelných závor, jednak pro zaměřování polohy na orientační majáčky se známou polohou.

```
1       REM Program pro PICAXE-20M: test IR
2       start: readadc 7,b0 b0=b0/16
3         if b0>7 then let b0=7 endif pins=0 high b0
4         goto start
```

Reflexní senzor QRB1134

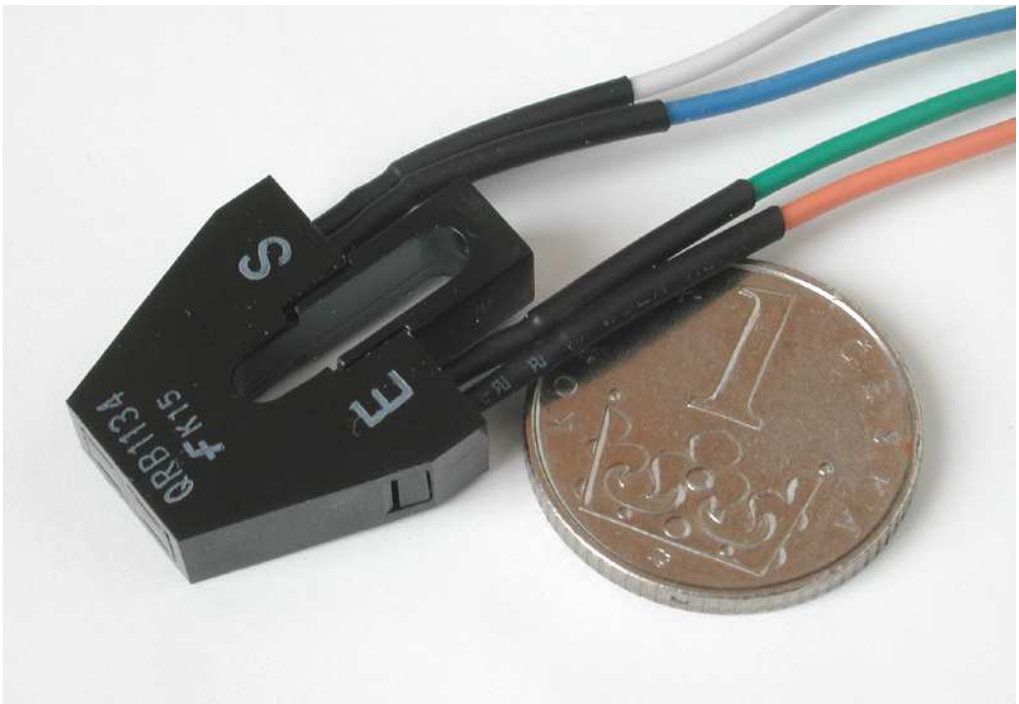
Pro zjištění těsného přiblížení k překážce můžeme použít odrazový senzor QRB1134. Součástka se dodává včetně 60 cm dlouhých vývodů, bílý a modrý pár patří fototranzistoru

(modrý = emitor, bílý = kolektor), oranžový a zelený LED (oranžový = anoda, zelený = katoda). Optimální vzdálenost pro detekci je podle katalogu kolem 4 mm od čela senzoru, dá se použít v rozsahu 2 až 15 mm a pokud připojíme fototranzistor na AD převodník, i na vzdálenost delší. LED pracuje s proudem 20 až 40 mA, takže k ní použijeme dva paralelně spojené rezistory 330Ω. Tranzistor zpočátku zapojíme místo IRE5 na napěťový vstup 7 a vyzkoušíme s předchozím programem „Test IR“. Dlouhé vodiče dovolují pohybovat čidlem, přibližovat jej k různým povrchům a na LED indikátoru sledovat výstupní signál. Čidlo pracuje v IR oblasti záření, takže může být rušeno světlem žárovek nebo přímého slunce, zářivkové osvětlení registruje málo a světlo LED (kromě IR) prakticky vůbec.

Čidlo není příliš vhodné pro úlohy typu sledování bílé čáry, protože mnoho materiálů, které se našim očím jeví jako výrazně různě barevné nebo světlé či tmavé má velmi podobnou odrazivost v IR spektru, naopak některé materiály pro nás téměř opticky stejné mohou mít různou odrazivost v IR, dost často se to stává třeba u černých plastů. Toho se dá i využít pro vytýčení téměř neviditelné cesty. Pro detekci přiblížení krátkého dosahu je toto čidlo velmi zajímavé, protože zjistí i překážky prakticky neviditelné pro ultrazvukové senzory, jako je tenká látka nebo papír. Je důležití si také uvědomit, že pro správnou funkci čidlo potřebuje určitou minimální vzdálenost od překážky a dojde-li už k dotyku, zakryje se okénko jak LED tak fototranzistoru a čidlo nic „nevidí“.

Po prvním seznámení zkusíme ve stejném zapojení funkci čidla bez AD převodníku. Jednoduchý program, ač na první pohled může vypadat nesmyslně, kopíruje stav ze vstupu 7 (tentokrát ovšem jen jako jeden bit) na výstup 7 s LED. Opět vyzkoušíme, jak čidlo reaguje na různé materiály.

```
start: pin7=pin7 goto start
```



Přenosy sériových dat

Kromě příkazu Debug, který přenese do PC po programovacím sériovém kabelu stav všech pamětí a pinů, je procesor PICAXE vybaven příkazy pro sériovou komunikaci, ať už s jiným procesorem nebo zařízením jako je třeba modul GPS, případně s nadřízeným počítačem. Jeden z těchto příkazů se efektivně využívá i k ladění programů, protože posílá data v textové formě, takže je můžeme snadno číst a zobrazit prostřednictvím terminálu, který náš editor v PC obsahuje. Proti příkazu Debud je objem dat malý, tudíž běh programu se zpomaluje mnohem méně, často bude dokonce nutné program přibrzdit, abychom vůbec číst stihli.

SERTXD

má za sebou v závorce seznam parametrů, které jsou brány jako ASCII znaky, a budou vyslány do počítače. Přenos probíhá pevně danou rychlostí 4800 Bd, bez parity (N), 8 bitů s 1 stopbitem, zkráceně zapsáno 4800,n,8,1. Jednotlivé parametry se oddělují čárkou. Chceme-li poslat konkrétní zadaný text, uvedeme jej v uvozovkách ("tohle se pošle"), pokud chceme poslat hodnotu proměnné, zapíšeme před její označení # (například #w0). Když uvedeme číslo, bude se brát jako kód znaku, který chceme poslat. (*25)

Odesílat znaky přes jejich kódy je většinou zbytečně pracné a rozhodně to není názorné. Pokud bychom to chtěli, tabulku znaků najdeme například na internetové adrese *www.labo.cz/mft/matasciit.htm*. V jednom případě ale děláme výjimku. Terminál bude zobrazovat přicházející znaky jeden za druhým a nijak nerozliší jednotlivé zprávy poslané jedním příkazem sertxd. To, že má odřádkovat, aby byl text přijatelně čitelný, musíme zadat sami. Znaky k tomu používané jsou dva, ten s kódem 13 (znak CR) obecně vrátí psaní na první pozici v řádku, znak 10 (LF) odřádkuje, řádek tedy zpravidla končíme sekvencí 13,10. Uvedený systém pochází ještě z dob dálnopisů a odpovídá i práci mechanického psacího stroje.

SEROUT

je obecnějším příkazem pro odesílání sériových dat. Jeho prvním parametrem je číslo pinu, na nějž bude výstup směrován, pak následuje způsob přenosu a rychlost, další parametry v závorce už jsou stejné jako v případě příkazu SERTXD. Kód způsobu přenosu a rychlosti je uveden písmenem T nebo N, T znamená normální přenos s klidovou úrovní True (vyšší napětí, logicky H), N znamená obrácené úrovně signálu s klidovou úrovní Negative (napětí blízké nule, logicky L). Číslo udává rychlost v Baudech. Přípustné rychlosti jsou 300, 600, 1200, 2400 a 4800, rychlost 4800 je tímto způsobem dostupná jen na procesorech -X. Všechny rychlosti se týkají hodinového kmitočtu procesoru 4 MHz. (*24)

Přenos dat je možný obousměrně, čímž můžeme ovládat své zařízení (program) přímo z PC a terminálu v něm, aniž by bylo nutné tyto prostředky nějak pracně programovat. Přenos do procesoru ale naráží na problém. Jakmile přijdou nějaká data po standardním programovacím kabelu na vývod Serial IN, spustí se vnitřní přerušení procesoru a ten se je snaží uložit jako nový program. Chceme-li posílat programu svoje data, musíme to uskutečnit přes jiný pin. Je-li to třeba, můžeme zapojit dva třípinové sériové konektory, jeden pro programování, druhý pro náš obousměrný přenos dat (řízení), a kabel přepojíme podle potřeby. Protože v sadě součástek dva třípinové konektory nejsou, pomůžeme si případně tím, že budeme signál TxD přepojovat drátovou propojkou na jiný vstup procesoru. Postačí-li přenos dat ze zařízení do PC, vystačíme si s původním programovacím konektorem a kabel ani nemusíme odpojovat.

Vrátíme se k programu Test IR a upravíme jej tak, aby se intenzita signálu od čidla QRB1134 přenášela do počítače a zobrazovala nejdříve čísly a potom v grafu. Program spustíme, přes menu – PICAXE – terminal (nebo klávesu F8) spustíme terminál, zkontrolujeme nastavení přenosové rychlosti 4800 Bd a vyzkoušíme přiblížení čidla k nějakému předmětu. Signál se současně zobrazuje i na pásku LED.

```
1       REM Program pro PICAXE-20M: test IR2
2       REM Snímání QRB1134 a přenos čísel
3       start: readadc 7,b0
4         b1=b0/16
5         if b1>7 then let b1=7 endif
6         pins=0 high b1
7         sertxd (#b0,13,10)
8         goto start
```

Úlohu rozšíříme o převod hodnot z termistoru. Proměnná W1 bude počítat jednotlivé vzorky měření. Program přeneseme do procesoru, pak odpojíme napájení a přes menu – PICAXE – Datalink (klávesu F9) spustíme datalink. V jeho Options nastavíme přenosovou rychlost 4800 Bd, počet senzorů 2 a případně zaškrtnutím políčka zvolíme grafické zobrazení. V menu – File – New spustíme nový záznam (případně neuložíme buffer), zapneme napájení procesoru a rychle odklepneme OK v okně čekajícím na jeho spuštění. Začne se vykreslovat graf, přiblížením reflexního čidla k předmětu na jedné křivce uvidíme signál nepřímo úměrný vzdálenosti od něj, průběh druhé můžeme mírně ovlivnit zahřátím termistoru rukou. Graf podle potřeby mění měřítko obou os. Chceme-li skončit, odpojíme napájení procesoru a zobrazování se po chvíli zastaví samo.

```
1       REM Program pro PICAXE-20M: test IR3
2       REM Snímání QRB1134 a termistoru - přenos do grafu
3       start:
4         readadc 7,b0                          ;převod z QRB1134
5         readadc 3,b4                          ;převod z termistoru
6         b1=b0/16                              ;úprava pro LED
7         if b1>7 then let b1=7 endif           ;omezení na 8 LED
8         pins=0 high b1                        ;zobrazení na LED
9         inc w1                                ;počítání vzorků
10        sertxd (#w1,",",#b0,",",#b4,13,10)    ;přenos do PC
11        pause 50                              ;zpomalení běhu
12        goto start
```

Všimneme si ještě jednou podrobně parametrů příkazu sertxd. Jako první se přenáší proměnná w1, v níž jsou vzorky očíslovány a podle toho vynášeny na osu x, pak jsou dvě proměnné (mohou být typu byte i word), všechny tři vzájemně oddělené textovými čárkami (ty dvě v uvozovkách), jednotlivé parametry oddělené čárkami. A konci je odřádkování číselně zadanými znaky CR LF (13,10). Můžeme zkusit, jak by vypadal výpis přenášených dat bez těchto posledních znaků.

Několikařádkový program spolu s režimem Datalink může zastoupit i jednoduchý pomaluběžný osciloskop a názorně ukázat časové průběhy na určitém vstupu nebo výstupu, stačí poslat příslušnou hodnotu na zobrazení do PC. Program „Pila“ ukazuje toto využití, svůj

průběh si ale generuje sám a využívá toho, že na postupně rostoucí proměnnou typu word se současně dívá ve spodní části jako na proměnnou typu byte, čímž vzniká průběh „vzestupné pily“. Ke grafu se dostaneme stejně jako v předchozím případě, jen musíme nastavit počet senzorů na 1.

```
1       REM Program pro PICAXE-20M: Pila
2       REM generování grafu - pily
3       start:
4         inc w0
5         sertxd (#w0,",",#b0,13,10)
6         goto start
```

Následující program jednoduše prověří možnost opačné komunikace. Do procesoru budeme posílat číslo, protože jsme v textovém režimu, tak vlastně číslice 0 až 9. Číslice 0 až 7 změní stav příslušné LED, číslice 8 nebo 9 všechny LED zhasnou. Využijeme dva nové příkazy. Prvním je *Serin* , který se zapisuje stejně jako *Serout*, jen místo vyslání dat očekává jejich přijetí. Druhým příkazem bude *Case*, ten není probírán v české příručce, v anglickém originále jej najdeme na straně 168.

select case / case / else / endselect

Příkaz zastupuje mnohonásobnou podmínku. Začíná slovem SELECT, za nímž je uvedena proměnná, podle jejíž hodnoty dochází k větvení. Potom jdou jednotlivé příkazy Case následované hodnotou (intervalem, podmínkou s relačními operátory), při níž se provede dále zapsaná sekvence příkazů. Na konci výčtu Case může být slovo Else a za ním část programu, která se provede ve všech ostatních případech. Zadané možnosti uzavírá slovo Endselect. Nenastane-li shoda v žádné z podmínek (a není použito Else), pak se nevykoná nic.

```
1       REM Program pro PICAXE-20M: Ovladani
2       REM Ovládání LED z PC po sériové lince
3       start:
4         serin 0,N2400,b0              ;načtení povelu
5         select b0                     ;větvení podle povelu
6           case 48 toggle 0            ;znak 0 - LED 0
7           case 49 toggle 1            ;znak 1 - LED 1
8           case 50 toggle 2            ;znak 2 - LED 2
9           case 51 toggle 3            ;znak 3 - LED 3
10          case 52 toggle 4            ;znak 4 - LED 4
11          case 53 toggle 5            ;znak 5 - LED 5
12          case 54 toggle 6            ;znak 6 - LED 6
13          case 55 toggle 7            ;znak 7 - LED 7
14          case 56 to 57 pins=0        ;znak 8,9 - vše zhasnout
15          endselect                   ;konec příkazu case
16        goto start
```

Před použitím programu musíme odpojit signál TxD (oba rezistory necháme spojené aby zajistily úroveň L na vývodu Ser IN procesoru) a přepojíme jej na Input 0 (vývod 10). Po vyzkoušení před novým programováním musíme samozřejmě spoje vrátit.

Modelářská serva

Jedním z velmi silných nástrojů procesorů PICAXE jsou jejich příkazy dovolující krajně jednoduché generování řídícího signálu pro modelářská serva i čtení a dekódování tohoto signálu. O tom, co lze od modelářských serv očekávat a jak se řídí, pojednávají jiné články v tomto čísle časopisu, takže přistoupíme rovnou k seznámení s příkazy. Protože servo není součástí připravené sady součástek pro experimenty, bude seznámení jen stručné a teoretické, později se k němu ale vrátíme při konstrukci robota.

PULSOUT

má dva parametry (*18), první udává Pin, s nímž se bude pracovat, druhý délku (jednoho) pulzu, který procesor vygeneruje. Pulz je opačný vůči úrovni, v níž je výstup na začátku, takže příkaz můžeme použít jak pro kladné tak záporné pulzy. Jednotkou pro délku pulzu je 10 mikrosekund při normálním hodinovém kmitočtu 4 MHz.

Pulsout je obecný a univerzální příkaz pro vytváření impulzů, má-li vytvořit jeden pulz pro řízení serva, bude mít tedy pro výstup 0 a střední polohu serva tvar *pulsout 0,150*, pro krajní polohy *pulsout 0,100* nebo *pulsout 0,200*. V reálním případě musíme tyto pulzy generovat každých přibližně 20 ms, nejjednodušší program pro udržování serva ve střední poloze by mohl vypadat takto:

```
start: pulsout 0,150 pause 19 goto start
```

SERVO

je speciální příkaz pro řízení serv (*25), podobně jako Pulsout prvním parametrem se udává Pin, druhým poloha serva respektive délka impulzu (neutrál odpovídá číslu 150). Zásadní rozdíl je v tom, že příkaz Servo využívá vnitřního časovače procesoru a automaticky opakuje zadaný pulz s periodou 20 ms až do další změny nastavení, procesor přitom nezávisle vykonává program dál. Potřebujeme-li ukončit ovládání serva, pošleme na příslušný výstup úroveň L (např. low 4).

Generování pulzů pro serva má úskalí, a tím je přesnost a stabilita pulzů v probíhajícím programu, serva jsou totiž poměrně citlivá a už na změny délky pulzů v řádu jednotek mikrosekund reagují chvěním a záškuby. Částečně si můžeme pomoci tím, že procesor zrychlíme ze základní frekvence hodin 4 MHz na dvojnásobek. Tím se ovšem zrychlí v procesoru vše, takže třeba příkaz Pause už nebude mít parametr v ms, ale polovinách ms, změní se i příkaz Pulsout, střední poloze serva bude odpovídat parametr 300 a krajům 200 a 400. Tento fígl se uplatní při programovém generování pulzů (Pulsout), příkaz Servo nemůže být použit při vyšší taktovací frekvenci než základní 4 MHz

SETFREQ

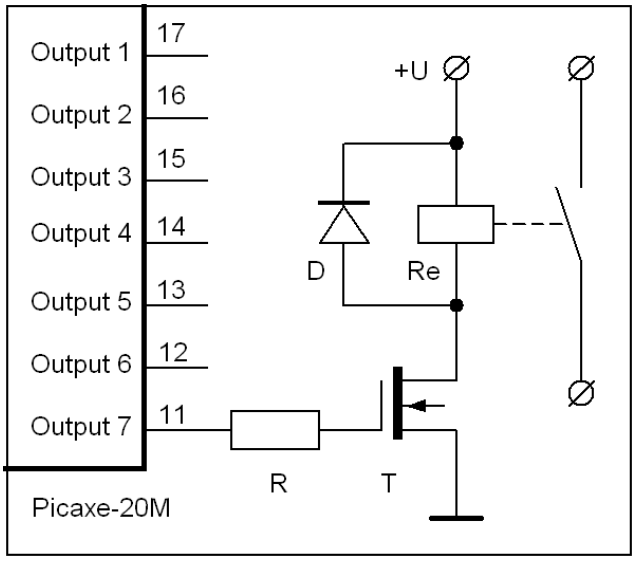
slouží k přepnutí hodinového kmitočtu procesoru, a to i za běhu programu. Parametr „m4“ udává frekvenci 4 MHz (základní) a běžně se nepoužívá, není-li frekvence měněna, parametr „m8“ znamená zrychlení na 8 MHz.

Pracuje-li procesor na 8 MHz, může při pokusu o přenos nového programu chvíli trvat, než se komunikace „chytne“ nebo může dokonce skončit po chvíli neúspěšně, v tom případě spustíme přenos programu a resetujeme procesor krátkým přerušením napájení. Funkci

příkazu můžeme nejjednodušeji vyzkoušet na programu „Blikající LED 2“ z 3. dílu našeho seriálu. Předřadíme-li mu příkaz „Setfreq m8“, bude LED blikat 2x rychleji.

Výkonové výstupy

Zatím jsme procesorem PICAXE 20M ovládali jen LED, na něž stačil proud poskytovaný přímo jeho výstupy. Je-li potřeba spínat větší proudovou zátěž (žárovku, relé, ventil, motor, …), musíme výstup posílit tranzistorem, a to nejlépe spínacím tranzistorem FET, jehož buzení je přizpůsobeno napěťovým úrovním logických obvodů. Nelze tedy použít jakýkoli FET, ty se standardním buzením (S) potřebují k plnému otevření napětí až 10V, ale prvky z buzením 5 V (třída L) nebo dokonce 3V (třída LL). Tranzistory přímo v sobě obsahují ochranné diody dimenzované podle přípustného proudu tranzistoru, takže jedinou další součástkou může být v případě zátěže indukčního charakteru (motory, cívky, elektromagnety, …) ochranná dioda přes zátěž. Rezistor v řídící elektrodě (mezi procesorem a tranzistorem) můžeme vynechat, pokud je zátěž spínána jen občasně, jestliže tranzistor slouží k vytváření pulzů plynulého řízení výkonu (PWM), je vhodné jej použít a jeho hodnota by měla být v řádu desítek ohmů, pro zkoušku vyhoví i 330Ω. Spínaný proud může podle výběru tranzistoru dosahovat až 30 A.

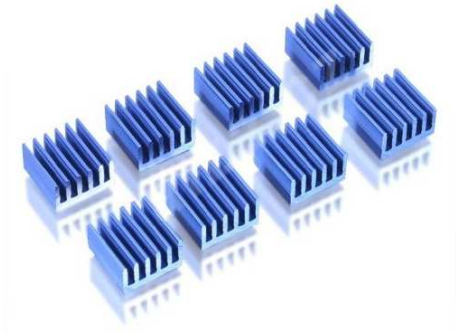


Pro řízení motorů (ale zdaleka nejen pro ně) je v anglické dokumentaci k procesorům PICAXE doporučován integrovaný obvod L293D. Jeho výkonnějším ekvivalentem je obvod SN754410, který je i v sadě součástek a budeme jej používat. Obvod obsahuje celkem čtyři výkonové členy, ty si můžeme představit jako zesilovače: mají jeden logický vstup od procesoru a jeden výkonový výstup, který může pracovat s jiným napájecím napětím (až +36V) než je napájení vlastního obvodu. Výstup se dá zatěžovat jak proti zemi, tak proti kladnému napájení a proud může být trvale až +/- 1,1 A, špičkově +/- 2A. Jedním obvodem tedy můžeme spínat například 4 žárovky nebo 4 relé.

Výkonové členy jsou sdruženy do dvou dvojic, každá z nich má svůj pomocný vstup, jímž se povoluje (úrovní H) činnost členů, pokud je na tomto vstupu značeném EN (Enable) úroveň L, jsou oba výkonové členy neaktivní, jejich výstup je ve stavu vysoké impedance (není spojen ani se zemí ani s kladným napájením). Dvě dvojice tedy umožňují obousměrně řídit dva stejnosměrné motory zapojené mezi jejich výstupy. Budou-li mít výstupy souhlasnou logickou hodnotu, motor stojí, při rozdílné se bude točit a směr otáčení závisí na tom, který z výstupů bude ve stavu H a který v L. Takto budeme obvod SN754410 převážně využívat. Další možností je řídit jedním obvodem jeden krokový motor se čtyřmi vinutími (bez změny směru proudu) nebo motor se dvěma vinutími (se změnou směru proudu).

I když je obvod ve standardním pouzdře DIL 16, je vlastně obvodem výkonovým a může na něm vznikat ztráta až 2W, takže je normální, pokud se při činnosti zahřívá. Měli bychom mu umožnit chlazení, jak je to jen možné. K chlazení slouží čtyři vývody (4,5,12,13), jenž jsou zároveň zemí a předpokládá se, že budou v desce s plošným spojem připájeny na rozsáhlejší

plochu mědi, ta teplo rozvede a vyzáří. Při našich pokusech na kontaktním poli toto chlazení odpadá, musíme obvod zatěžovat podstatně méně. Použité motory odebírají proud přibližně 5x menší, než je maximum obvodu. Lze také vytvořit dodatečné chlazení těla obvodu podobně, jako se to dělá s paměťmi v počítači, buď přímo chladiči určenými pro paměti, nebo například chladičem V5619A BLK (GM electronic) o rozměrech 6,3 x 19 x 5 mm. Trochu to pomůže.



Budeme předpokládat, že máme dvojkolový podvozek a k pohybu potřebujeme dva motory (jeden vpravo a jeden vlevo), jenž současně slouží k řízení směru pohybu. Zachováme-li v návodu uvedené přiřazení jednotlivých pinů a funkcí ovládání motoru, lze využít speciální příkazy Forward, Backward a Halt. Bohužel, ačkoli podle návodu i přehledu instrukcí v obslužném programu tyto příkazy jsou dostupné i pro náš procesor typu 20M, program je nepřijme, přesto si je uvedeme, pro PICAXE 20X se použít dají. Levý motor (A) musí být ovládán výstupními piny procesoru 4 a 5 a pravý motor (B) je řízen piny 6 a 7. Použijeme stejné přiřazení vývodů, ale budeme si muset vystačit s příkazy „high“ a „low“.

FORWARD

zapíná pohon motoru pro směr vpřed. Jeho parametr (A nebo B) určuje, ke kterému motoru se vztahuje. Tento příkaz je ekvivalentní sekvenci „high 4 low 5“ pro motor A nebo „high 6 low 7“ pro motor B.

BACKWARD

zapíná pohon motoru určeného parametrem A nebo B pro směr vzad. Tento příkaz je ekvivalentní sekvenci „low 4 high 5“ pro motor A nebo „low 6 high 7“ pro motor B.

HALT

vypíná chod motoru určeného parametrem A nebo B. Tento příkaz je ekvivalentní sekvenci „low 4 low 5“ pro motor A nebo „low 6 low 7“ pro motor B.

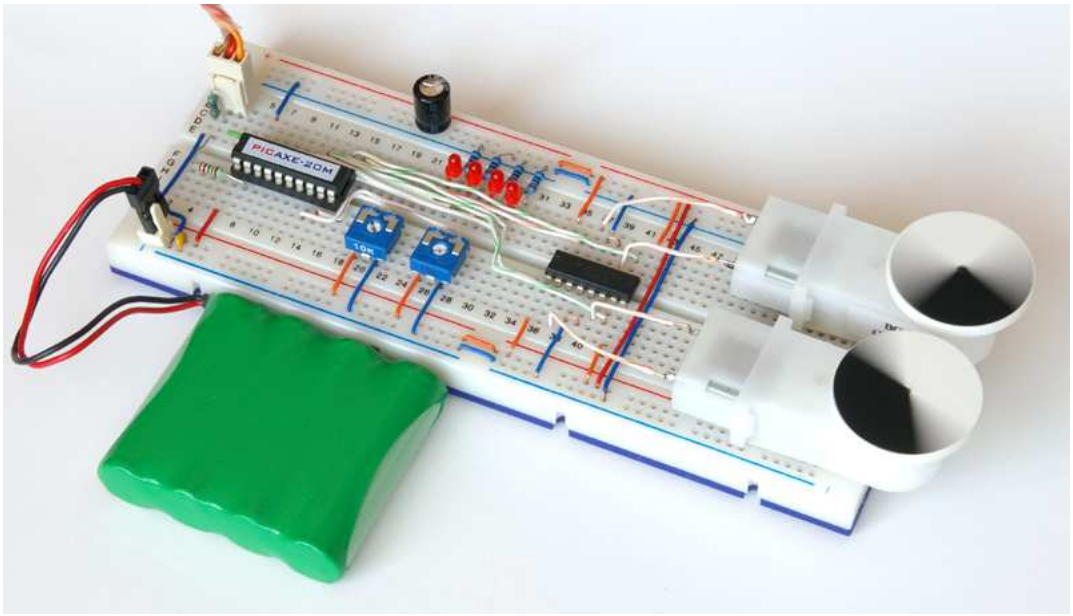
Ovládací program nejprve vyzkoušíme na dosavadním zapojení, uvidíme činnost na LED připojených na vývodech 4 až 7. Využijeme i trimry, ten na vstupu 1 bude ovládat motor A (levý), na vstupu 2 motor B (pravý). Při poloze kolem středu bude motor stát (obě LED ve dvojici zhasnuty), v krajích se rozjede jedním nebo druhým směrem (svítí právě jedna LED z dvojice).

```
1       REM Program pro PICAXE-20M: Motor1
2       REM Řízení motorů trimy (spínání)
```

```
3       start:
4         readadc 1,b0                ;načtení napětí z trimu A
5         readadc 2,b1                ;načtení napětí z trimu B
6         select b0                   ;obsluha motoru A
7           case <80 high 4 low 5     ;chod vpřed A
8           case >175 low 4 high 5    ;chod vzad A
9           else low 4 low 5          ;zastavit A
10          endselect                 ;konec obsluhy motoru A
11        select b1                   ;obsluha motoru B
12          case <80 high 6 low 7     ;chod vpřed B
13          case >175 low 6 high 7    ;chod vzad B
14          else low 6 low 7          ;zastavit B
15          endselect                 ;konec obsluhy motoru B
16        goto start
```

Radikálně upravíme obvod na kontaktním poli. Ponecháme LED na pinech 0 až 3, ponecháme oba trimy a základní součástky kolem procesoru PICAXE, ostatní odstraníme. Doplníme elektrolytický kondenzátor 470M kvůli vyhlazení napájení společného pro procesor i motory, zapojíme IO SN754410. K vývodům obou motorů musíme opatrně připájet vodiče. Lze i odizolovat konce v délce asi 15 mm a vodiče kolem vývodů omotat, ale velmi opatrně, protože vývody jsou měkké a nesmí se ulomit. Motory můžeme ke kontaktnímu poli přichytit oboustrannou lepicí páskou nebo kouskem kvalitní izolepy. Aby bylo zřetelně vidět otáčení, vystřihneme si z kartónu nebo měkkého plastu dvě kolečka o průměru asi 25 – 30 mm, nakreslíme na ně značky, do středu propíchneme otvor nepatrně menší než jsou výstupní osy a kotoučky na ně nasadíme. Stačí, když drží třením.

Při reálné konstrukci robota není vhodné napájet řídící obvody a pohony přímo z jednoho zdroje, hrozí nebezpečí výkyvů napětí při pohybu a rušení do napájení procesoru, ten se následně může občas nevysvětlitelně „zakousnout“. Je-li použit jeden akumulátor (sada baterií), měl by být na vyšší napětí (7 – 12 V) a procesor by měl mít vlastní stabilizátor napětí 5V. Přímé napájení všeho ze čtyř tužkových článků je použitelné jen pro pokusy nebo velmi malý odběr pohonu.



Vyzkoušíme program Motor1, oba „pohony“ by mělo jít jednotlivě ovládat pomocí trimů, jsou však jen spínané, běží plnou rychlostí, nebo stojí. Pokusíme se o plynulé řízení otáček, zatím v jednom směru. To je možné pulzy s proměnnou střídou při stabilním kmitočtu nebo pulzy se stabilní délkou při proměnném kmitočtu. V sortimentu příkazů PICAXE je dvojice povelů, jenž k tomu slouží, a to PWM a PWMOUT, ani jeden z nich ale není funkční s procesorem typu 20M, takže si budeme muset opět vystačit s příkazy High, Low a Pulsout. Program Motor2 reguluje výkon motoru A od nuly, využívá proměnné šířky pulzů a prakticky konstantní mezery mezi nimi, takže nedovoluje dosáhnout plné rychlosti.

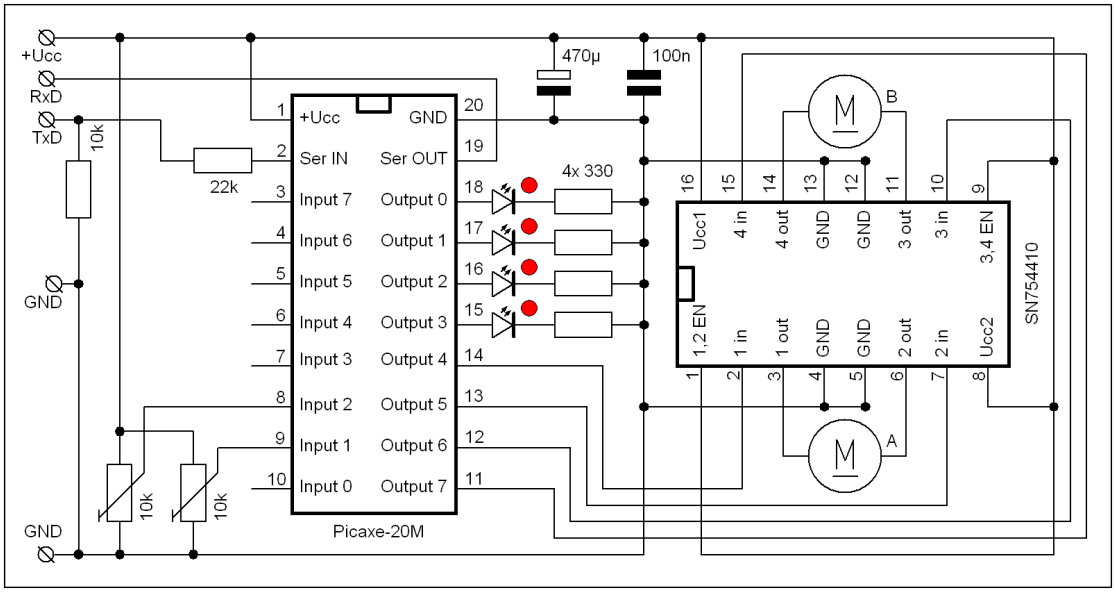
```
1       REM Program pro PICAXE-20M: Motor2
2       REM Jednosměrné proporcionální řízení motoru A
3       start:
4         readadc 1,b0                ;načtení napětí z trimu A
5         w1=b0*20                    ;úprava na delší pulzy
6         pulsout 4,w1                ;generování pulzu
7         pause 10                    ;zvětšení prodlevy mezi pulzy
8         goto start
```

Věnujme pozornost koeficientu (zde 20), jímž se upravuje šířka pulzů. Zvolíme-li jej příliš malý (1), bude velmi jemná regulace v oblasti nejnižších otáček a rotor motoru, který je vidět přes plastový kryt, se bude jen líne převalovat, nejvyšší nastavitelný výkon bude ale jen zlomkem plného. Zvolíme-li koeficient příliš vysoký (100), bude regulace fungovat téměř do maxima, ale nastavit pomalé otáčky nepůjde. Vyzkoušíme různé hodnoty a jejich důsledky, optimum leží někde mezi 10 a 20. Druhým parametrem na vyzkoušení je délka prodlevy, příliš malá vede ke ztrátě možnosti nastavení malých rychlostí, příliš dlouhá k trhavému pohybu. Je vidět, že výsledek ovlivňují oba parametry a je třeba hledat vhodné hodnoty v součinnosti.

```
1       REM Program pro PICAXE-20M: Motor3
2       REM Obousměrné proporcionální řízení motoru A
3       start:
4         readadc 1,b0
5         if b0>130 then let w1=b0-128*40 pulsout 4,w1 endif
6         if b0<126 then let w1=128-b0*40 pulsout 5,w1 endif
7         pause 20
8         goto start
```

Program Motor3 pracuje v podstatě stejně, jen dělí rozsah vstupní hodnoty získané z trimru na dvě části a podle výsledku volí směr pohybu posíláním pulzů na piny 4 nebo 5. Malá oblast mezi 126 a 130 zajišťuje, že uprostřed dráhy snadno nalezneme polohu, v níž bude motor zcela v klidu.

Úvodem sedmého pokračování našeho seriálu je třeba uvést schéma zapojení určeného k vyzkoušení práce s pohony, v každém případě doporučuji vyzkoušet si na něm nejprve jednodušší programy Motor1 až Motor3 z minulého dílu.



Obsluha obou motorů současně, pokud mají být řízeny proporcionálně, je podstatně složitější. Program Motor4 víceméně jen zdvojuje předchozí příklad, jenže v tom je ta potíž. Procesor může vykonávat jen jeden příkaz „pulsout“ současně a tak jak se zvětšují pulzy pro jeden motor, roste vlastně i mezera mezi pulzy pro druhý motor a obráceně. Motory se vzájemně silně ovlivňují, což by v praxi vedlo k tomu, že s každou změnou výkonu by začal robot někam zatáčet, to asi nebude dobré. Vyzkoušíme si, jak se takový program projevuje, ale už začíná být jasné, že tudy cesta nevede a je třeba navrhnout algoritmus jinak, aby se oba motory daly řídit nezávisle na sobě, což příkazem pulsout nepůjde.

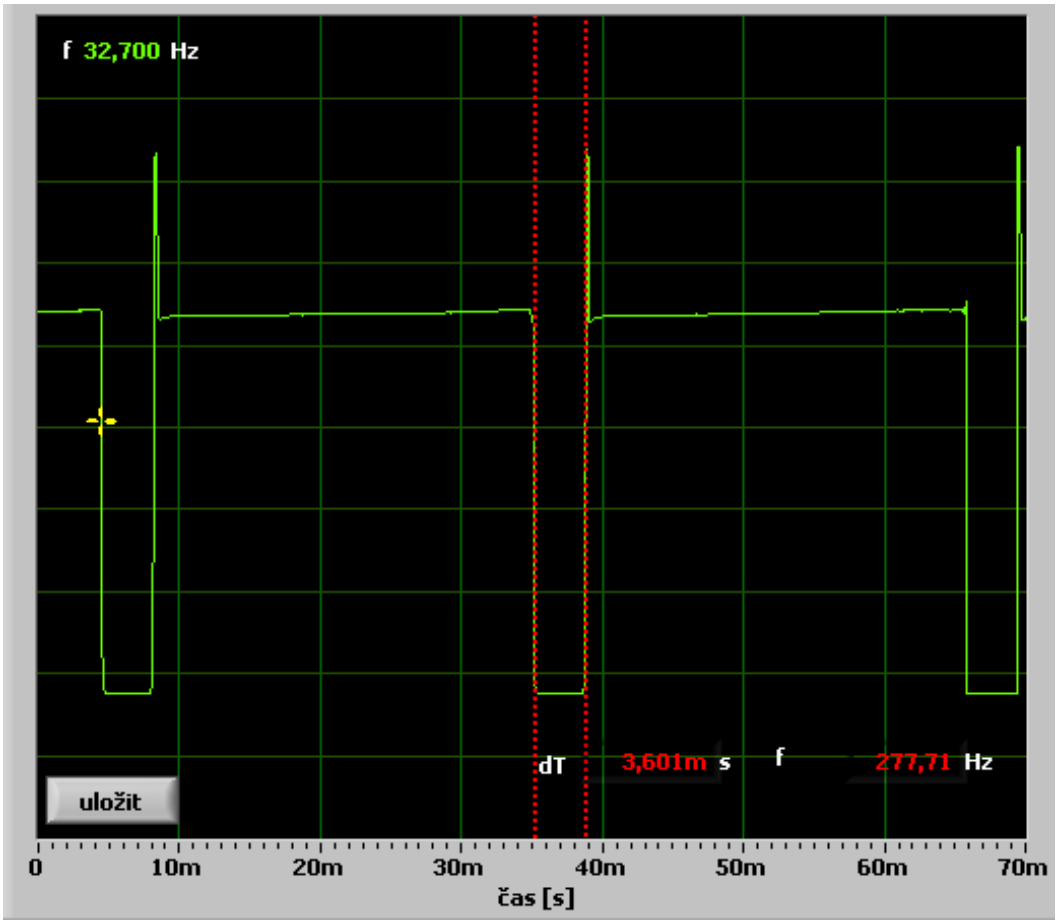
```
1       REM Program pro PICAXE-20M: Motor4
2       REM Obousměrné proporcionální řízení obou motorů
3       start:
4         readadc 1,b0                          ;načtení napětí z trimu A
5         readadc 2,b1                          ;načtení napětí z trimu B
6         if b0>130 then
7           let w1=b0-128*40 pulsout 4,w1 endif ;motor A vpřed
8         if b0<126 then
9           let w1=128-b0*40 pulsout 5,w1 endif ;motor A vzad
10        if b1>130 then
11          let w2=b1-128*40 pulsout 6,w2 endif ;motor B vpřed
12        if b1<126 then
13          let w2=128-b1*40 pulsout 7,w2 endif ;motor B vzad
14        pause 10
15        goto start
```

Pokusíme se o jinou taktiku. Pulzy pro oba motory budeme dělat vlastním programem (příkazy High, Low) a pokusíme se zajistit, aby procesor bez ohledu na režim obou motorů vykonával stejné (nebo ekvivalentní) příkazy, prostě aby průběh programu trval vždy stejně dlouho. Začneme s jednosměrným pohybem, ten je jednodušší. Tento způsob generování pulzů je ovšem podstatně náročnější na výkon procesoru, takže musíme někde slevit,

v daném případě omezíme počet rychlostních stupňů na 16. Tyto stupně jsou za chodu při pomalém pohybu trimrů znatelné, ale v praxi se jeví téměř stejně jako předchozí proporcionální řízení, které je vlastně také stupňovité, jen používá více rychlostních kroků. Všimneme si proměnné b3 a její funkce. Zdánlivě nedělá vůbec nic, nikde se nepoužívá, jen se od ní občas odečte jednička. To je právě ta kompenzace doby chodu programu respektive příkazů „dec b0“ a „dec b1“. Vyzkoušíme, jak program pracuje, pokud je znát nějaké ovlivnění rychlosti jednoho motoru druhým, doporučuji použít kvalitnější zdroj, proti snížení napětí při větším zatížení zapojení samozřejmě odolné není.

```
1       REM Program pro PICAXE-20M: Motor5
2       REM Jednosměrné proporcionální řízení obou motorů
3       setfreq m8                      ;nastavení hodin 8 MHz
4       start:                          ;hlavní smyčka PWM pulzů
5         readadc 1,b0                  ;načtení napětí z trimu A
6         readadc 2,b1                  ;načtení napětí z trimu B
7         b0=b0/16 b1=b1/16             ;16 stupňů rychlosti A,B
8         for b4=16 to 1 step -1        ;tvorba jednoho pulzu
9           if b0>0 then high 4 dec b0
10            else low 4 dec b3 endif   ;pulz pro motor A
11          if b1>0 then high 6 dec b1
12            else low 6 dec b3 endif   ;pulz pro motor B
13          next b4                     ;konec generování pulzu
14        goto start                    ;další načtení a pulz
```

S předchozími zkušenostmi už není obtížné sestavit program pro obousměrné řízení. Ponecháme 16 stupňů rychlosti, ale polovinu jich přidělíme jednomu směru pohybu, polovinu druhému. Při maximální rychlosti pulzy přechází v trvalé sepnutí. Stejně jako v předchozím případě zvýšíme rychlost procesoru přepnutím na hodinovou frekvenci 8 MHz, i tak kmitočet PWM pulzů řízení motorů nepřesahuje 33 Hz. Stupně rychlosti jsou dobře znát, opravdu jasnou představu o skutečné činnosti procesoru nám udělá ale až prohlídka signálů osciloskopem (viz článek na jiném místě tohoto čísla časopisu). Snímek z programu Soundcard Scope zachycuje chod na čtvrtinu výkonu snímaný přímo na jednom z vývodů motoru, změny řízení se v pravidelných intervalech projevují skokovou změnou střídy pulzů a lze ověřit, že v plném chodu pulzy zanikají.

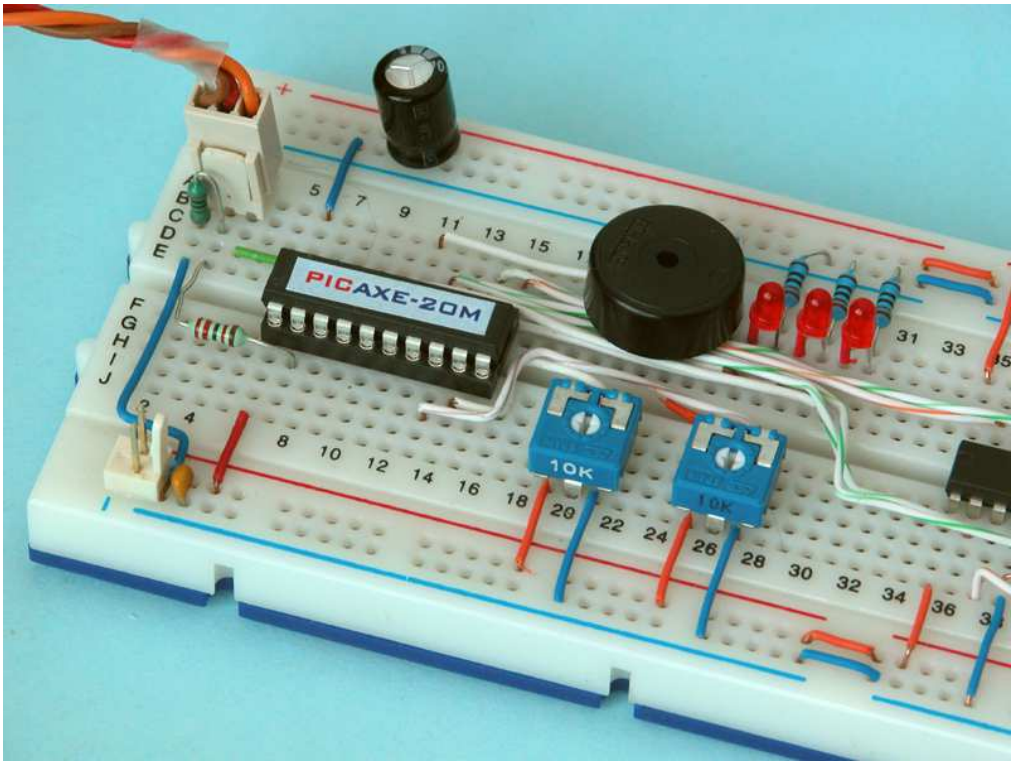



```
1       REM Program pro PICAXE-20M: Motor6
2       REM Obousměrné proporcionální řízení obou motorů
3       setfreq m8                          ;nastavení hodin 8 MHz
4       start:                              ;hlavní smyčka PWM pulzu
5         readadc 1,b0                      ;načtení napětí z trimu A
6         readadc 2,b1                      ;načtení napětí z trimu B
7         b0=b0-127/16+100                  ;8+8 stupňů rychlosti A
8         b1=b1-127/16+100                  ;8+8 stupňů rychlosti B
9         for b4=8 to 1 step -1             ;tvorba jednoho pulzu
10          if b0>100 then high 4 dec b0
11            else low 4 dec b3 endif       ;motor A směr >
12          if b0<100 then high 5 inc b0
13            else low 5 inc b3 endif       ;motor A směr <
14          if b1>100 then high 6 dec b1
15            else low 6 dec b3 endif       ;motor B směr >
16          if b1<100 then high 7 inc b1
17            else low 7 inc b3 endif       ;motor B směr <
18          next b4                         ;konec generování pulzu
19        goto start                        ;další načtení a pulz
```

Generování zvuku

I když vytváření zvuku nebude asi pro procesor aplikovaný v robotovi tou nejdůležitější činností, nějaké to pípnutí nebo efekt oznamující chybu se určitě hodí. Předem se budeme muset rozloučit s představou, že by procesor zvládal třeba řídit plynule motory, „rozhlížet“ se po zdroji světla a k tomu průběžně vytvářet perfektní zvukovou kulisu. Každou z těchto činností zvládne samostatně, současně ne, na to jeho výkon nestačí. Budeme si muset zvolit, buď bude uvedené činnosti dělat postupně (střídavě), nebo potřebujeme více procesorů, aby se každý postaral o to svoje, a zařídit komunikaci mezi nimi.

Upravíme zapojení. Odpojíme LED na výstupu 3 i její rezistor a mezi tento výstup a zem zapojíme piezoměnič, žádná další součástka není třeba. Piezoměnič ze sady má tak malou spotřebu, že může být přímo buzen procesorem. Šlo by použít také malý reproduktor s vyšším odporem (třeba ze sluchátek), ale ten by se musel oddělit kondenzátorem 10M,



příklad zapojení je v aplikační příručce na straně 10. Je-li třeba silnější zvuk, je nutné výstup procesoru posílit tranzistorem případně můžeme využít i jeden z kanálů obvodu SN754410, pokud by nebyly využity všechny.

Zvuk není nic jiného než generování pulzů, frekvencí. Zkusíme v předchozím programu těsně před FOR cyklus vsunout dva příkazy: toggle 3 toggle 2. Ty způsobí, že se během činnosti motorů bude ozývat vrčení a současně blikat LED v polovičním rytmu, než jsou generovány pulzy PWM. Přemístíme-li oba příkazy za příkaz FOR, tón se zvýší. Lidské ucho je velmi citlivé na změnu tónu a snadno poznáme, že při ovládání motorů se přece jen tón nepatrně mění, takže program neběží vždy přesně stejně, ale jinak než sluchem to těžko poznáme. Vrčení je asi tak všechno, co procesor zvládne, aniž by se narušilo ovládání motorů. Přidáme další příkazy:

SOUND

vytváří sérii pípnutí, má proměnlivý počet parametrů. Prvním je výstup, na nějž má výstup směřovat, pak následují v závorce dvojice bytů (nejméně jedna) s udáním výšky tónu a délky tónu v desítkách ms. Parametr výšky tónu 1 – 127 vytváří čistý zvuk, hodnoty 128 – 255 generují šum, 0 znamená pauzu. Tento příkaz se používá ze zvukové signalizaci nebo jako součást efektů, není vhodný pro reprodukci hudby, i když jednoduchá melodie se jím zahrát dá. Po dobu generování zvuku se procesor nemůže věnovat ničemu jinému. (*27)

TUNE

slouží ke generování jednoduchých melodií v rozsahu tří oktáv. První parametr udává, na který výstup se zvuk bude směřovat, druhý parametr řídí tempo přehrávání skladby a následuje skladba v datech. K jednoduššímu pořízení dat pro tento příkaz slouží samostatný program Tune Wizard. Příručka programátora (*28) uvádí příkaz Tune odlišně a to ve verzi, jak pracuje s procesorem PICAXE 08M.

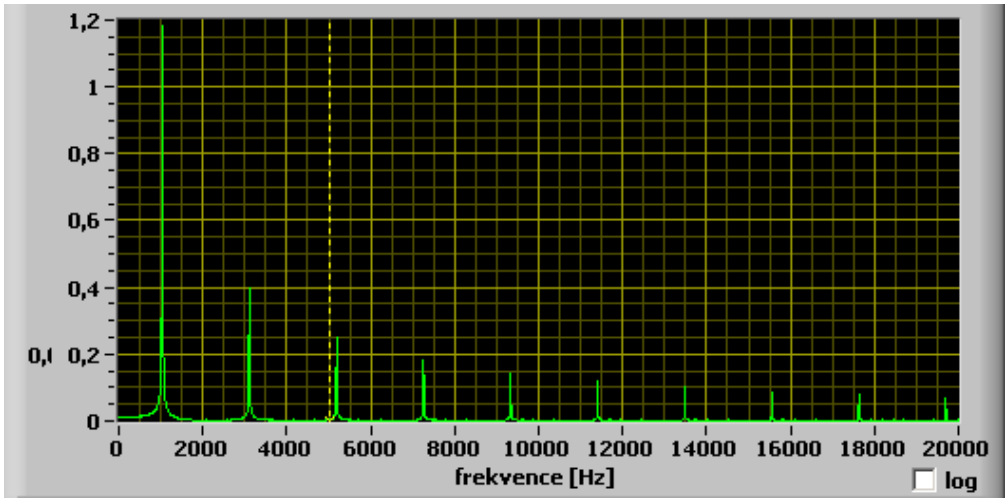
Poznámka: Bílý šum je signál s rovnoměrnou výkonovou spektrální hustotou, to znamená, že signál má stejný (podobný) výkon v jakémkoli pásmu frekvencí shodné šířky. Bílý šum se nazývá podle analogie s bílým světlem, jež také obsahuje všechny frekvence (barvy). Dokonalý bílý šum neexistuje, v praxi se mu blíží signál, který má ploché frekvenční spektrum. Při generování zvuku se používá jako složka různých šelestů a šustění, ruchů z dopravy a podobně. Samostatně má zajímavou vlastnost, nezpůsobuje ozvěnu.

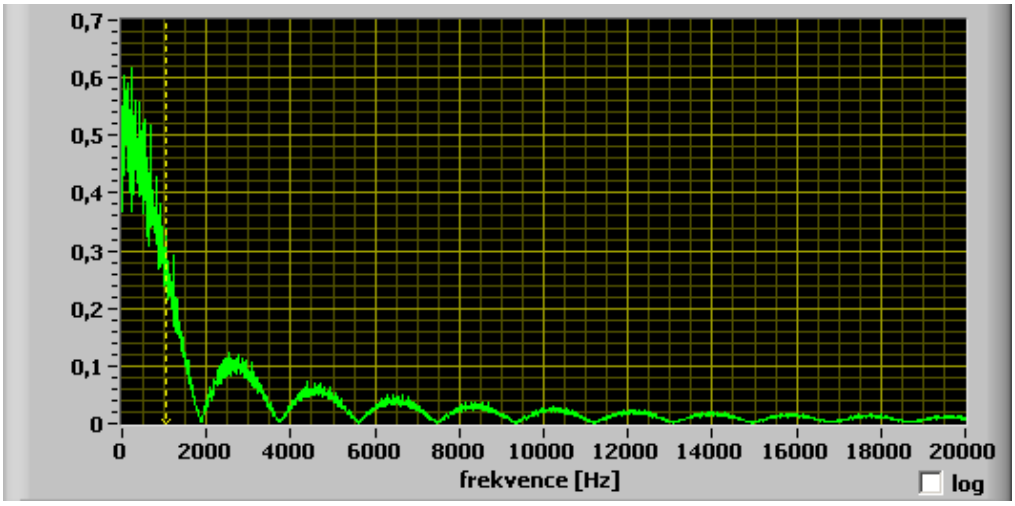
První program dělá v pravidelných intervalech jednoduchý signál ze čtyř tónů, výstup posílá na Output 3. Parametry v závorce jsou kvůli názornosti mezerami rozdělené do dvojic, podobný způsob zápisu se vyplatí dodržovat i když samozřejmě není nutný, nejčastější chybou bývá vynechání nebo zdvojení parametru, které se v dlouhé řadě špatně hledá. Vyzkoušíme si různě měnit výšku i délku tónu.

```
1       REM Program pro PICAXE-20M: Zvuk1
2       REM Generování tónů
3       start: sound 3,(50,10, 100,10, 50,10, 100,40)
4         pause 4000
5         goto start
```

Druhý program je téměř totožný, jen všechny parametry výšky tónu zvýšíme o 128 (tedy do režimu šumů) a aby bylo vůbec něco slyšet, prodloužíme doby 10x. Ozve se skřípání s škvrčení, v němž se dá náznak melodie rozpoznat. Generovaný šum není bílý, pro porovnání si ukážeme spektrum „čistého“ obdélníkového tónu blízkého frekvenci 1 kHz (parametr 119), v němž je vidět řada lichých harmonických složek, a odpovídající šum (parametr 247) s mnohem větším rozprostřením energie. Případným zvýšením hodinového kmitočtu na 8 MHz se všechny tóny zvýši o oktávu a zkrátí na polovinu.

```
1       REM Program pro PICAXE-20M: Zvuk2
2       REM Generování šumů
3       start: sound 3,(178,100, 228,100, 178,100, 228,400)
4         pause 4000
5         goto start
```





Program Zvuk3 předvádí postupně zrychlovaný klouzavý tón generovaný For cyklem. Dá se vymyslet nepřeberné množství různě znějících sirén a efektů, kvůli úspoře místa v programu ale zpravidla zůstáváme u jednoduchých signálů z několika málo tóny. Příkaz Sound se nemusí použít jen ke generování zvuku, ale obecně obdélníkových pulzů se střídou 1:1, které se hodí třeba i k buzení LED do nižšího jasu.

```
1       REM Program pro PICAXE-20M: Zvuk3
2       REM Ukázka "klouzavého" zvukového efektu
3       setfreq m8
4       start:
5         for b1 = 4 to 1 step -1
6           for b0 = 50 to 120 sound 3,(b0,b1) next b0
7           for b0 = 120 to 50 step -1 sound 3,(b0,b1) next b0
8         next b1
9         pause 8000
10        goto start
```

Psát melodie přímo pomocí příkazu Tune je poměrně pracné, originální manuál se tomu věnuje na straně 207 - 209, ale v editoru je výkonná pomůcka, program Tune Wizard (Menu

– PICAXE – Wizards – Ring Tone Tunes). Jednohlasá melodie se do něj zadává v tónech (délka, nota, oktáva, výstup), lze ji uložit, načíst, přehrát na ukázku v PC a také exportovat do zvukového WAV souboru nebo rovnou vygenerovat příslušný příkaz na pozici kurzoru do zápisu programu. Tune Wizard práci usnadní, chceme-li zapsat konkrétní melodii, jde-li spíše o to si s programem „pohrát“ nebo použít nějakou hudbu (lze-li tomu ovšem hudba vůbec říkat), jde najít na internetu stovky hotových skladeb, některé i v přijatelné kvalitě. Ze stránek Robot Revue si spolu s programy použitými v tomto díle můžete stáhnout balíček s téměř tisícem skladeb, které jsou vlastně samostatnými programy. Doporučuji zkusit třeba soubor StarWars. Před naprogramováním do procesoru je nutné změnit směrování výstupu (všechny skladby posílají zvuk na Output 0).

Nastavení a obsluha přerušení

Co je přerušení a kdy se efektivně použije? V mnoha případech program něco dělá, ale je potřeba, aby současně také sledoval vstupy (tlačítka, taktilní čidla, odrazové infračidlo hlídající náraz do překážky, …) a při změně neprodleně zareagoval. Můžeme toho docílit pravidelným testováním stavu vstupů v programu, ale aby se zaregistroval i krátký impulz (nebo dlouhý včas), musí se vstupy testovat velmi často, což narušuje plynulý běh programu. V podobných situacích se používá interrupt, což je automatické přerušení běhu programu a odskok do připraveného podprogramu při zadané kombinaci stavů na vstupech. Je to velmi silný a často používaný nástroj, který je třeba se naučit používat. Budeme potřebovat nový příkaz.

SETINT

má dva parametry zadávané zpravidla pro názornost ve dvojkové soustavě (není podmínkou), prvním se nastavuje, jaký má být stav na vstupech, aby došlo k aktivaci interruptu, druhým se označí ty vstupy, které se berou v úvahu. Příkaz je popsán v české příručce jen stručně a s odkazem na originální anglický manuál (strana 64 a dál), proto budeme muset postupovat podrobněji a s příklady.

Setint %00000000,%00000001

Tento příkaz říká, že k přerušení dojde, když na vstupu vstupu 0 (jednička v druhém čísle, počítáme zprava podle vzoru 76543210) bude logická nula (nula na stejné pozici v prvním parametru). Obdobně

Setint %00001000,%00001000

znamená, že přerušení vyvolá vstup 3 (poloha jedničky v druhém parametru) bude-li na něm logická jednička (jednička na odpovídající pozici prvního parametru). Stavy můžeme kombinovat, vždy musí platit všechny zadané podmínky současně, takže:

Setint %00001010,%00001111

očekává na vstupu 0 hodnotu 0, současně na vstupu 1 hodnotu 1, současně na vstupu 2 hodnotu 0 a na vstupu 3 hodnotu 1. Bude-li vše splněno, odskočí program na návěští s pevně

daným pojmenováním „interrupt“.

Interrupt je podprogram jako každý jiný, musí být tedy ukončen návratem (return) a na jeho návěští se lze programově odkázat podle potřeby (příkazy gosub, goto). Je tu ale jedna zásadní odlišnost, na rozdíl od podprogramu, jenž voláme vždy ze zadaného místa a tedy víme (respektive máme vědět), v jakém stavu jsou proměnné a k čemu která zrovna slouží, interrupt může být zavolán vnějším podnětem z kteréhokoli místa programu v době, kdy program přechází z jednoho příkazu na druhý (typicky mezi řádky) nebo dokonce v průběhu provádění příkazu, to se týká třeba příkazu Pause. Po skončení výkonu interruptu se program vrátí na místo, odkud byl odvolán, a pokračuje v činnosti.

Nastavení příkazem Setint platí pro jedno jediné volání interruptu, má-li být používán opakovaně, musíme jej vždy znovu „nahodit“ opětovným použitím Setint, třeba rovnou v závěru obslužného podprogramu interruptu. Lepší je používat Setint na jiném místě programu a pokud možno se vyhnout možnosti, že výkon podprogramu interrupt bude přerušen jeho dalším voláním. Pro procesor PICAXE 20M platí omezení, že interrupt lze detekovat jen na vstupech 1 – 5, na ostatních je nefunkční, takže při už konstrukci pokud možno na tyto vstupy připojujeme čidla, u nichž budeme požadovat rychlou reakci.

Používají se dva triky. Jde-li o to, aby procesor zaregistroval i krátký pulz, ale následná reakce nemusí přijít hned, interrupt jen uloží předem danou hodnotu do proměnné (nastaví příznak toho, že k události došlo) a okamžitě se vrátí. Program občas (když může) otestuje proměnnou (běžným IF) a podle výsledku provede akci. Interrupt můžeme využít i jinak. Potřebujeme, aby procesor něco provedl vždy v přesnou dobu. Na příslušný vstup připojíme obvod řízený krystalem, který posílá v pravidelných intervalech impulzy. Důležitá část programu bude obsluhou tohoto interruptu, jinak se může procesor věnovat jiné činnosti, ovšem bez dlouhých příkazů.

Do zapojení přidáme rezistor (třeba 330 ohmů) mezi vstup 3 a +5V, stejný vstup spojíme přes tlačítko se zemí. K vyzkoušení interruptu použijeme upravený program se zvukovým efektem, po stisku tlačítka má LED změnit svůj stav a hřídel jednoho z motorů se pootočí přibližně o půl otáčky. Budeme sledovat především to, jak obsluha motoru naruší zvuk. LED reaguje ihned, zvuk se přeruší a pokračuje se v něm po skončení obsluhy motoru. Pokud tlačítko přidržíme, budou se motor a zvuk střídat, protože k přerušení dojde vždy okamžitě po „nahození“ interruptu před začátkem zvukového cyklu.

```
1       REM Program pro PICAXE-20M: Přerušení1
2       start:
3         setint %00000000,%00001000              ;nastavit interrupt
4         for b0 = 80 to 100
5         sound 3,(b0,3) next b0                  ;generování zvuku
6         goto start
7       ;
8       interrupt:
9         toggle 0                                ;změna LED
10        high 4                                  ;rozjet motor
11        pause 300                               ;počkat 1/2 otáčky
12        low 4                                   ;zastavit motor
13        return                                  ;návrat z podprogramu
```

Konec programu a spánek procesoru

Zatím jsme vždy program končili skokem na jeho začátek, pracoval tedy v nekonečné smyčce. Má-li se program provést jen jednou nebo v určité situaci zastavit, musíme k tomu použít příslušný povel. Bylo by samozřejmě možné program zacyklit v malé nekonečně a nic nedělající smyčce, je tu ale určitý rozdíl ve spotřebě procesoru. Druhým případem je, že potřebujeme procesor na omezenou předem danou dobu „uspat“, tedy převést do stavu s minimální spotřebou, aby se šetřila energie.

END

Ukončí běh programu, procesor přejde do stavu s minimální spotřebou. Obnova činnosti je možná jen vypnutím a zapnutím napájení nebo zapsáním nového programu z PC. (*7)

STOP

Podobně jako End ukončí běh programu, procesor však nepřejde do stavu s minimální spotřebou, takže pokud to konkrétní typ procesoru PICAXE dovoluje, pokračuje se ve výkonu příkazů, které běží „na pozadí“, respektive využívají služeb interního časovače. Obnova činnosti je možná jen vypnutím a zapnutím napájení nebo zapsáním nového programu z PC. (*27) Spotřeba procesoru je obdobná jako při běhu v nekonečné smyčce.

NAP

má jeden parametr v rozsahu 0 až 7, jímž se určuje doba přechodu do režimu s nízkou spotřebou (v němž procesor nevykonává program). Jednotkou je 18 ms, výsledná doba se spočte jako 2^hodnota parametru * 18 ms. „Nap 1“ uspí procesor na 32 ms, „Nap 2“ na 72 ms atd. Nejdelší spánek, který lze takto vyvolat, trvá 2,304 s. V průběhu uspání se nevyhodnocuje ani interrupt.

SLEEP

slouží stejným způsobem jako Nap k uspání procesoru, ovšem na delší dobu. Má jeden parametr v rozsahu 0 až 65535, jednotkou doby je 2,3 s. „Sleep 2“ uspí procesor na 4,6 s, maximální hodnota parametru na téměř 42 hodin. V průběhu spánku procesor nereaguje ani na pokus o zavedení nového programu.

Všechny uvedené povely můžeme vyzkoušet na krátkém programu, který jen pípne aby bylo vidět že něco dělá, a pak jej ukončíme nebo uspíme. Program však musíme zakončit skokem na začátek, aby se v případě uspání na omezenou dobu projevilo, že pokračuje v činnosti.

Ukládání dat do EEPROM

Všechny programy, které jsme dosud zkoušeli, vždy při zapnutí napájení začínaly znovu za stejných „startovních podmínek“. Cokoli, co program uložil do proměnných, bylo vypnutím napájení definitivně zapomenuto. Často se hodí, aby si program mohl poznamenat hodnotu nějakého parametru a využít ji i příště. Je to možné. Procesor PICAXE 20M má k dispozici

256 bytů paměti EEPROM , do níž se program zapisuje „zdola“ a současně můžeme „shora“ ukládat data. Musíme samozřejmě zajistit, aby data nepřepsala program, ten by přestal fungovat. I k tomu se hodí údaj o délce programu, který se vypisuje při každém přenosu programu do procesoru.

WRITE

má dva parametry, první je adresa (0 - 255) místa, kam se má byte uložit, přičemž adresa „0“ je úplně „nahoře“ v paměti, nejvzdálenější od začátku programu. Druhý parametr tvoří data, která se mají na danou pozici uložit. Pracuje se s jednotlivým byty a pokud je potřeba uložit takto proměnnou typu word, uloží se dvěma příkazy jako odpovídající proměnné typu byte, například w0 uložíme jako b0 a b1(na různé adresy).

READ

je podobný příkaz, pracuje obráceně. První parametr určí adresu z níž budeme číst, druhým je proměnná, do níž se přečtená hodnota uloží.

Počet zápisů do paměti EEPROM je sice velmi vysoký, ale přece jen omezený, takže tuto možnost budeme využívat z hlediska programu zřídka. Poznamenáme-li si třeba vždy zapnutí zařízení, pak při 100000 přepisech EEPROM a použití 10x denně dospějeme k deklarované (minimální) životnosti paměti za 27 roků. To je přijatelný výsledek. Budeme-li ovšem bezhlavě zapisovat do EEPROM v cyklu, můžeme stejného limitu dosáhnout během hodin, to asi nebude rozumné. Počet čtení není rozhodující..

Program Eeprom1 plní 101 bytů paměti po sobě jdoucími čísly, po každém krátce pípne a nakonec vydá hlubší tón a zastaví se. Běží-li tento program v procesoru, nemáme prakticky možnost zkontrolovat jeho činnost, můžeme se ale na něj podívat v režimu simulace. Otevře se jednak okno s proměnnými, v němž můžeme sledovat postupný nárůst hodnoty proměnné b0, jednak nové okno ukazující obsah paměti, v němž je vidět, jak hodnoty postupně paměť plní. Neotevře-li se okno s obsahem paměti, musíme jej povolit (Menu – Simulate – Simulation Panels – Standard - Memory). Pokud bychom prodloužili cykly a data se „potkala“ s programem, v simulaci se běh zastaví a oznámí chyba, při reálném běhu v procesoru ale dojde k přepsání programu, který se tím definitivně poškodí.

```
1       REM Program pro PICAXE-20M: EEPROM1
2       REM Ukázka plnění paměti v simulaci
3       for b0=0 to 100             ;cyklus 101 čísel
4       write b0,b0                 ;zapsat číslo
5       sound 3,(110,2)             ;pípnout
6       next b0
7       sound 3,(55,100)            ;zahoukání nakonec
8       end                         ;zastavit program
```

Druhý program, jímž si ověříme činnost zápisu do EEPROM, bude mít velmi jednoduchou funkci, vždy po spuštění pípne tolikrát, pokolikáté byl zapnut. Program spouštíme vypnutím a opětovným zapnutím napájení. Pauza musí být delší, aby se vybil i elektrolytický kondenzátor (nebo jej dočasně vyjmeme).

```
1       REM Program pro PICAXE-20M: EEPROM2
2       REM Počítání spuštění programu
3       read 0,b0                       ;čtení z adresy 0
4       for b1=0 to b0                  ;pípnutí podle dat
5         sound 3,(120,10)
6         pause 200
7         next b1
8       inc b0                          ;zvýšení hodnoty o 1
9       write 0,b0                      ;zápis na adresu 0
10      end                             ;konec programu
```

Náhodný generátor

Někdy se hodí zavést do programu prvek náhody respektive pseudonáhody, třeba když program může volit z několika rovnocenných možností. V takovém případě náhodná volba zvyšuje dojem, že zařízení se chová jako živý tvor, ne přesně naprogramovaný stroj.

RANDOM

má jeden parametr, proměnnou typu word, do níž se ukládá pseudonáhodné číslo v rozsahu 0 – 65535. Toto číslo současně slouží jako výchozí pro výpočet následujícího pseudonáhodného čísla, takže obsah proměnné nesmíme programem měnit. (*20)

Po zapnutí procesoru mají všechny proměnné hodnotu 0, pseudonáhodná sekvence čísel se tedy opakuje vždy stejně. Vadí-li to, nastavíme počáteční hodnotu proměnné podle nějaké skutečnosti, kterou nemůžeme přesně ovlivnit, třeba podle doby stisku tlačítka změřené na jednotky milisekund, pak bude pseudonáhodná sekvence jedinečná. Naopak, chceme-li, aby se „náhodná“ sekvence opakovala i v průběhu činnosti procesoru, nastavíme proměnnou na předem zvolenou hodnotu. Program Random generuje náhodné zvuky. Můžeme si na něm vyzkoušet, že opravdu je sekvence po zapnutí stejná.

```
1       REM Program pro PICAXE-20M: Random
2       REM Ukázka generátoru náhodných čísel
3       start:
4         random w0                     ;generování náhodného čísla
5         b2=b0/2                       ;spodní byt vydělit dvěma
6         sound 3,(b2,10)               ;výška tónu podle čísla
7         goto start
```

Závěr

Jsme u konce základního seznámení s procesorem PICAXE 20M a jeho možnostmi. Neprobrali jsme všechny dostupné příkazy, jen ty, u nichž se dá očekávat, že budou používány nejčastěji. K podrobnému seznámení s příkazy je určena příručka programátora a aplikační dokumentace, případně nápověda v obslužném programu. Na toto seznámení volně navážeme konstrukcemi jednoduchých robotů, v nichž se využijí jak probrané příkazy, tak součástky zkušební sady.

Obsah