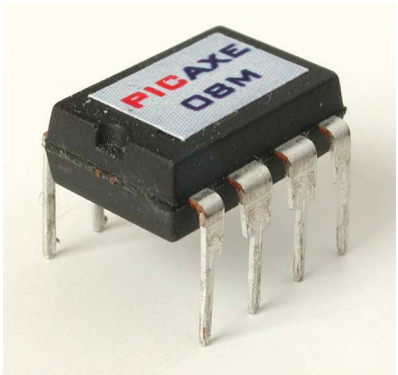
– procesor pro neprogramátory

Následující text byl poprvé zveřejněn v modelářském časopisu RC Revue (www.rcrevue.cz) v roce 2009 jako seriál článků. Protože připoutal pozornost značného množství zájemců o jednoduché použití mikrokontrolérů i mimo skupinu aktivních modelářů a čtenářů uvedeného časopisu, dávám jej nyní k dispozici v jiné grafické podobě, nicméně s nezměněným, tedy také neaktualizovaným obsahem. Je pravda, že v průběhu uplynulých let přibyly do sortimentu mikrokontrolérů PICAXE nové (a rychlejší) typy a také nové příkazy, zde uvedené příklady volené pochopitelně z oblasti RC modelářství by však měly být použitelné beze změny i dnes, přestože nemusí být optimálním řešením dané úlohy. O to ale vůbec nejde, cílem bylo názorně představit úplným začátečníkům možnosti mikrokontrolérů PICAXE na příkladu typu 08M a provést zájemce úvodem při psaní vlastních programů a aplikací těchto mikrokontrolérů.

Ing. Michal Černý

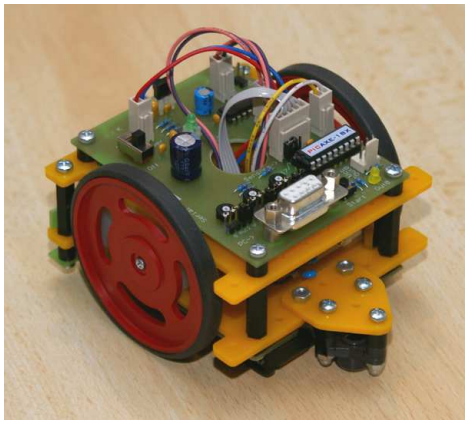
Začínáme

Slovo „procesorový“ se u modelářských zařízení víceméně stalo synonymem pro výraz nastavitelný. Přestože nejde o programování v plném slova smyslu, ale ve své podstatě právě jen o nastavení výrobcem připravených a omezených parametrů, „programujeme“ své vysílače, regulátory, a v poslední době i jednotlivá serva. Programování procesorů při vývoji a výrobě zařízení přenecháváme odborníkům jednoduše proto, že mu nerozumíme, a ono opravdu není jednoduché naučit se programovat na potřebné úrovni. Nemusí to ale platit vždy.

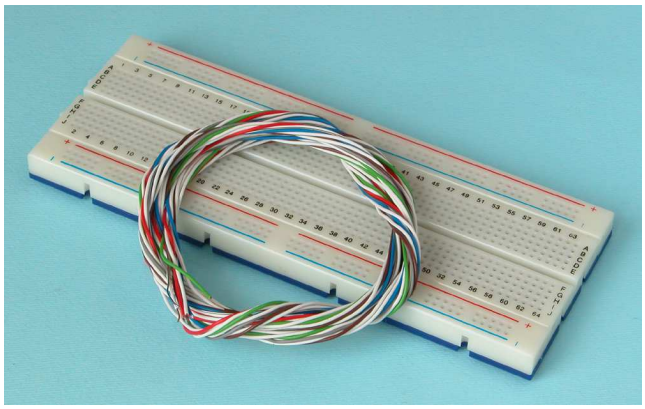


Pro výukové účely a konstrukce z oblasti robotiky byly ve Velké Británii vyvinuty procesory PICAXE, jejichž programování je maximálně zjednodušeno jak z hlediska nutného vybavení, tak složitosti psaní programu. Dá se říci, že jsou to obvody pro ty, kdo neumí nebo nechtějí programovat tak, jak se s procesory obvykle pracuje, tedy

v assembleru, ale potřebují použít procesor a rychle vyřešit svůj problém. Konstrukce malých robotů, jenž spolu soupeří, nebo jen předvádí co umí, je docela rozšířenou zálibou, a právě v těchto zařízeních se procesory PICAXE velmi často používají. Při bližším seznámení zjistíme, že domácí robotiku by bylo klidně možné považovat i za specifickou oblast modelářství, vždyť v robotech se většinou používá právě modelářská elektronika. A naopak, složitější modely schopné vykonávat určité úkony samostatně, mají velmi blízko k robotům. Přímo se tedy nabízí možnost využít procesory PICAXE také pro jednoduché typicky modelářské konstrukce.



Seriál, který právě začínáme, si bere za cíl přiblížit na praktických příkladech programování procesorů PICAXE i těm, kteří žádné předchozí zkušenosti s programováním nemají. Podmínkou je používat PC se systémem Windows, protože v počítači budeme programy pro PICAXE tvořit, a být schopen podle schématu sestavit jednoduchý obvod na úrovni blikače. Ke zkoušení konstrukcí budeme používat nepájivé kontaktní pole, po otestování funkce pak bude možné obvod osadit na desku, aby mohl plnit svůj účel v modelu.



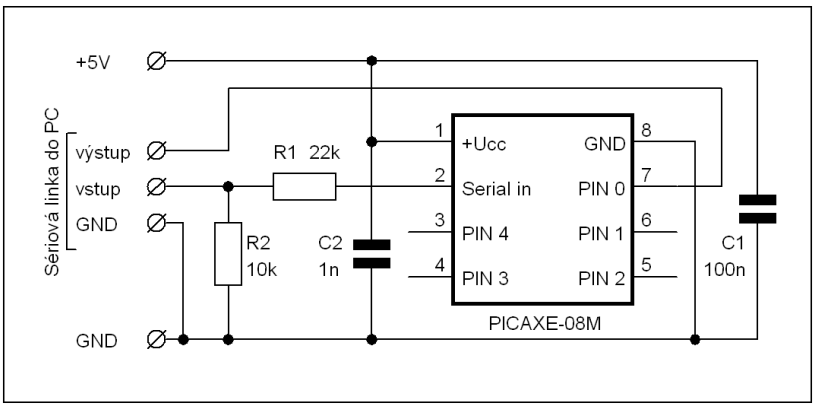
PICAXE vychází z procesorů PIC vyráběných firmou Microchip, do nichž byl již ve výrobě vložen speciální zaváděcí program a předprogramována řada užitečných funkcí. Zaváděcí program dovoluje přeprogramování uživatelského programu z PC bez nutnosti použít nákladný programátor, místo něj nám stačí pouhý kabel k sériovému portu počítače. Obvody lze přeprogramovat zhruba 100000x, což zjevně neomezuje možnosti pokusů. Zavaděč nesmíme z procesoru smazat, v tom okamžiku by se z něj stal „obyčejný“ PIC a byl by pro naše účely ztracen. Programové vybavení do PC, které budeme používat, takové smazání zavaděče neumožňuje, musíme si ale dávat pozor na dvě věci:

Nikdy nesmíme napájecí napětí procesoru přepólovat a nikdy nesmí napájení převýšit 5,5 V. I když sám procesor jako součástka obě události po krátkou dobu asi přežije, zavaděč se tím nenávratně smaže.

Procesorů PICAXE je několik typů, od malého osmivývodového PICAXE-08M po čtyřicetivýodový PICAXE-40X1. Většina konstrukcí bude postavena na nejmenším a nejlevnějším procesoru 08M, ten pro začátek naprosto stačí. Později si ukážeme i některé další možnosti, které přináší větší procesory typu 14M a 18X. První pokusy s programováním lze uskutečnit zcela bez finančních nákladů, vývojové prostředí se dá volně stáhnout a dovoluje simulovat běh programů. Simulace je zajímavá, cílem ovšem není hrát si s počítačem, ale naučit se vytvořit skutečné fungující zařízení do modelu. Souběžně s tím, jak se dostaneme k realizaci, bude možné zakoupit všechen potřebný materiál, tj. procesory, programovací kabel, nepájivé pole i balíček součástek společný pro všechny pokusné konstrukce v internetovém obchodě na stránkách *www.snailshop.cz* , celková cena nepřekročí 400 Kč. Můžeme tedy začít.

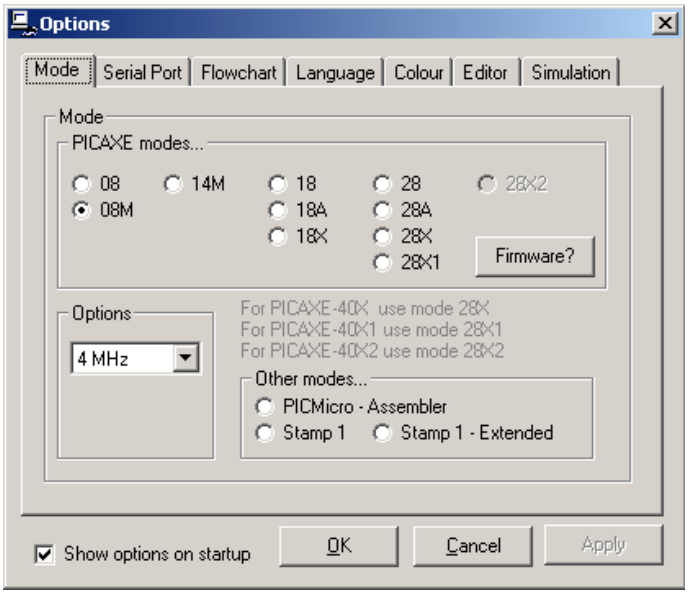
Rozmístění vývodů PICAXE-08M založeném na obvodu PIC 12F683-I/P je na schématu. Vývod 1 je kladné napájení, nemělo by klesnout pod 4,5 V a přesáhnout 5,5 V, vhodným zdrojem je čtyřčlánek Nixx akumulátorů. Vývod 2 je rezervovaný pro programování obvodu z počítače a k jiným účelům použít nejde. Má-li procesor správně pracovat (mimo programování), musí být na tomto vývodu trvale logická 0. Vývody 3 až 7 jsou vstupy a výstupy obvodu a budeme je značit v souladu s obrázkem Pin 4 až Pin 0. Pin 0 je vždy výstupem a slouží spolu s vývodem 2 ke zpětné komunikaci s počítačem, může být ale využit i v našem programu. Pin 3 je vždy nastaven jako vstup, ostatní (Piny 1,2 a 4) můžeme podle potřeby použít buď jako vstup nebo jako výstup, případně je z programu přepínat. Po zapnutí napájení procesoru jsou tyto vývody nastaveny jako vstupy. Poslední vývod číslo 8 je zem.

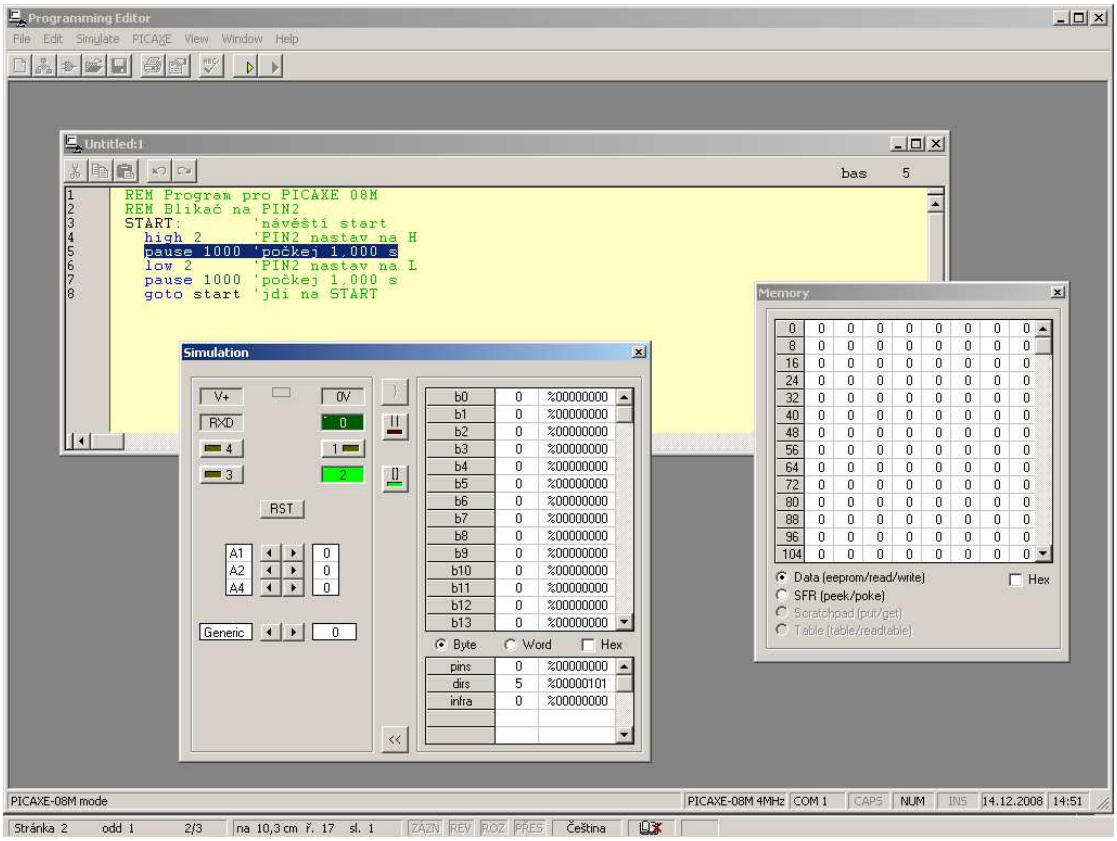
Základní zapojení procesoru, které bude obsaženo ve všech následujících konstrukcích, je na schématu. Rezistory R1 a R2 zajišťují funkci programovacího vývodu, navíc přibyl kondenzátor C1 blokující napájení a má-li být zapojení opravdu v modelu, pak je dobré ještě k němu paralelně a co nejtěsněji k vývodům procesoru přidat kondenzátor C2 typu NPO kvůli důkladnému odrušení. Vývody lze zatížit proti zemi nebo kladnému napájení proudem až 25 mA.



Prvním krokem v dalším postupu bude stáhnout si z internetových stránek *www.rcrevue.cz* programátorskou příručku v češtině (formát PDF), na niž se budeme často odkazovat, a také balíček s vývojovým prostředím, jehož součástí je editor pro psaní programu, prostředky pro vyzkoušení a simulaci uživatelských programů i kompletní dokumentace v angličtině. Program nainstalujeme do PC běžným způsobem. Další informace lze najít na českých stránkách *www.hobbyrobot.cz* nebo domovských stránkách výrobce *www.rev-ed.co.uk/picaxe*.

Po spuštění programu se jako první ukáže tabulka nastavení, vybereme si v ní typ procesoru 08M, v další záložce sériový port, přes nějž budeme procesor (časem) programovat. Můžeme si tedy vysvětlit první příkazy a začít psát. Jednotlivé řádky programu jsou očíslované, číslují a přečíslovávají se samy. Budeme-li se odvolávat na českou příručku, bude v závorce číslo strany příručky uvedené hvězdičkou**,** například (*7) je sedmá strana příručky programátora.





REM

Prvním příkazem, který poznáme, je Rem. Může být také stručněji vyjádřen středníkem (;) nebo apostrofem (‘), což je rovnocenné. Cokoli za tento příkaz napíšeme se až do konce řádku považuje za komentář, do nějž si můžeme dělat libovolné poznámky. Pokud používáme více druhů procesorů je rozumné si vždy na začátek programu poznamenat, pro který typ je program odladěn a samozřejmě také to, jak se jmenuje nebo co dělá. Komentáře s vysvětlením funkce kromě toho můžeme psát rovnou za výkonné příkazy do řádku (*1).

NÁVĚŠTÍ

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).

HIGH, LOW

Tyto dva příkazy slouží k ovládání výstupů, jejichž číslo následuje. Například High 1 tedy nastaví PIN 1 na hodnotu H (napětí blízké kladnému napájení), Low 1 nastaví PIN 1 na hodnotu L, tedy napětí blízké zemi. Podrobněji viz (*10) a (*15). Tyto příkazy současně samy nastaví příslušný PIN jako výstup, pokud by předem tak nastaven nebyl.

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 tedy bude čekat 1,5 s. Podrobněji viz (*17).

GOTO

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

Náš první program, který poslouží k „rozblikání“ výstupu PIN 2, tedy můžeme napsat do editoru, komentáře na koncích řádků podrobně připomínají, který příkaz co dělá.

```
1       REM Program pro PICAXE 08M
2       REM Blikač na PIN2
3       START:              'návěští start
4       high 2              'PIN2 nastav na H
5       pause 1000          'počkej 1,000 s
6       low 2               'PIN2 nastav na L
7       pause 1000          'počkej 1,000 s
8       goto start          'jdi na START
```

Jakmile je program napsaný, zkusíme jeho simulaci přes Ctrl+F5 nebo lépe tlačítkem se žlutou šipkou vpravo v horní liště nástrojů. Vedlejší tlačítko s podobnou modrou šipkou zatím nepoužíváme, to slouží k přenesení programu do procesoru přes programovací kabel, k tomu se dostaneme až příště. Vykonává se příkaz za příkazem, ty naše tvoří smyčku, která běhá stále dokola.

Objeví-li editor v programu nějakou formální chybu, třeba napíšeme REN místo REM, označí příslušný řádek a chybu ohlásí. Horší je to s chybami logickými, například když v parametru příkazu Low napíšeme číslo 1 místo 2, to pak samozřejmě program bude dělat přesně to, co jsme mu zadali, nemá možnost zjistit, že to není to, co chceme.

Spuštěním simulace vyvolá okno, v jehož levé horní části je schématicky naznačené rozmístění vývodů PICAXE-08M a příslušné Piny jsou označeny čísly (viz obrázek z programu). Ploška s číslem 2 (ve skupině osmi vpravo dole) by se měla rozblikat zelenou barvou přesně s periodou 2 s. Simulaci zastavíme opětovným stiskem tlačítka se žlutou šipkou. Tím byl první úkol splněn, blikač je hotový, zatím alespoň jako simulace.

Pro lepší seznámení je dobré si vyzkoušet několik úprav, například zrychlit blikání dvakrát,

dělat krátké záblesky prokládané dlouhými mezerami nebo naopak a změnit program tak, aby místo Pinu 2 blikal jiný, třeba Pin 4.

I takto jednoduchý program je možné zkrátit, a to docela podstatně. Seznámíme se s dalším příkazem.

TOGGLE

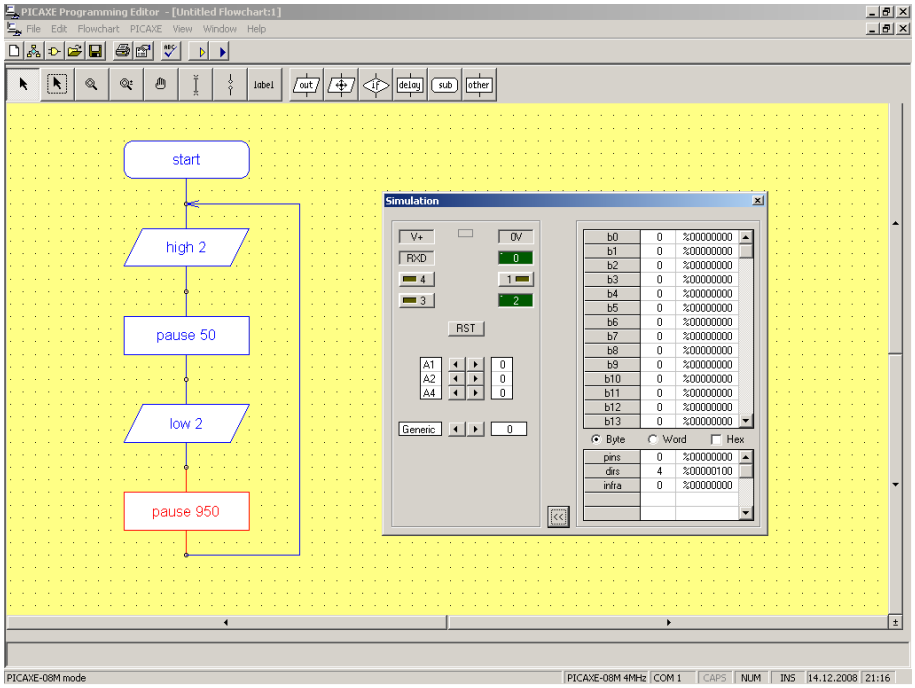
Toggle nastavuje zadaný výstup podobně jako High nebo Low, ale s tím rozdílem, že vždy změní stav výstupu, pokud tedy byl v L, nastaví H, jestliže byl v H, nastaví L. Také Toggle (*28) si sám upraví příslušný Pin do režimu výstupu.

Náš program tím 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. Doba potřebná na vykonání ostatních příkazů je zcela zanedbatelná, program tráví naprostou většinu času čekáním. Z programu není vidět, co bude na začátku na výstupu, což ovšem nijak nevadí.

```
1       REM Program pro PICAXE 08M
2       REM Blikač na PIN2 verze 2
3       START:      'návěští start
4       toggle 2    'PIN2 změň stav
5       pause 1000  'počkej 1,000 s
6       goto start  'jdi na START
```

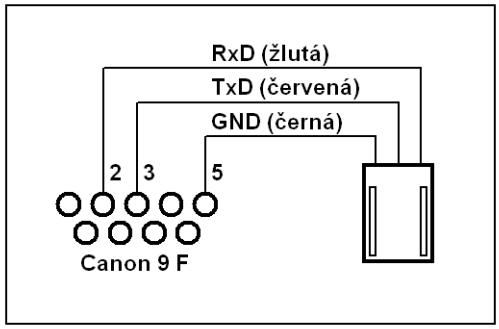
Opět si zkusíme modifikovat program tak, aby blikal jiný výstup, třeba PIN 1, zrychlit a zpomalit blikání.

Úloha z minulého dílu jde do editoru zadat i jinak než textově. Když prostřednictvím menu/file/new/new_flowchart otevřeme editor pro kreslení, či spíše sestavování blokových schémat, můžeme z nabídky nahoře brát jednotlivé příkazy a pospojovat z nich podobný program. Na obrázku je verze programu s konstantami nastavenými na krátké záblesky 0,05s s opakovací frekvencí 1 Hz. Parametry příkazů jako třeba časová konstanta v Pause se

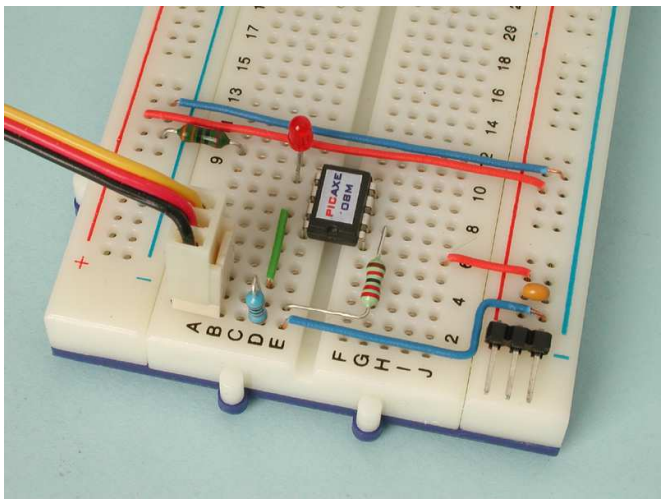


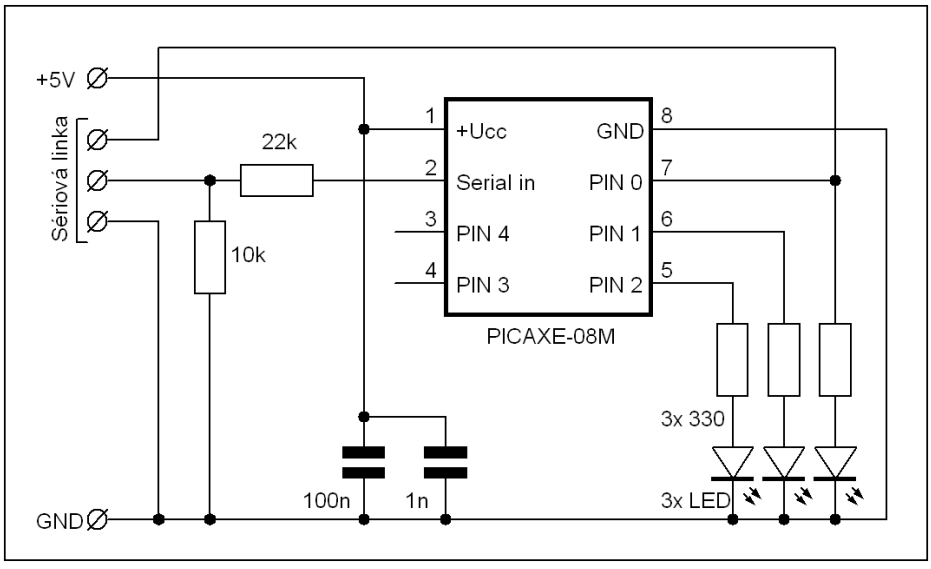
mění v dolním dialogovém řádku. Tento způsob zadávání je někdy přehlednější a sám připomíná a nabízí použitelné příkazy, vyžaduje ale podstatně více prostoru a rozsáhlejší struktury už přehlednost ztrácí. Bohužel, mezi oběma způsoby se nedá volně přecházet.

Dostáváme se k praktické realizaci. Jako první budeme potřebovat programovací kabel, ten můžeme podobně jako ostatní potřebné součástky buď koupit (*www.snailshop.cz*) nebo vyrobit, stačí k tomu devítikolíkový konektor Canon (s dutinkami, ne s kolíky, protikus sériového konektoru v PC), třížilový kabel a konektor PFH02-03P s kontakty. Zapojení je na schématu.



Obvod pospojujeme 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. Procesor můžeme osadit do precizní objímky, vzhledem k tomu, že se s ním prakticky nehýbe, to však není nutné. Pro napájení použijeme třípinový konektor ze zahnuté propojkové lišty a krajní kontakt odštípneme, takže orientací nelze napájení přepólovat. Až budeme používat tlačítka, před zasunutím do pole jim plochými kleštěmi narovnáme vývody. Dlouhé drátové vývody rezistorů nezkracujeme nebo jen málo, aby se daly použít opakovaně v různých zapojeních, výjimku tvoří rezistory R1 a R2 základního zapojení, které vždy zůstávají na svém místě. Nožičky LED zkrátíme asi na polovinu a anodu si označíme navlečením bužírky.



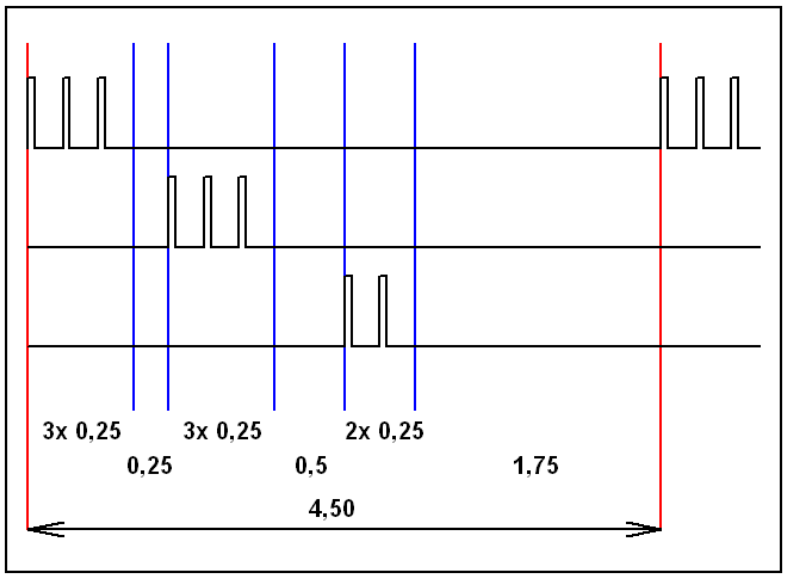


Po kontrole zapojení připojíme sériovou linku a napájecí čtyřčlánek, zatím by se nic dít nemělo. V editoru máme připravený program pro rozblikání LED na výstupu Pin 2. Stiskneme tlačítko s modrou šipkou pro přenos programu do procesoru. Mělo by se ukázat okno s obrázky částí robotů a modrý ukazatel sledující průběh programování. Při úspěšném konci procesu se zobrazí údaj o délce programu a kapacitě paměti, ten pro nás bude podstatný až u rozsáhlejších programů. Nepodaří-li se přenos, bývá problém v nepřipojeném napájení nebo napoprvé v chybně nastaveném sériovém portu, nefunkčnost po úspěšném přenosu často způsobují obráceně zapojené LED.

Jakmile se podaří první naprogramování obvodu a LED se rozbliká, vyzkoušíme opět různé varianty změn rychlosti blikání a výstupů, aby se návyk postupu programování a vyzkoušení výsledků zažil.

Zábleskova světla

Jednoduchý blikač by v praxi určitě nemělo smysl stavět s procesorem, jakmile je ale potřeba vytvořit složitější sekvenci imitující pomocí vysoce svítivých LED zábleskové výbojky, to už je něco jiného. I když máme zatím k dispozici velmi omezený sortiment příkazů, můžeme se pokusit o první skutečně prakticky využitelnou konstrukci. Použijeme tři výstupy a předem si rozkreslíme na čtverečkovaný papír požadovanou sekvenci – příklad je na obrázku. Nejprve 3x blikne LED na Pinu 0, pak 3x LED na Pinu 1 a po mezeře 2x LED na Pinu 2, následuje delší mezera a cyklus se opakuje. Novinkou je využití výstupu Pin 0, jenž současně slouží při sériovém přenosu, a tak LED připojená na tento výstup v průběhu přenášení programu svítí nebo bliká, to ale nevadí. Schéma zkušebního zapojení realizovaného na kontaktním poli je na obrázku.




```
1       REM Záblesky1 pro PICAXE 08M
2       start:          ;začátek smyčky
3         high 0        ;3 pulzy na PIN0
4         pause 50
5         low 0
6         pause 200
```

```
7       high 0
8       pause 50
9       low 0
10      pause 200
11      high 0
12      pause 50
13      low 0
14      pause 200
15      pause 250  ;mezera
16      high 1     ;3 pulzy na PIN1
17      pause 50
18      low 1
19      pause 200
20      high 1
21      pause 50
22      low 1
23      pause 200
24      high 1
25      pause 50
26      low 1
27      pause 200
28      pause 500  ;mezera
29      high 2     ;2 pulzy na PIN2
30      pause 50
31      low 2
32      pause 200
33      high 2
34      pause 50
35      low 2
36      pause 200
37      pause 1750 ;mezera dlouhá
38      goto start ;skok na začátek
```

První verze programu popisuje přesně všechny změny stavu za sebou tak, jak je čteme z rozkreslení sekvence. Je to program plně funkční, i když značně neefektivní. Jednak jeho zápis je zbytečně dlouhý, jednak zabírá (jak hlásí PC po přenesení programu) 64 byte, což je čtvrtina dostupné paměti procesoru. Asi nejdůležitější je to, že při pokusu o úpravy rychlosti běhu celé sekvence, prodloužení nebo zkrácení záblesků a podobně, bychom museli měnit stejné parametry na mnoha místech. Můžeme zkusit zkrátit dobu bliknutí LED na polovinu, přičemž časování celé sekvence zůstane zachováno, příslušné mezery se prodlouží. To si vyžádá zásah do programu na 16 místech, což je pracné a vytváří příležitost vzniku chyb. Je čas seznámit se s konstantami, symboly a operátory.

KONSTANTY

jsme již použili v prvním programu, číslo 1000 je v programu pevně zadané, tedy konstanta. Zatím stačí uvědomit si konstantu jako pojem, jiné formy zápisu v šestnáctkové nebo dvojkové soustavě případně konstanty textové najdeme v (*1).

SYMBOLY

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

OPERÁTORY

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

Nyní se pokusíme celý program napsat znovu s využitím symbolicky pojmenovaných konstant.

```
1       REM Záblesky2 pro PICAXE 08M
2       symbol perioda = 250
3       symbol blik = 50
4       symbol klid = perioda - blik
5       symbol mezera1 = 2*perioda
6       symbol mezera2 = 7*perioda
7       start:                  ;začátek smyčky
8         high 0                ;3 pulzy na PIN0
9         pause blik
10        low 0
11        pause klid
12        high 0
13        pause blik
14        low 0
15        pause klid
16        high 0
17        pause blik
18        low 0
19        pause klid
20        pause perioda         ;mezera
21        high 1                ;3 pulzy na PIN1
22        pause blik
23        low 1
24        pause klid
25        high 1
26        pause blik
27        low 1
28        pause klid
29        high 1
30        pause blik
31        low 1
32        pause klid
33        pause mezera1         ;kratší mezera
34        high 2                ;2 pulzy na PIN2
```

```
35      pause blik
36      low 2
37      pause klid
38      high 2
39      pause blik
40      low 2
41      pause klid
42      pause mezera2     ;delší mezera
43      goto start        ;skok na začátek
```

Z hlediska délky zápisu programu jsme si nepomohli, ta dokonce vzrostla, délka programu přenášeného do procesoru zůstala stejná, nesrovnatelně se však usnadnila možnost vyzkoušení různých délek záblesku a mezer. Zmenšením hodnoty konstanty „perioda“ zrychlíme celý cyklus při zachování délky (jasnosti) záblesku, konstanta „blik“ určuje nezávisle délku záblesku. „Mezera1“ a „mezera2“ odvozují délky prodlev z konstanty perioda a je třeba je použít především proto, že příkaz pause nemůže mít v parametru výpočetní výraz, jen konstantu nebo proměnnou. Tento program umožní vyzkoušet si různé nastavení konstant, zjistit, jaký nejkratší záblesk je ještě dobře vidět a podobně. V dalším kroku zkusíme program zkrátit, na to budeme potřebovat proměnné a cykly.

PROMĚNNÉ a PŘIŘAZENÍ

Proměnná je místo v paměti procesoru, kam můžeme uložit nějakou hodnotu a používat ji. Po zapnutí napájení mají všechny proměnné hodnotu 0. K dispozici máme celkem 14 proměnných jednobytových (hodnoty 0 až 255) pojmenovaných b0 až b13, samozřejmě si je můžeme označit vlastním symbolickým jménem. Je-li potřeba uložit vyšší hodnoty v rozsahu 0 až 65535, můžeme stejný prostor v procesoru 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 na stejné místo v paměti procesoru odvolávali jednou jako na proměnnou typu byte a jindy jako na word, takže pokud potřebujeme třeba w5, už bychom neměli v programu používat b10 a b11 (a naopak). Podrobnější údaje jsou v (*2). Zadání hodnoty do proměnné neboli přiřazení je jednoduché, `LET w6 = 123` uloží do proměnné w6 hodnotu 123, v některých případech se dokonce nemusí ani slovo `LET` uvádět.

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 proměnná, podle počtu musí být typu byte nebo word. 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 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 bychom neměli 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 i krok.

```
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 také 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 LED 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).

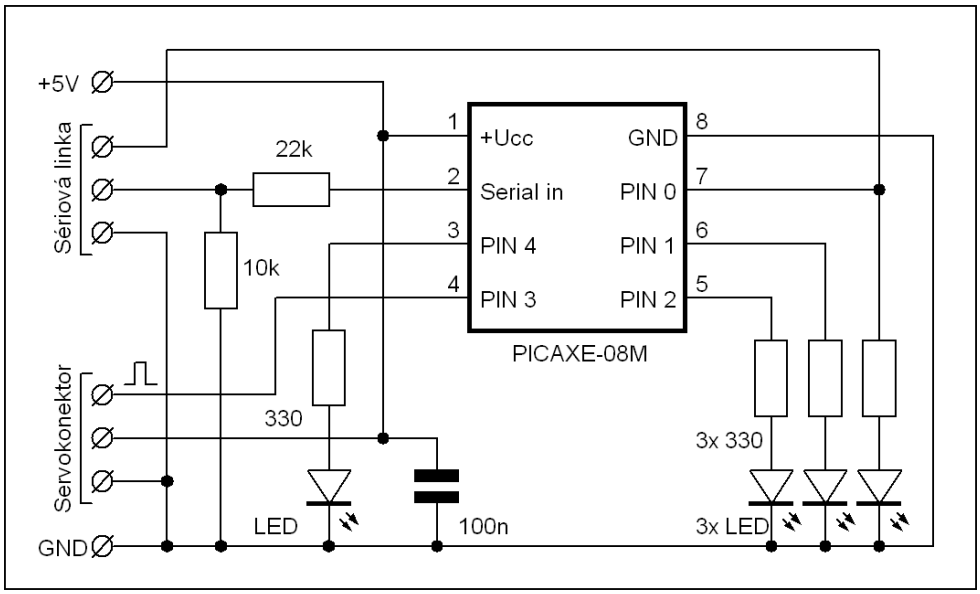
Úlohu se zábleskovými světly nyní přepíšeme za pomoci cyklů, ty pomohou nejen ke zkrácení, ale také velmi usnadní pokusy, rozhodneme-li se měnit dodatečně počty záblesků v jednotlivých sériích. Místo přepisování kusu programu stačí změnit jedno číslo v horní mezi cyklu.

```
1   REM Záblesky3 pro PICAXE 08M
2   symbol perioda = 250
3   symbol blik = 50
4   symbol klid = perioda - blik
5   symbol mezera1 = 2*perioda
6   symbol mezera2 = 7*perioda
7   start:                      ;začátek smyčky
8    for b0 = 1 to 3            ;3x pulz PIN0
9    high 0
10   pause blik
11   low 0
12   pause klid
13   next b0
14   pause perioda              ;mezera
15   for b0 = 1 to 3            ;3x pulz PIN1
16   high 1
17   pause blik
18   low 1
19   pause klid
20   next b0
21   pause mezera1              ;mezera delší
22   for b0 = 1 to 2            ;2x pulz PIN2
23   high 2
24   pause blik
```

```
25      low 2
26      pause klid
27      next b0
28      pause mezera2           ;mezera dlouhá
29      goto start              ;skok na start
```

Všimneme si, že se zkrátil i vlastní program přenášený do procesoru, a to asi o 20%, což není mnoho, ale roli už to hraje. Samozřejmě cyklus nám ušetří tím víc, čím vícekrát se děj opakuje, u dvou opakování možná nic, u třech trochu, ale pokud se něco má opakovat třeba 100x, jinak než cyklem to už prakticky napsat nejde.

K programu pro řízení zábleskových světel se ještě vrátíme, zatím si rozšíříme zkušební zapojení podle dalšího schématu. Volný konec servokabelu připájíme na zahnuté kontakty a zasuneme do pole, signálový vodič přivedeme na Pin 3. Kabel bude sloužit k připojení přijímače můžeme případně použít i servotester. Naučíme se číst a zpracovávat servosignál a vytvořit jednoduché i složitější RC spínače. Do sortimentu příkazů si doplníme další dva.



Seznam součástek pro zkušební konstrukce k seriálu PICAXE:

1	IRL3705N (FET buzení L)
2	IRE5 (fototranzistor)
2	BC337-40 (tranzistor)
4	LED červené průměr 3 mm
1	1N5820 (dioda)
1	PT1540-P (piezoměnič)
2	P-B1720A (tlačítko)
2	C keramický 100 n
1	Jumperová lišta úhlová, 12 pinů
1	PSH02-03WG konektor
1	Drát průměr 0,5 v PVC, cca 5 m
1	Servokabel
2	Trimr 10k naležato, 10 mm
6	R 330 ohm

2 R 180 ohm
2 R 22k
8 R 10k
4 R 1 k
Mimo balíček:
1 Kontaktní pole BX-4112N
1 Programovací sériový kabel
1 PICAXE 08M

PULSIN

PICAXE má speciální příkaz pro měření délky pulzu na libovolném vstupu, který se „shodou okolností“ velmi dobře hodí pro čtení servosignálu. Příkaz „Pulsin“(*18) má tři parametry, první je číslo pinu, s nímž má pracovat, druhý určuje, zda se bude pulz měřit od náběžné hrany (1) nebo sestupné hrany (0) signálu a třetí udává proměnnou, do níž se uloží délka pulzu v desítkách mikrosekund. Standardní servosignál je tvořen kladnými impulzy o délce 500 až 2500 mikrosekund (v nejširším možném pojetí, běžné meze jsou 1000 – 2000 mikrosekund), takže jako druhý parametr se zpravidla uplatní číslo1 a proměnná může být i typu byte, lépe je ale používat proměnnou typu word. Jestliže pulz nepřijde, skončí měření po 0,65 s a v proměnné vrátí číslo 0, tak můžeme otestovat, že měření bylo neúspěšné. Například „pulsin 3,1,w6“ bude čekat, až se na Pin 3 dostane náběžná hrana impulzu, pak změří jeho délku v desítkách mikrosekund a výsledek vrátí v proměnné w6 (takže proměnné b12 a b13 již nebudeme používat, ty využívají stejné místo v paměti). Servosignál nám vrátí hodnoty 50 až 250 (obvykle 100 - 200), neutrál by měl odpovídat číslu 150.

IF … THEN

Velmi často je potřeba program rozvětvit a podle hodnoty nějaké proměnné pokračovat buď jednou nebo druhou cestou. K tomu slouží příkaz IF (*11) za nímž je název proměnné, s kterou budeme pracovat, pak relační operátor (nejčastěji =; <; > nebo <> (nerovná se)), konstanta nebo druhá proměnná, jejíž hodnotu porovnáváme. Za slovem THEN bude 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 však i jinou podobu, kterou v příručce nenajdeme. 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 napsat „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.

RC spínač

Zkušební zapojení máme připravené a připojené k přijímači (a víme, kterým ovladačem na vysílači se použitý kanál řídí), takže můžeme zapsat a vyzkoušet program pro nejjednodušší RC spínač, jímž podle ovládání rozsvítíme nebo zhasneme LED připojenou na Pin 4.

```
1       REM RC spinač 1 pro PICAXE 08M
```

```
2       symbol stred = 150              ;hodnota pro mezní pulz
3       start:                          ;návěští smyčky programu
4         pulsin 3,1,w6                 ;měření kladných pulzů na PIN3
5         if w6 > stred then zapni      ;když je pulz > střed zapni LED
6         if w6 <= stred then vypni     ;když je pulz <= střed vypni LED
7         goto start                    ;skok na začátek programu
8       zapni:                          ;návěští pro zapnutí LED
9         high 4                        ;nastav výstup PIN4 na H
10        goto start                    ;skok na začátek programu
11      vypni:                          ;návěští pro vypnutí LED
12        low 4                         ;nastav výstup PIN4 na L
13        goto start                    ;skok na začátek programu
```

Takto zapsaný program vychází z vysvětlení příkazu IF v příručce. Konstanta „stred“ určuje mez pro rozsvícení LED, 150 odpovídá neutrálu. Použijeme-li druhou uvedenou možnost, bude stejný program vypadat takto:

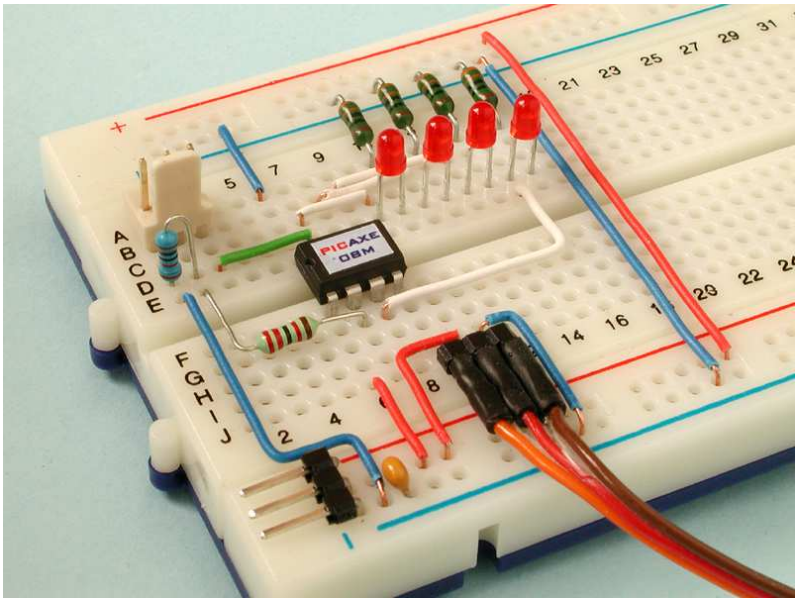
```
1       REM RC spinač 2 pro PICAXE 08M
2       symbol stred = 150              ;hodnota pro mezní pulz
3       start:                          ;návěští smyčky programu
4         pulsin 3,1,w6                 ;měření kladných pulzů na PIN3
5         if w6 > stred then high 4 else lov 4 endif
6         goto start                    ;skok na začátek programu
```

Zápis programu se zkrátil na necelou polovinu, délka přenášeného programu klesla o čtvrtinu. Výhody tohoto druhého způsobu jsou asi nejvýraznější, tvoří-li stejně jako v tomto případě obě větve příkazu IF jen jeden příkaz. Je čas si vyzkoušet posunout mez rozsvícení LED do poloviny výchylky ovladače nebo blízko k jeho kraji.

Naprogramovaný RC spínač zatím reaguje v těsném okolí meze sepnutí velmi citlivě a LED poblikává, to je samozřejmě nežádoucí jev. Stejně tak bývá zvykem ošetřit pulzy mimo regulérní rozsah, což není nijak složité. Za meze servopulzů si pro další práci si stanovíme 0,8 a 2,2 ms, vyjádřeno v desítkách mikrosekund konstanty 80 a 220.

```
1       REM RC spinač 3 pro PICAXE 08M
2       start:
3         pulsin 3,1,w6
4         if w6>151 and w6<220 then high 4 endif
5         if w6<149 or w6>220 then low 4 endif
6         goto start
```

Je-li pulz delší než 1,51 ms a současně kratší než 2,20 ms, LED se rozsvítí, je-li kratší než 1,49 ms nebo delší než 2,20 ms, LED zhasne. O to, aby LED na mezi neblikala, se stará „pásmo nikoho“ mezi 1,49 a 1,51 ms, v němž zůstane zachován předchozí stav (program se o toto pásmo nestará). Zkusíme změnit poslední program tak, aby se mez spínání posunula. Jestliže není pásmo necitlivosti na ovládací páce zřetelně znát, rozšíříme jej třeba na dvojnásobek.



Velmi podobným způsobem ve stejném zapojení můžeme vyzkoušet složitější RC spínač, a to rovnou čtyřnásobný s využitím všech čtyř LED. Spínač bude mít klidovou polohou v neutrálu a aktivuje své výstupy vždy jednotlivě. Aby při přepínání LED neblikaly, jsou meze podobně jako v předchozím případě ošetřeny hysterezí, to je ono „pásmo nikoho“, v němž se skryje případná drobná nestabilita zdroje signálu nebo jeho vyhodnocení. Program je připravený pro experimenty a má na začátek vynesených šest parametrů, jimiž lze upravit jeho funkci. Šířka hystereze 1 většinou stačí, je-li třeba, rozšíříme ji na 2 nebo 3, vyšší hodnoty než 5 už postrádají smysl.

```
1     REM RC spinač 4 pro PICAXE 08M
2     symbol mez0 = 160                ;mez sepnutí PIN0 (směr nahoru)
3     symbol mez1 = 180                ;mez sepnutí PIN1 (směr nahoru)
4     symbol mez2 = 140                ;mez sepnutí PIN2 (směr dolů)
5     symbol mez4 = 120                ;mez sepnutí PIN4 (směr dolů)
6     symbol mezmin = 80               ;dolní mez regulérního signálu
7     symbol mezmax = 220              ;horní mez regulérního signálu
8     symbol hyst = 1                  ;hystereze pro sepnutí bez kmitů
9     symbol mez0h = mez0 + hyst       ;mez rozšířená o hysterezi nahoru
10    symbol mez1h = mez1 + hyst       ;mez rozšířená o hysterezi nahoru
11    symbol mez2h = mez2 - hyst       ;mez rozšířená o hysterezi dolů
12    symbol mez4h = mez4 - hyst       ;mez rozšířená o hysterezi dolů
13    start:                           ;začátek programu
14     pulsin 3,1,w6
15                                     ;spínání PIN0 ošetřené hysterezí
16     if w6>mez0h and w6<mez1 then high 0 endif
17     if w6<mez0 or w6>mez1h then low 0 endif
18                                     ;spínání PIN1 ošetřené hysterezí
19     if w6>mez1h and w6<mezmax then high 1 endif
20     if w6<mez1 or w6>mezmax then low 1 endif
21                                     ;spínání PIN2 ošetřené hysterezí
22     if w6<mez2h and w6>mez4 then high 2 endif
23     if w6>mez2 or w6<mez4h then low 2 endif
24                                     ;spínání PIN4 ošetřené hysterezí
```

```
25      if w6<mez4h and w6>mezmin then high 4 endif
26      if w6>mez4 or w6<mezmin then low 4 endif
27      goto start                              ;skok na začátek programu
```

Pro některé aplikace, třeba spínání okruhů světel na lodi, se hodí spínač s neutrálem na kraji výchylky a postupným spínáním výstupů tak, že se další vždy přidávají k předchozím. Jak by vypadala úprava pro tento případ? Program uvádíme už bez komentářů a ve zhuštěné formě, na začátku jsou nastavitelné čtyři meze spínání a také mez regulérnosti pulzů a velikost hystereze. Podobných dalších modifikací můžeme najít a vyzkoušet velké množství.

```
1       REM RC spinač 5 pro PICAXE 08M
2       symbol mez0=120 symbol mez1=140 symbol mez2=160
3       symbol mez4=180 symbol mezmax=220 symbol hyst = 1
4       symbol mez0h=mez0+hyst symbol mez1h=mez1+hyst
5       symbol mez2h=mez2+hyst symbol mez4h=mez4+hyst
6       start: pulsin 3,1,w6
7         if w6>mez0h and w6<mezmax then high 0 endif
8         if w6<mez0 then low 0 endif
9         if w6>mez1h and w6<mezmax then high 1 endif
10        if w6<mez1 then low 1 endif
11        if w6>mez2h and w6<mezmax then high 2 endif
12        if w6<mez2 then low 2 endif
13        if w6>mez4h and w6<mezmax then high 4 endif
14        if w6<mez4 then low 4 endif
15        goto start
```

Poslední úlohou na toto téma bude spínač rozšiřující počet ovládaných funkcí u lodě jednoduchým trikem. Vstup připojíme paralelně k servu ovládajícímu směr. Budeme-li mít trim uprostřed nebo jen mírně vychýlený, projeví se pohyby ovládací páky jen na servu, spínač reagovat nebude. Vychýlíme-li trim na doraz a současně ťukneme o doraz i ovladačem, zapne se jedna doplňková funkce (třeba světla), vypne se následně stejným způsobem. Trim i páka vychýlené na druhou stranu mohou podobně ovládat druhou funkci (druhý okruh světel), aby to ale bylo zajímavější, budeme předpokládat, že druhou ovládanou funkcí je siréna a ta má na povel zahoukat předem nastavenou dobu, i když povel mezitím skončil. Zatím bereme sirénu jen jako spínané zařízení, časem se dostaneme i k tomu, že zvuk budeme generovat přímo procesorem.

Zapojení nemusíme nijak upravovat, vstup řídícího signálu máme, výstup světel bude na Pinu 1, spínání sirény na Pinu 2, stav obou budeme pozorovat na LED. K sepnutí dochází až při návratu páky zpět.

```
1       REM RC spinač 6 pro PICAXE 08M
2       REM ovládání světel a sirény kanálem směru
3       symbol mez1h=210            ;nastavení příznaku světla
4       symbol mez1d=200            ;zapnutí/vypnutí světla
5       symbol mez2h=100            ;nastavení příznaku siréna
6       symbol mez2d=90             ;spuštění siréna
7       symbol mezmax=220           ;maximální regulérní pulz
8       symbol mezmin=80            ;minimální regulérní pulz
9       symbol houk=200             ;délka sirény v 1/50 s
10      REM w0 - čtení vstupu
```

```
11      REM b2,b3 - pracovní, sledování výchylky
12      REM w2 - počítání doby houkání
13      smycka:
14        pulsin 3,1,w0                    ;načtení vstupu
15        ;páka nahoru -> nastavení příznaku pro světlo
16        if w0>mez1h and w0<mezmax then let b2=1 endif
17        ;páka dolů -> nastavení příznaku pro sirénu
18        if w0<mez2d and w0>mezmin then let b3=1 endif
19        ;návrat páky -> zapnutí / vypnutí světla
20        if w0<mez1d and b2=1 then toggle 1 let b2=0 endif
21        ;navrat páky -> spuštění sirény
22        if w0>mez2h and b3=1 then let w2=houk let b3=0 endif
23        ;počítání doby sirény
24        if w2>0 then high 2 let w2=w2-1 else low 2 endif
25        goto smycka ;zpět na smyčku programu
```

Zábleskova světla podruhé

Odbočení k RC spínačům jsme uzavřeli, i když ne úplně. Vrátíme se ke konstrukci zábleskových světel a doplníme ji tak, aby se dala ovládat třípolohovým spínačem na vysílači. V jedné krajní poloze budou světla zhasnuta, ve střední začnou blikat zábleskova světla (LED na Pin 0; 1 a 2) a v druhé krajní se k nim přidá rozsvícení přistávacího světlometu (LED na Pin 4). Tento případ v sobě spojuje dvě již zvládnuté úlohy, nicméně musí dokázat prostřídat obě linie činnosti tak, aby se změny v ovládání projevily pokud možno co nejdříve i když celá sekvence záblesků trvá kolem 4 sekund. Stále pracujeme se stejným zapojením. Přidáme si další příkaz.

GOSUB ... RETURN

Gosub je příkaz skoku, ale na rozdíl od Goto 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 nazývanou podprogram, jež se má na zavolání Gosub vykonat, musíme samozřejmě označit návěštím. Počet použití příkazů Gosub je omezen, viz (*9).

```
1       REM Záblesky4 pro PICAXE 08M
2       symbol perioda = 250                ;nastavení periody blikání
3       symbol blik = 50                    ;délka záblesku
4       symbol klid = perioda - blik        ;pracovní, mezera mezi záblesky
5       symbol mezera1 = 2                  ;mezera mezi 1. a 2. sérií (*per)
6       symbol mezera2 = 7                  ;mezera mezi 2. a 3. sérií (*per)
7       symbol mez1 = 130                   ;mez sepnutí záblesků
8       symbol mez2 = 170                   ;mez sepnutí přistávacího světla
9       symbol mezmin = 80                  ;minimální regulérní impulz
10      symbol kam = b1                     ;pracovní proměnná - určuje LED
11      start:                              ;začátek smyčky hlavního programu
12        ;1. série
13        for b0 = 1 to 3 let kam=0 gosub zablesk next b0
14        ;1. mezera
15        pause perioda
```

```
16      ;2. série
17      for b0 = 1 to 3 let kam=1 gosub zablesk next b0
18      ;2. mezera
19      for b0 = 1 to mezera1 kam=4 gosub zablesk pause perioda next b0
20      ;3. série
21      for b0 = 1 to 2 let kam=2 gosub zablesk next b0
22      ;3. mezera
23      for b0 = 1 to mezera2 kam=4 gosub zablesk pause perioda next b0
24      goto start                                  ;konec hlavního programu
25     zablesk:                                     ;začátek podprogramu zablesk
26      pulsin 3,1,w6                               ;čtení pulzů z přijímače
27      if w6 < mezmin then let w6=250 endif        ;pro krátké pulzy rozviť
28      if w6 > mez2 then high 4 else low 4 endif ;zapnutí blikání
29      if kam<4 then                               ;blikej jen pro LED 0-2
30      if w6 > mez1 then high kam endif            ;záblesk LED podle "kam"
31      pause blik
32      low kam
33      pause klid
34      endif
35      return                                      ;konec podprogramu zablesk
```

RC spínač se stal součástí podprogramu, jenž vykonává jeden záblesk LED. Bylo třeba zajistit, aby tento podprogram podle situace pracoval s LED na Pinu 0, 1 nebo 2, k tomu slouží proměnná „kam“, kterou musíme před použitím vždy nastavit. Ovládání světlometu (Pin 4) má vždy nějaké zpoždění, ale v praxi zcela nenápadné díky tomu, že i během dlouhých mezer se volá podprogram „zablesk“ každou periodu, ale s nastavením kam=4, kdy se jen zkontroluje, zda se nemá světlomet zapnout nebo vypnout. Ošetření vstupních pulzů mimo rozsah je tu opačné, světla se v případě chyby rozsvítí, takže za letu poblikávání světel případně signalizuje rušení nebo při ztrátě řízení usnadní hledání modelu za tmy.

Záblesky se zatím střídaly se železnou pravidelností, pokud bychom chtěli, aby budily dojem, že každé zábleskové světlo má svou vlastní periodu, budeme zkrátka tuto periodu pro každý výstup počítat zvlášť. Následující program Záblesky 5 ukazuje, jak to lze řešit pro tři (okruhy) LED. Program není zpomalován příkazem Pause, ale běží maximální rychlostí. Jestliže zadáme do parametrů jednotlivých period větší prvočísla (maximální hodnota je 255, použili jsme bytovou proměnnou), zopakuje se situace za poměrně dlouhou dobu, v našem konkrétním případě po provedení 1495103 cyklů a protože jeden cyklus trvá přibližně 7 ms, odpovídá perioda celého děje téměř 3 hodinám. Vhodná prvočísla mezi 100 až 255 jsou uvedena v posledních dvou řádcích v komentáři.

Parametr nastavení doby bliknutí vychází poměrně malý, dá se tedy upravit jen v hrubých krocích. Hodilo by se nám mít rychlejší procesor, což by umožnilo na stejný dosažený čas zvýšit hodnoty parametrů a mít tím možnost je nastavit jemněji. Taková možnost opravdu existuje a je v programu použita před začátkem programové smyčky, stačí odstranit středník a komentář změnit na výkonný řádek.

SETFREQ

PICAXE 08M může příkazem změnit vnitřní hodinovou frekvencí na dvojnásobnou, tím se většina časových údajů zkrátí, třeba parametr Pause by již nebyl v ms, ale násobcích 0,5 ms.

Parametr „m4“ nastaví základní obvyklou frekvenci 4 MHz, „m8“ přepne na 8 MHz, více možností není. Rychlejší běh procesoru bychom měli používat jen pokud to má opravdu smysl, může totiž způsobit problémy v provádění některých příkazů, třeba při sériové komunikaci.

Vyzkoušíme si chod jak se základní hodinovou frekvencí, tak se zvýšenou, podobně můžeme experimentovat s délkou period a rozšířit obsluhu na všechny čtyři LED. Program Záblesky5 je velmi vhodný pro simulaci v PC, běh se dá dobře pozorovat na hodnotách proměnných a indikátorech výstupů, při simulaci ale samozřejmě rozdíl v hodinové frekvenci nemá žádný vliv.

```
1   REM Záblesky5 pro PICAXE 08M
2   symbol perioda0=101             ;perioda blikání PIN0
3   symbol perioda1=113             ;perioda blikání PIN1
4   symbol perioda2=131             ;perioda blikání PIN2
5   symbol blik = 6                 ;délka záblesku (jednotná)
6   ;proměnné pro počítání periody blikání na PIN0,1,2 jsou b0,b1,b2
7   ;proměnné pro počítání doby záblesků na PIN0,1,2 jsou b3,b4,b5
8   ;setfreq m8 ;nastavení hodinové frekvence 8MHz
9   cyklus:                         ;smyčka hlavního programu
10    let b0=b0+1                   ;počítání periody PIN0
11    let b1=b1+1                   ;počítání periody PIN1
12    let b2=b2+1                   ;počítání periody PIN2
13    ;pro všechny LED: proběhla perioda ==> nastav bliknutí
14    if b0=perioda0 then let b0=0 let b3=blik endif
15    if b1=perioda1 then let b1=0 let b4=blik endif
16    if b2=perioda2 then let b2=0 let b5=blik endif
17    ;pro všechny LED: aktivní bliknutí ==> rozsviť (jinak zhasni)
18    if b3>0 then let b3=b3-1 high 0 else low 0 endif
19    if b4>0 then let b4=b4-1 high 1 else low 1 endif
20    if b5>0 then let b5=b5-1 high 2 else low 2 endif
21    goto cyklus                   ;opakuj cyklus
22  ;prvočísla: 101,103,107,109,113,127,131,137,139,149,151,157,163
23  ;167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251
```

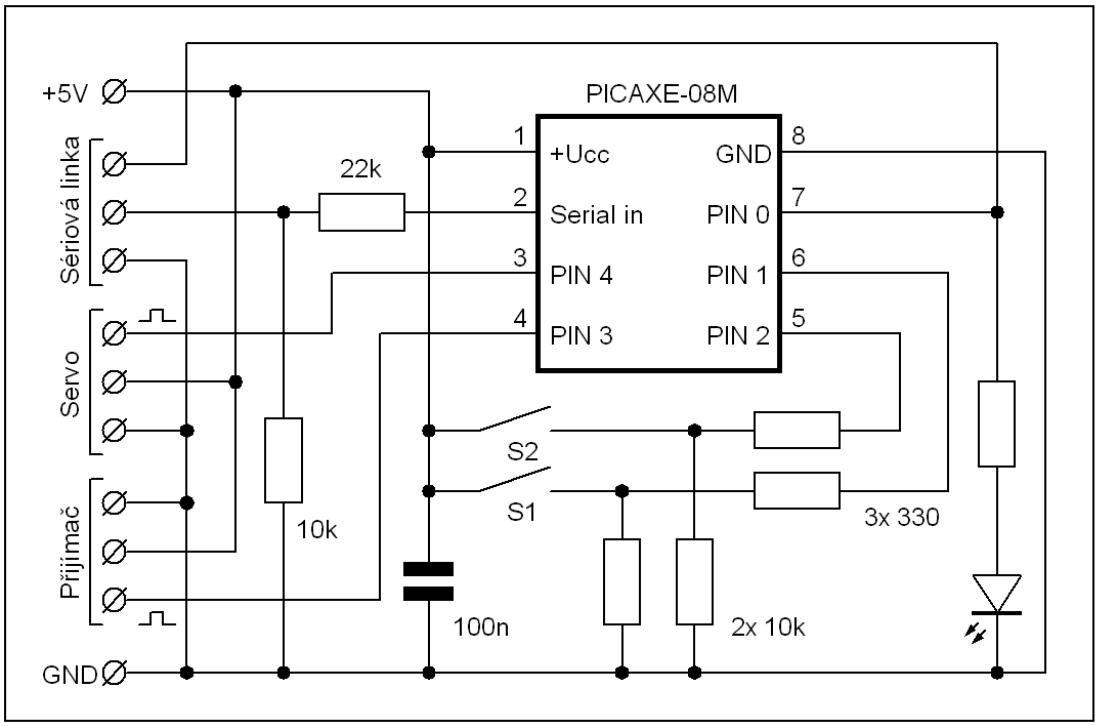
Dalším úkolem, na nějž je vhodné se zaměřit, je rozšířit program Záblesky5 o ovládání servosignálem podobně, jako to bylo v programu Záblesky4. Je to vlastně jednodušší než původně, nemusíme se starat o nějaké střídání, zkrátka v každém cyklu změříme impulz servosignálu a podle výsledku necháme procházet část starající se o blikání nebo ne, podle porovnání s druhou mezí rozsvítíme nebo zhasneme LED na Pinu 4 (světlomet). V tomto případě dojde k zajímavému jevu, základní cyklus programu trvá kolem 7 ms, ale čtení servosignálu bude vždy čekat až na výskyt pulzu na vstupu, takže se program „přichytí“ a zasynchronizuje na opakování servosignálu přibližně po 20 ms. Budeme muset výrazně upravit (2 až 3x zmenšit) hodnoty parametrů pracujících s délkou cyklu jako jednotkou času, a to i pokud procesor přepneme na vyšší rychlost. Vyšší rychlost se projeví na výstupu z příkazu „pulsin“, neutrál servosignálu bude teď odpovídat číslu 300. Tato verze programu označená Záblesky6 je dostupná pro kontrolu v balíčku souborů k tomuto dílu na internetových stránkách *www.rcrevue.cz* .

Obsluha serv

Už jsme se naučili servosignál číst a využívat, je na čase zvládnout také generování servosignálu a ovládání serv, tím se otevře široká škála jednoduchých a současně velmi užitečných aplikací. Vrátíme se současně, alespoň pro začátek, ke krátkým a snadno pochopitelným programům. Upravíme zkušební zapojení na desce podle schématu, odpadnou tři LED, naopak přidáme výstupní konektor pro připojení serva a připravíme si i čtyřnásobný DIP spínač, z něhož využijeme zatím polovinu. Ke zkouškám je lepší použít běžné analogové servo než nějaké superpřesné digitální, signál z procesoru PICAXE projevuje v některých případech drobné nestability, které pásmo necitlivosti levných serv schová, zatímco na velmi přesných servech můžeme pozorovat chvění.

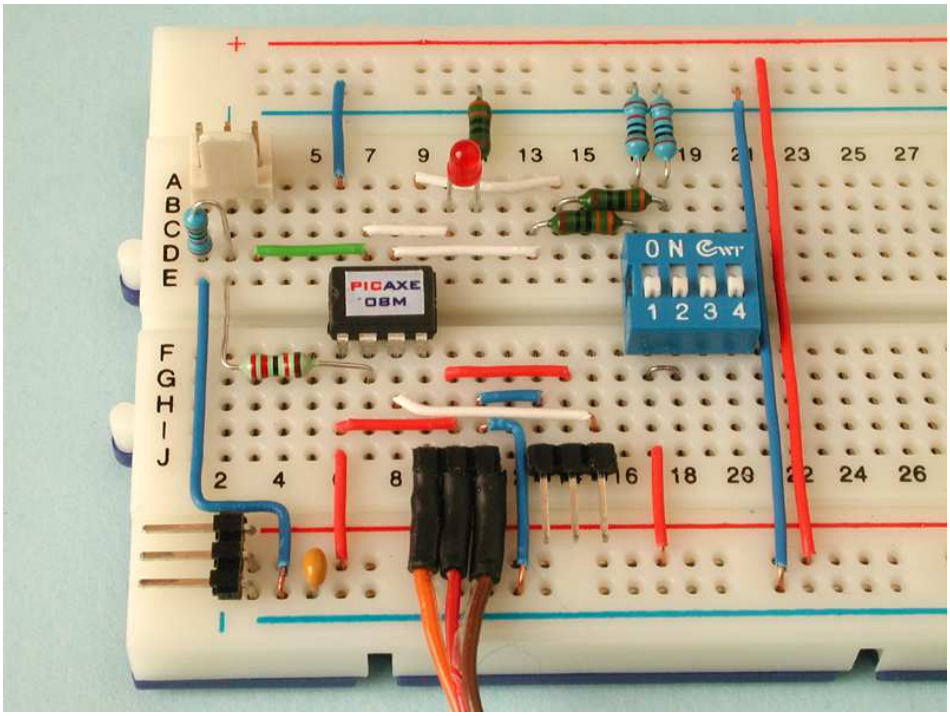
PULSOUT

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 nebo 5 mikrosekund při 8 MHz.



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). Příkaz Servo nemůže být použit při vyšší taktovací frekvenci než základní 4 MHz.



Pro ochranu serva bude třeba zajistit, aby jeho řídící pulzy ani při experimentech neopustily meze 0,8 až 2,2 ms, k tomu využijeme operátory MIN a MAX (*3), s nimiž jsme se již seznámili, ale zatím je nepoužili. Je na místě také připomenout, že matematické výrazy se striktně vyhodnocují zleva doprava bez ohledu na obvyklou přednost operací a nepomohou ani závorky, ty editor nezná.

Budeme potřebovat čtení stavu vstupu, to jsme také zatím nedělali. Není nutný žádný nový příkaz, jednoduše pracujeme s označením Pin1 až Pin4 (Pin0 ne, to je vždy výstup) tak, jako by to byly proměnné nebo konstanty, najdeme v nich číslo 0 pro úroveň vstupu L nebo číslo 1 pro úroveň vstupu H. Lze tedy napsat „IF pin1=0 then …“ nebo přiřadit hodnotu ze vstupu do proměnné „LET b0=pin1“.

```
1       REM SERVO1 pro PICAXE 08M
2       REM kopírování a omezení polohy
3       setfreq M8                      ;frekvence 8 MHz
4       smycka:                         ;začátek programu
5         pulsin 3,1,w0                 ;čtení vstupu do w0
6         REM sem vložíme výkonnou část
7         let w0=w0 min 160 max 440     ;omezení
8         pulsout 4,w0                  ;vyslání pulzu dle w0
9         goto smycka                   ;zpět na začátek
```

Program Servo1 čte signál z přijímače a přenese stejné ovládání na výstup, navíc omezuje délku pulzů shora i zdola. Budeme využívat zvýšený hodinový kmitočet procesoru 8 MHz, protože poskytuje jemnější rozlišení polohy. Může se zdát, že sám o sobě má tento program mizivý význam, vlastně jen zpožďuje signál, mezi načtení a vyslání pulzu ale místo komentáře vložíme následně příkazy, jimiž signál nebo funkci nějak upravíme, například doplníme indikaci překročení mezí servosignálu rozsvícením LED na Pinu 0 (kompletní program Servo2 v balíčku souborů na internetových stránkách *www.rcrevue.cz*).

```
if w0>440 or w0<160 then
  high 0 else low 0 endif
```

Nemůžeme-li funkci indikace a omezení vyzkoušet přímo signálem z vysílače, zmenšíme interval tak, abychom se do něj dostali s podporou trimu (například na 1,1 až 1,9 ms, což odpovídá konstantám 220 a 380). Dále vyzkoušíme obrácení výchylky serva, je to jednodušší, než se na první pohled zdá (program Servo3).

```
w0=600-w0
```

Stejně jednoduché je i rozšíření nebo zúžení výchylky serva tak, aby poloha středu zůstala zachována, jen musíme správně určit konstanty. Procesor nepracuje s desetinnými čísly, takže rozšíření zapíšeme jako zlomek, třeba 1,27 = 127/100. Vynásobením jsme ale posunuli střed (300), 300*127/100 = 381, musíme tedy odečíst 81, abychom se dostali zpět na číslo 300 (program Servo4). Podobně můžeme rozsah i zúžit, například pro koeficient 0,67 = 67/100 vychází posun 300 – (300*67/100) = 99.

```
w0=w0*127/100-81
```

nebo

```
w0=w0*67/100+99
```

Na stejném základu může pracovat i změna charakteristiky průběhu serva obvykle nazývaná exponencialita. V našem zjednodušeném případě nahradíme hladkou křivku rozdělením pohybu na tři úseky, v prostředním bude výchylka zúžená, v krajních naopak rozšířená tak, aby horní a dolní mez výchylky alespoň přibližně zůstaly zachovány a samozřejmě v místě zlomů na sebe úseky přesně navazovaly.

Zvolíme-li ve středním úseku (1,25 – 1,75 ms) zúžení výchylky na polovinu, vychází nám výraz (w0 / 2 + 150). Aby se zmenšení výchylky „dohnalo“ v běžně používaných mezích 1,0 a 2,0 ms, v krajích výchylku rozšíříme na 3/2, pro každý úsek ale bude jiný posun, dole (w0 * 3 / 2 - 100), nahoře (w0 * 3 / 2 - 200), viz program Servo5 .

```
if w0<=350 and w0>=250 then let w0=w0/2+150 endif
if w0>350 then let w0=w0*3/2-200 endif
if w0<250 then let w0=w0*3/2-100 endif
```

Potřebujeme-li snížit rychlost pohybu serva, jde to samozřejmě také, ale je to složitější. Budeme pracovat s proměnnou w0, ta přináší požadovanou polohu ze vstupu, a w1, která plní úlohu výstupní proměnné, respektive z hlediska procesoru obsahuje skutečnou polohu serva. LED svítí, pokud je požadovaná poloha různá od skutečné, tedy pokud se zpomalení pohybu uplatňuje. Tak jak je program Servo6 v nejjednodušší verzi napsán, pracuje s pevně danou rychlostí odpovídající přejezdu serva mezi koncovými polohami při buzení 1 – 2 ms za 4 s.

```
1    REM SERVO6 pro PICAXE 08M
2    REM snížení rychlosti serva
3    REM w0 požadovaná poloha
```

```
4       REM w1 skutečná poloha (výstup)
5       setfreq M8                      ;frekvence 8 MHz
6       let w1 = 300                    ;výchozí (střed)
7       smycka:                         ;začátek programu
8         pulsin 3,1,w0                 ;čtení vstupu do w0
9         if w1<w0 then let w1=w1+1 endif
10        if w1>w0 then let w1=w1-1 endif
11        if w1<>w0 then high 0
12          else low 0 endif            ;indikace
13        let w1=w1 min 160 max 440
14        pulsout 4,w1                  ;vyslání pulzu
15        goto smycka                   ;zpět na začátek
```

Následující program Servo7 kombinuje tři předchozí funkce, stále zapnuté omezení výchylky a samostatnými spínači ovládaný reverz a snížení rychlosti serva. Zajímavě je využitá indikační LED, plně svítí při překročení mezí servoimpulzů a částečným jasem vyvolaným rychlým spínáním při zpomalování chodu serva. Navržený program dovoluje spínači měnit funkci kdykoli během chodu, zkuste jej modifikovat tak, aby funkce odpovídala nastavení spínačů po zapnutí napájení a pozdější manipulace se spínači již neměla žádný vliv. Tuto verzi najdete v kompletním rozsahu pod názvem Servo8 (v programovém balíčku na stránkách www.rcrevue.cz).

```
1       REM SERVO7 pro PICAXE 08M
2       REM spínání dvou funkcí
3       setfreq M8
4       let w1 = 300
5       smycka:
6         pulsin 3,1,w0
7         ;spinač 1 - reverze
8         if pin1=1 then let w0=600-w0 endif
9         ;spinač 2 - servoslow
10        if pin2=1 then
11          if w1<w0 then let w1=w1+1 endif
12          if w1>w0 then let w1=w1-1 endif
13          if w1<>w0 then high 0
14            else low 0 endif
15        else let w1=w0 endif
16        ;vždy funkce omezovače s indikací
17        if w1>440 or w1<160 then
18          high 0 else low 0 endif
19        let w1=w1 min 160 max 440
20        pulsout 4,w1
21        goto smycka
```

Fail safe

Častým požadavkem je zajistit definované chování serva v případě, že se ztratí řídící signál nebo opustí obvyklé meze. I když touto funkcí nazývanou failsafe jsou moderní digitální serva již vybavena od výrobce, připravíme program, který totéž zajistí pro jakékoli připojené

servo (program Servo9). Hlídá meze 0,8 až 2,2 ms a protože příkaz „pulsin“ při absenci pulzu vrací hodnotu 0, zareaguje a nastaví neutrál serva i v případě přerušení ovládání. Určitý rozdíl tu však je. Jestliže pulz přijde a je mimo rozsah, je servo ovládáno průběžně, při výpadku se však nevyhneme tomu, že servo dostává řídící pulzy jen každých 0,6 s, poté, co neúspěšně proběhne příkaz „pulsin“. Chování na mezi rozsahu také není ošetřeno hysterezí a může docházet ke kmitům, úprava v tomto smyslu je menší úlohou vhodnou k procvičení samostatné práce.

```
1       REM SERVO9 pro PICAXE 08M
2       REM fail safe 1
3       setfreq M8                      ;frekvence 8 MHz
4       smycka:
5         pulsin 3,1,w0                 ;čtení vstupu do w0
6         if w0>440 or w0<160 then
7           let w0=300 endif            ;jdi na stred
8         pulsout 4,w0                  ;vyslání pulzu
9         goto smycka                   ;zpět na začátek
```

Stejnou základní funkci plní i následující program Servo10 využívající příkaz „servo“, jeho chování je ale přece jen jiné. Tím, že výstup pulzů je zajištěn vnitřním časovačem, jsou pulzy pro servo generovány i v průběhu měření vstupu a k jejich delším výpadkům nedochází.

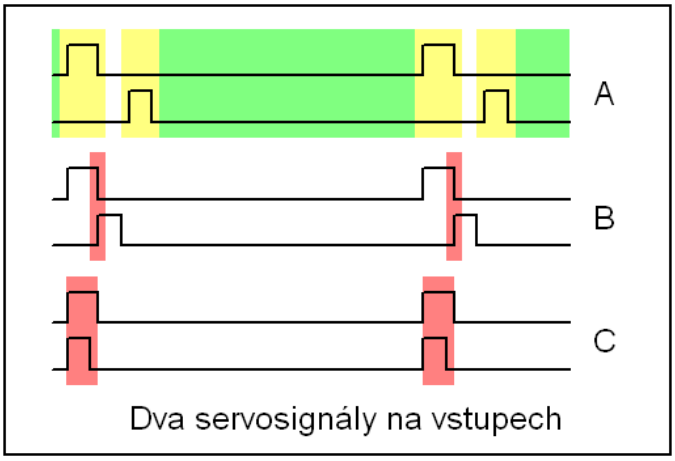
```
1       REM SERVO10 pro PICAXE 08M
2       REM fail safe 2
3       smycka:
4         pulsin 3,1,w0                 ;čtení vstupu do w0
5         if w0>220 or w0<80 then
6           let w0=150 endif            ;jdi na stred
7         servo 4,w0                    ;nastavení pulzů
8         goto smycka                   ;zpět na začátek
```

Reálně pracující fail safe by měl mít nejméně dvě, optimálně tři fáze, v první podrží poslední platnou hodnotu na vstupu a snaží se zamaskovat drobný výpadek signálu. V druhé fázi, v níž přejde do přednastavené polohy většinou blízké neutrálu, by se měl snažit zajistit let modelu, pokud tento má dostatečnou vlastní stabilitu (výjimku tvoří fail safe na ovládání motoru u auta, ten by měl hned stáhnout plyn). Třetí fáze se aktivuje až po delší době (desítky sekund), kdy už je velmi pravděpodobné, že k navázání spojení nedojde. V této fázi by měl být model uveden do řízeného pádu (například sestup s nataženou výškovkou), protože i model poškozený havárií a nalezený ve vzdálenosti „na dohled“ většinou znamená menší ztrátu, než model zcela ztracený. Napsání takového programu může být námětem pro další pokusy. Bez servosignálu na vstupu, bude průběh příkazu „pulsin“ trvat 0,65 s, to je také jednotka času, v níž budeme moci snadno nastavit dobu prodlevy.

Sekvencery pro serva

Obrátíme svou pozornost jinam, k problému čtení a řízení více serv. Příkaz pulsin může v jednom okamžiku číst jen jeden vstup a musí být aktivní už jistou dobu před jeho začátkem, po ukončení pulzu také potřebujeme určitou dobu přinejmenším na spuštění dalšího příkazu pulsin hlídajícího jiný vstup. Používáme-li běžnou modulaci PPM, jsou výstupy přijímače

aktivovány postupně, ale konec pulzu na daném kanále jde téměř současně se začátkem pulzu na kanále následujícím. Budeme-li využívat dva kanály, které nejsou sousední (nejlépe, jsou-li ob jeden kanál), může procesor PICAXE obsloužit dva, případně i tři vstupy, to je případ „A“ na obrázku. V ideálním případě (2 vstupy ob kanál) máme po načtení dvou vstupů ještě v každém cyklu asi 12 ms na vykonání programu a vyslání výstupních pulzů (20 = 2,5 + 2,5 + 2,5 + 12,5), takže lze realizovat například mixer se dvěma vstupy a dvěma výstupy.



Dva servosignály na vstupech

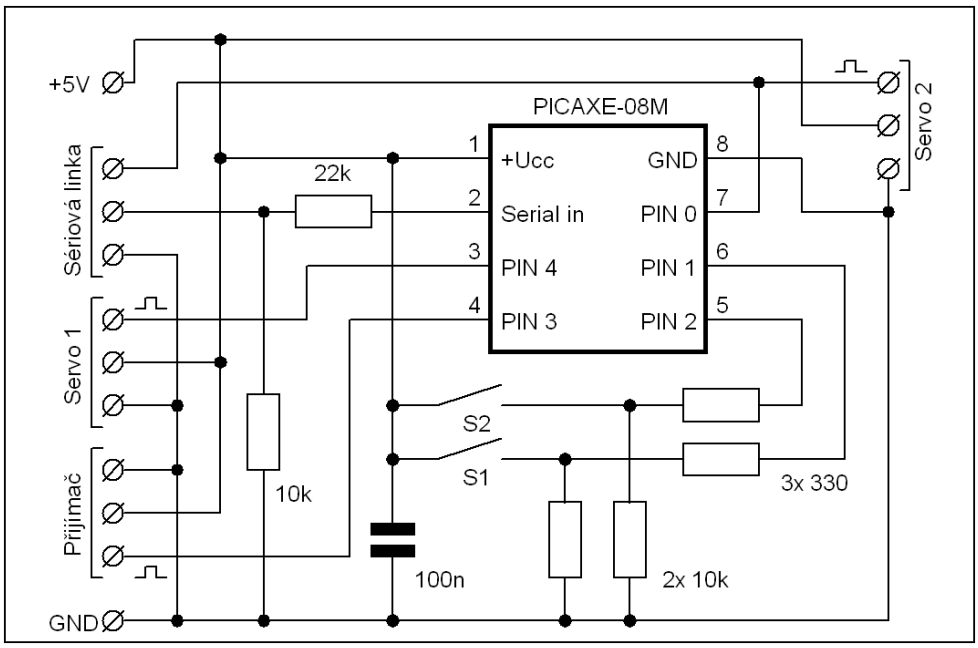
Případ „B“ zachycuje stejný PPM přijímač, ale dva sousední vstupy. Po skončení pulzu na jednom kanále nám náběžná hrana pulzu na následujícím kanále „uteče“ a budeme na ni muset čekat do dalšího cyklu, takže nepůjde číst všechny pulzy a ani obecně generovat dva výstupy tak, aby serva dostávala své ovládání pravidelně po 20 ms. V tomto případě lze využít dva vstupy k takovému účelu, který nevyžaduje sledování každého pulzu na vstupu a drobná prodleva nevadí, například jeden vstup povoluje generování zvuku (spínání) a druhý vstup (motor) určuje, jaký zvuk se bude generovat, vlastní vytvoření zvuku má ale na starosti nezávislý vnitřní časovač.

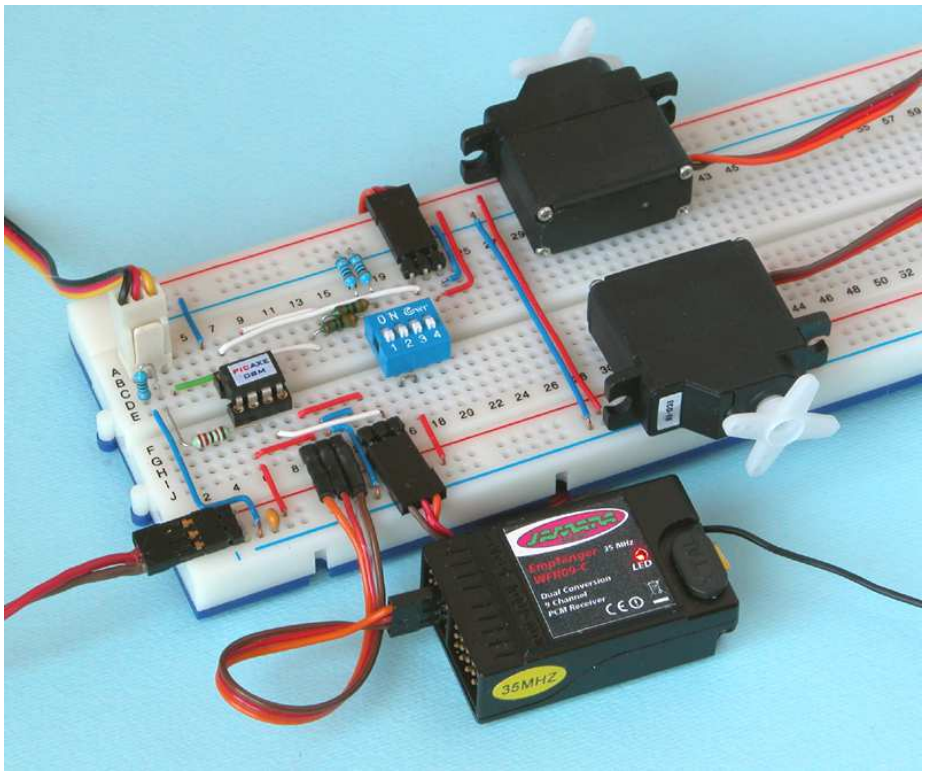
Případ „C“, kdy dva servopulzy se prakticky překrývají, nastává na některých výstupech přijímačů PCM nebo u přijímačů v pásmu 2,4 GHz. Důsledky jsou podobné jako v případě „B“, v každém cyklu více vstupů číst nemůžeme, jedině střídavě.

Nabízí se otázka, jak je možné, že vyráběná zařízení s případy „A“ ani „B“ problém většinou nemají a to ani na procesorech stejného typu, jaký tvoří základ PICAXE. Odpověď je jednoduchá, daní za názorné psaní programu v jazyce (Basic) je zpomalení chodu programů, takže leckdy to, co je na úrovni programu přímo v assembleru řešitelné, jazyk PICAXE nestíhá. Kromě toho zde používané algoritmy jsou ukázkové, funkční, ale mnohem jednodušší a méně zabezpečené než ty, které se najdeme v profesionálních výrobcích. Soustředíme-li se na zařízení s jedním vstupem, není problém obsloužit i procesorem PICAXE 08M až čtyři serva.

Sekvencer na jednoduchý povel ovládaný vypínačem z vysílače vykoná několika servy předem naprogramovaný pohyb. Typickým příkladem jsou víka podvozkových šachet a nohy podvozku. Předpokládejme, že v našem případě se nejprve musí otevřít šachta, tak vyklopit noha a opět šachta zavřít. Nebudeme se zabývat ošetřením toho, aby se víka a noha nemohly „potkat“, pokud by při zapnutí neodpovídala jejich poloha nastavení vysílače. Pro realistický vzhled bude pohyb víka (servo 1 na Pin 4) zpomalený stejně jako vyklopení nohy (servo 2 na Pin 0). Zpomalení půjde zapnout spínači S1 a S2. Zapojení mírně upravíme podle schématu,

má to však jedno úskalí. Je-li servo 2 připojeno během přenášení programu, nekoordinovaně sebou cuká, protože dostává na vstup přenášená data. Je lepší před přenosem programu servo odpojovat.





```
1       REM SEKVENCER1 pro PICAXE 08M
2       symbol mez0=140                 ;mez - zatáhnout
3       symbol mez1=160                 ;mez - vytáhnout
```

```
4     symbol mezmin=80                ;mez spodní
5     symbol mezmax=220               ;mez horní
6     symbol viko0=100                ;poloha víko zavřené
7     symbol viko1=200                ;poloha víko otevřené
8     symbol noha0=100                ;poloha noha zatažená
9     symbol noha1=200                ;poloha noha vytažená
10    symbol doba1=100                ;pauza víko/noha v 1/50 s
11    symbol doba2=100                ;pauza noha/víko v 1/50 s
12    REM b0 - pomocná pro cykly; b1 - poloha noha, b2 - poloha víko
13    REM b8 - stav podvozku; w2 - vstup pulzu
14    let b1=noha0 let b2=viko0 ;počátek - noha zatažená, víko zavřené
15    smycka:                         ;hlavní program
16      gosub krok                    ;čtení vstupu a generování výstupu
17      if w2>mez1 and b8=0 then gosub otevri
18      if w2<mez0 and b8=1 then gosub zavri
19      goto smycka                   ;konec hlavního programu
20    krok:                           ;cyklus čtení vstupu/synchronizace
21      pulsin 3,1,w2 ;čtení
22      if w2>mezmax or w2<mezmin then let w2=150 endif       ;ošetření mezí
23      pulsout 4,b1                  ;generování pulzu pro nohu
24      pulsout 0,b2                  ;generování pulzu pro víko
25      return
26    otevri:                                                 ;otevírání
27      if pin2=0 then let b2=viko1 else                      ;otevřít víko
28        for b2=viko0 to viko1 step 1 gosub krok next b2 endif
29      for b0=1 to doba1 gosub krok next b0                  ;počkat
30      if pin1=0 then let b1=noha1 else                      ;vytáhnout nohu
31        for b1=noha0 to noha1 step 1 gosub krok next b1 endif
32      for b0=1 to doba2 gosub krok next b0                  ;počkat
33      if pin2=0 then let b2=viko0 else                      ;zavřít víko
34        for b2 = viko1 to viko0 step -1 gosub krok next b2 endif
35      let b8=1                                              ;podvozek vysunut
36      return
37    zavri:                                                  ;zavírání
38      if pin2=0 then let b2=viko1 else                      ;otevřít víko
39        for b2=viko0 to viko1 step 1 gosub krok next b2 endif
40      for b0=1 to doba1 gosub krok next b0                  ;počkat
41      if pin1=0 then let b1= noha0 else                     ;zatáhnout nohu
42        for b1=noha1 to noha0 step -1 gosub krok next b1 endif
43      for b0=1 to doba2 gosub krok next b0                  ;počkat
44      if pin2=0 then let b2=viko0 else                      ;zavřít víko
45      for b2=viko1 to viko0 step -1 gosub krok next b2 endif
46      let b8=0                                              ;podvozek zasunut
47      return
```

Při vypnutých spínačích S1 a S2 běží pohyb serv plnou rychlostí, při zapnutých zpomaleně. Jednou spuštěné otevírání nebo zavírání nejde zastavit nebo zvrátit, děj vždy doběhne až do konce. Celý program je synchronizovaný pulzy z přijímače, které by měly přicházet každých 20 ms, je-li jejich perioda jiná, je třeba upravit počítání čekacích dob a také se změní rychlost pohybu serv. Podprogram „krok“ při každém volání přečte vstup z přijímače, počká na

servopulz a pak vygeneruje pulzy pro obě serva, tím se především stará o synchronizaci. Vždy se sice načtená hodnota ze vstupu uloží do proměnné w2, ale jen v případě, že je podprogram volán z hlavní smyčky programu, je tato hodnota dále skutečně využita a zpracována. Za zmínku stojí i to, že program Sekvencer1 prakticky plně využívá paměť procesoru PICAXE 08M (zbývá volných 20 bytů), tím zhruba ukazuje, jaké jsou limity využití tohoto procesoru.

Jednou z otázek, jež je v praxi často třeba řešit, je změna rychlosti pohybu serv. Pro jednoduchost se budeme snažit jen o rychlost poloviční nebo dvojnásobnou. V základní verzi chod zpomaleného serva odpovídá změně vstupních pulzů o 0,01 ms každou periodu servosignálu (20 ms), což při standardních hodnotách znamená přejezd za 2 s. Zrychlení serv je snadné, v podprogramu „otevri“ a „zavri“, tam kde jsou použity FOR cykly s určením kroku (STEP), stačí změnit krok 1 za 2 respektive –1 za –2.

Zpomalení spočívá v dvojnásobném volání podprogramu „krok“ těsně za sebou ve stejných podprogramech „otevri“ a „zavri“. Logika je jasná a jednoduchá, na jednu úpravu délky servopulzu připadne dvakrát delší doba, jenže narazíme na limit programu, jenž dovoluje použít příkaz GOSUB maximálně 16x. Tímto způsobem můžeme zpomalit pouze servo podvozku (volání 2x). Příklad použití se zrychleným pohybem šachet a zpomaleným pohybem nohy podvozku najdete v programu Sekvencer2 v balíčku souborů k tomuto dílu na internetových stránkách *www.rcrevue.cz* .

Všimněme si ještě proměnné b8, do níž se ukládá jen hodnota 0 nebo 1 vyjadřující stav podvozku. Proměnná zabírá 1 byte, ale vlastně by na uložení stejné informace stačilo 8x méně, jediný bit. Je-li takových proměnných v programu víc, stane se používání bytových proměnných k tomuto účelu přílišným luxusem a proměnné nám rychle dojdou. Na rozdíl od osobních počítačů, kde jsme zvyklí nad nějakým tím kilobytem, megabytem a v poslední době i gigabytem mávnou rukou, u jednočipových procesorů se často stává několik bytů volné paměti na program nebo uložení dat tím, co rozhoduje o úspěchu práce. Je na místě snažit se nejen o funkční programy, ale současně programy krátké, jenž účelně využívají paměť.

Ke stejnému prostoru vyhrazenému pro data můžeme přistupovat jako k 7 proměnným typu word (w0 – w6) nebo jako k 14 proměnným typu byte (b0 – b13), existuje ale ještě jeden způsob. Prostor proměnné w0 (respektive b0 a b1) můžeme rozkouskovat na jednotlivé bity a vytvořit z něj 16 jednobitových proměnných označovaných jako bit0 až bit15 *(*2)*.

Na téma šetření paměti navážeme malým příkladem na přemýšlení. Dejme tomu, že v nějakém programu využíváme dvě hodnoty, jsou uložené v proměnných b4 a b5. To, co potřebujeme, je tyto dvě hodnoty v proměnných vzájemně prohodit. Obvyklý postup využívá pomocné proměnné, řekněme b8 a vypadá asi takto:

```
let b8=b4
let b4=b5
let b5=b8
```

Když na začátku byla v b4 hodnota 10 a v b5 hodnota 25, tak po provedení těchto tří příkazů bude v b4 číslo 25 a v b5 číslo 10, to si může každý ověřit. Jak to ale udělat, když žádnou proměnnou, kterou bychom mohli použít jako pomocnou, nemáme? Zkuste vymyslet postup prohození obsahu dvou proměnných bez účasti jakékoli jiné proměnné. Řešení tohoto velmi jednoduchého fíglu si prozradíme příště.

V programu sekvenceru jsme zatím předpokládali, že všechny pohyby se ve skutečnosti vykonají tak, jak jsme chtěli, někdy je ovšem účelné se o tom raději přesvědčit. Zapojení zůstane stejné, jen ke spínačům budeme nyní přistupovat jinak. Představme si, že jak dveře

šachty, tak podvozková noha, jsou vybaveny koncovými spínači. Oba spínače na noze jsou spojené paralelně a oba spínače na dveřích také. Bude-li kterékoli servo v koncové poloze, ať už odpovídající otevření nebo zavření, bude na příslušném vstupu logická hodnota 1, bude-li někde uprostřed, hodnota 0. Do další fáze otevírání nebo zavírání teď postoupíme jen tehdy, jestli spínače signalizují, že předchozí fáze opravdu skončila. Při otevírání lze čekat tuto posloupnost:

	dveře	noha
Podvozek zasunut	1	1
Otevírání dveří	0	1
Dveře otevřeny	1	1
Vyklápění nohy	1	0
Noha vyklopena	1	1
Zavírání dveří	0	1
Podvozek vysunut	1	1

Z programu můžeme odstranit čekací cykly a možnost přepínání rychlosti pohybu serv, pro jednoduchost necháme jen plnou rychlost. Seznámíme se s dalším příkazem, který v základní příručce programátora, z níž obvykle vycházíme, není uveden, nicméně v anglickém popisu nebo české příručce speciálně pro procesory PICAXE 08M a 14M již obsažen je. V našem případě velmi pomůže.

DO ... LOOP

může být použit k vytvoření cyklu ukončeného splněním (nebo nesplněním) zadané podmínky dvěma základními způsoby, buď s podmínkou na začátku (podmínka se testuje před provedením příkazů a ty se tedy nemusí provést ani jednou) nebo na konci (podmínka se testuje až po provedení příkazů a ty se vždy provedou alespoň jednou). Zápis může vypadat takto

DO UNTIL podmínka příkazy LOOP

DO WHILE podmínka příkazy LOOP

nebo takto

DO příkazy LOOP UNTIL podmínka

DO příkazy LOOP WHILE podmínka

Cyklus je prováděn dokud je podmínka splněna (klíčové slovo WHILE), případně dokud není splněna (klíčové slovo UNTIL).

To co vypadá na první pohled složitě, nijak složité není. V našem případě budeme potřebovat volat podprogram „krok“, dokud se na Pin1 neobjeví hodnota logická 1 (čekáme dokud není servo v koncové poloze), což zapíšeme takto:

DO gosub krok LOOP UNTIL pin1=1

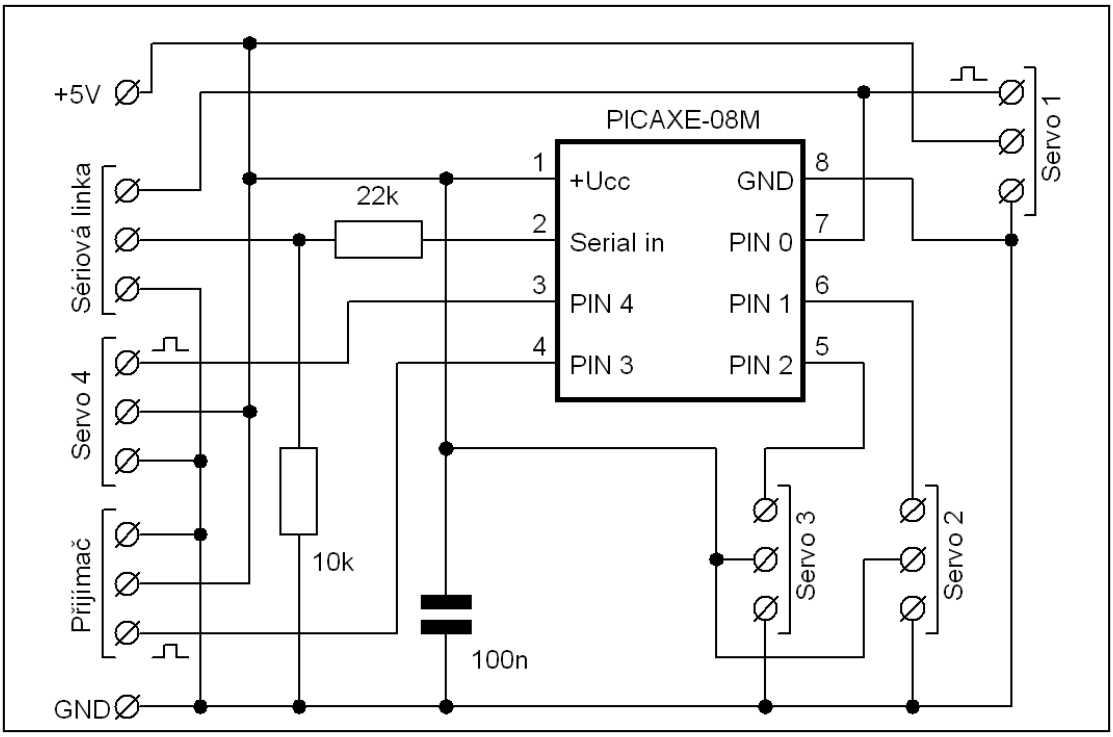
Náš program Sekvencer3 se po úpravě na využití koncových spínačů podstatně zkrátí.

```
1       REM SEKVENCER3 pro PICAXE 08M
2       symbol mez0=140                 ;mez - zatáhnout
```

```
3       symbol mez1=160                 ;mez - vytáhnout
4       symbol mezmin=80                ;mez spodní
5       symbol mezmax=220               ;mez horní
6       symbol viko0=100                ;poloha víko zavřené
7       symbol viko1=200                ;poloha víko otevřené
8       symbol noha0=100                ;poloha noha zatažená
9       symbol noha1=200                ;poloha noha vytažená
10      REM b1 - poloha noha, b2 - poloha víko
11      REM b8 - stav podvozku w2 - vstup pulzu
12      let b1=noha0 let b2=viko0       ;počátek - noha zatažená, víko zavřené
13      smycka:                         ;hlavní program
14        gosub krok                    ;čtení vstupu a generování výstupu
15        if w2>mez1 and b8=0 then gosub otevri
16        if w2<mez0 and b8=1 then gosub zavri
17        goto smycka                   ;konec hlavního programu
18      krok:                           ;cyklus čtení vstupu/synchronizace
19        pulsin 3,1,w2                 ;čtení
20        if w2>mezmax or w2<mezmin then let w2=150 endif       ;ošetření mezí
21        pulsout 4,b1                  ;generování pulzu pro nohu
22        pulsout 0,b2                  ;generování pulzu pro víko
23        return
24      otevri:                                 ;otevírání podvozku
25        let b2=viko1                          ;otevřít víko
26        do gosub krok loop while pin2=1       ;čekej dokud se nezačne otevírat
27        do gosub krok loop while pin2=0       ;čekej dokud se neotevře úplně
28        let b1=noha1                          ;vytáhnout nohu
29        do gosub krok loop while pin1=1       ;čekej dokud se nezačne vyklápět
30        do gosub krok loop while pin1=0       ;čekej dokud se nevyklopí úplně
31        let b2=viko0                          ;zavřít víko
32        do gosub krok loop while pin2=1       ;čekej dokud se nezačne zavírat
33        do gosub krok loop while pin2=0       ;čekej dokud se nezavře úplně
34        let b8=1                              ;podvozek vysunut
35        return
36      zavri:                                  ;zavírání podvozku
37        let b2=viko1                          ;otevřít víko
38        do gosub krok loop while pin2=1       ;čekej dokud se nezačne otevírat
39        do gosub krok loop while pin2=0       ;čekej dokud se neotevře úplně
40        let b1= noha0                         ;zatáhnout nohu
41        do gosub krok loop while pin1=1       ;čekej dokud se nezačne vyklápět
42        do gosub krok loop while pin1=0       ;čekej dokud se nevyklopí úplně
43        let b2=viko0                          ;zavřít víko
44        do gosub krok loop while pin2=1       ;čekej dokud se nezačne zavírat
45        do gosub krok loop while pin2=0       ;čekej dokud se nezavře úplně
46        let b8=0                              ;podvozek zasunut
47        return
```

Je poměrně dobře vidět, že podprogramy „otevri“ a „zavri“ se liší jen nepatrně, zkusme je tedy sloučit do jednoho s názvem „pohyb“ a tím program opět podstatně zkrátit a navíc si uvolnit počet volání GOSUB pro další případné úpravy. Výsledek pod názvem Sekvencer4

pro kontrolu najdete opět v balíčku souborů.



Poslední ze sekvencerů, který otestujeme, bude obsluhovat 4 serva. Ze zkušební desky odstraníme přepínač a příslušné rezistory, z Pin1 i Pin2 vyvedeme signály k novým konektorům pro serva. Nebudeme se pro začátek zabývat žádnými ochranami ani parametry vynesenými na začátek programu do konstant, úkol je jednoduchý, na povel z přijímače zamávat postupně všemi čtyřmi servy. Program Sekvencer5 tuto úlohu plní, zkusíme jej následně modifikovat tak, aby výchylky serv byly různé, aby některé ze serv běželo poloviční rychlostí nebo nejdřív všechna serva postupně přejela do krajní výchylky a pak se zase ve stejném pořadí vrátila zpátky.

```
1       REM SEKVENCER5 pro PICAXE 08M
2       let b0=100 let b1=100                          ;počáteční stav
3       let b2=100 let b3=100
4       smycka:                                        ;hlavní program
5         gosub krok                                   ;čtení vstupu a synchro
6         if w2>150 then
7           b0=200 gosub cekej b0=100 gosub cekej      ;obsluha serva 1
8           b1=200 gosub cekej b1=100 gosub cekej      ;obsluha serva 2
9           b2=200 gosub cekej b2=100 gosub cekej      ;obsluha serva 3
10          b3=200 gosub cekej b3=100 gosub cekej      ;obsluha serva 4
11          endif
12        goto smycka                                  ;konec hlavního programu
13      krok:                                          ;čtení vstupu/synchro
14        pulsin 3,1,w2                                ;čtení
15        pulsout 0,b0                                 ;pulz servo 1
16        pulsout 1,b1                                 ;pulz servo 2
17        pulsout 2,b2                                 ;pulz servo 3
```

```
18      pulsout 4,b3                          ;pulz servo 4
19      return
20    cekej:                                  ;čekání v 1/50 s
21      for b8=1 to 50 gosub krok next b8     ;(v periodě servosignálu)
22      return
```

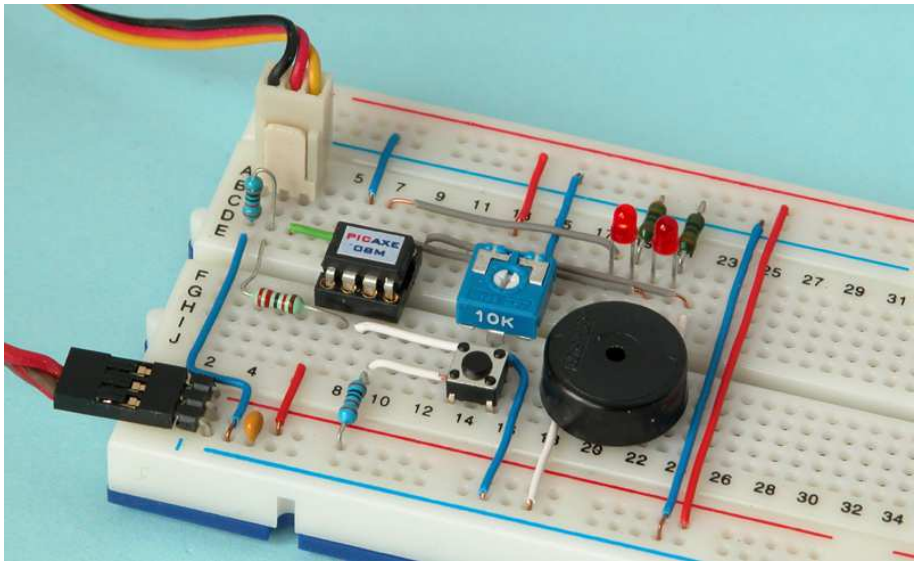
Práce s napětím

Procesory PICAXE jsou vybaveny AD převodníkem, takže mohou bez dalších součástek číst napětí na některých ze svých vstupů, 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 a nikoli s přesnou referencí a že není příliš přesný, používá se desetibitový převod spíše výjimečně a když už, tak především pro lepší rozlišení změn. Osmibitový převod poskytuje uspokojivé výsledky, vyhneme-li se oblastem těsně kolem nuly a plného napájecího napětí. Využijeme také příkaz Debug pro přenášení hodnot z procesoru programovacím kabelem do PC.

Vrátíme se k námětu na přemýšlení, jenž se týkal prohození obsahu proměnných b4 a b5. Řešení nevyžadující účast žádné další proměnné může vypadat třeba takto:

```
let b4=b4+b5
let b5=b4-b5
let b4=b4-b5
```

A teď již zpátky k tématu čtení, zpracování a využití analogového signálu (napětí) procesorem PICAXE 08M. 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 6 procesoru), převede na osmibitové číslo (0 - 255) a výsledek uloží do proměnné b0. U PICAXE 08M můžeme takto využít Piny 1, 2 a 4. (*20)

READADC10

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

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 ty se zobrazí v přehledné tabulce. Sériový přenos poměrně velkého množství dat výrazně zpomalí běh programu, takže se zpravidla nehodí pro odlaďování programů vyžadujících synchronizaci s vnějšími událostmi, navíc pin 0 už prakticky nemůže být používán k jiným účelům. (*6)

Zapojení na zkušební desce radikálně změníme, většinu odstraníme, ponecháme jen základní zapojení procesoru se sériovým přenosem (první schéma). Pro zkoušení převodu napětí využijeme trimr 10k ze sady součástek zapojený krajními vývody na napájení, jezdec spojíme s Pin 4, tak jak je to v levé části prvního schématu. Trimr má širší nožičky, takže nejde do desky zasunout přímo, buď musíme obrousit vývody ze stran asi na polovinu (což je ve výsledku mechanicky pevnější a celkově lepší, ale pracnější) nebo na ně připájet asi 1 cm dlouhé kousky drátu a ty do dutinek desky zasunout. Jezdec nastavíme přibližně do středu dráhy. Hned po spuštění programu by se mělo ukázat okno 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 hodnota mění. Přepojením výstupu trimru a úpravou programu vyzkoušíme totéž pro Pin2 a potom pro Pin1.

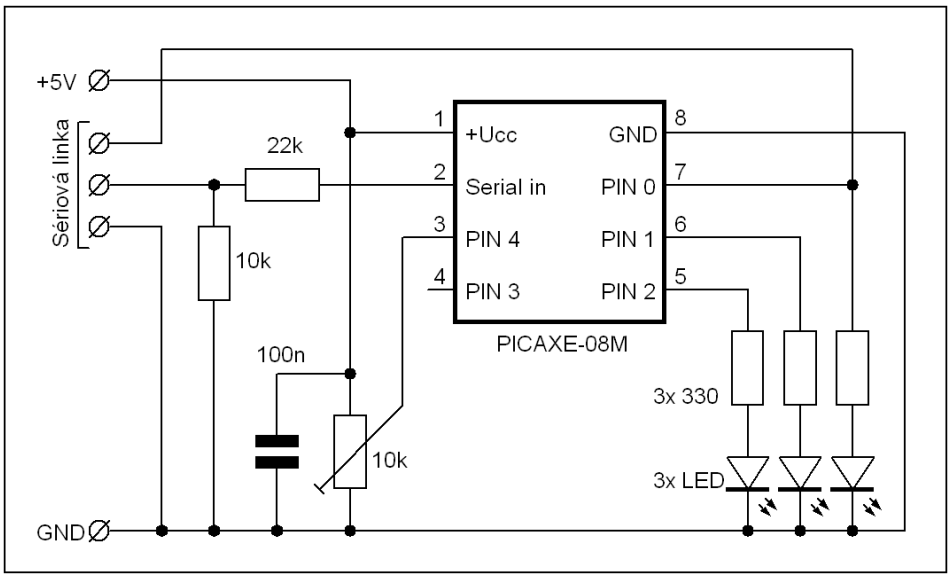
```
1       REM ANALOG1 pro PICAXE 08M
2       start:              ;smyčka programu
3         readadc 4,b0      ;převod z Pin4
4         debug             ;přenos do PC
5         goto start        ;zpět na začátek
```

Program modifikujeme pro desetibitový převod a opět vyzkoušíme. Při střední poloze trimu bude přečtená hodnota ve w0 (respektive v b0 a b1) čtyřikrát vyšší, kolem 512.

```
1       REM ANALOG2 pro PICAXE 08M
2       start:              ;smyčka programu
3         readadc10 4,w0    ;převod z Pin4
4         debug ;           přenos do PC
5         goto start        ;zpět na začátek
```

Pokusíme se naprogramovat jednoduchý voltmetr, který by vracel změřenou hodnotu v desetinách Voltu do počítače. Budeme předpokládat, že napájecí napětí procesoru je 5 V. Čtyřčlánek Nixx, který pravděpodobně k při pokusech 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 stejně jako v předchozích příkladech nejdřív vynásobením 10 a pak vydělením číslem 51.




```
1       REM ANALOG3 pro PICAXE 08M
2       REM jednoduchý voltmetr
3       start:                  ;smyčka programu
4         readadc 4,w0          ;převod AD z Pin4
5         let w0=w0*10          ;dělení 5,1
6         let w0=w0/51          ; ...
7         debug                 ;výsledek do PC
8         goto start            ;zpět na začátek
```

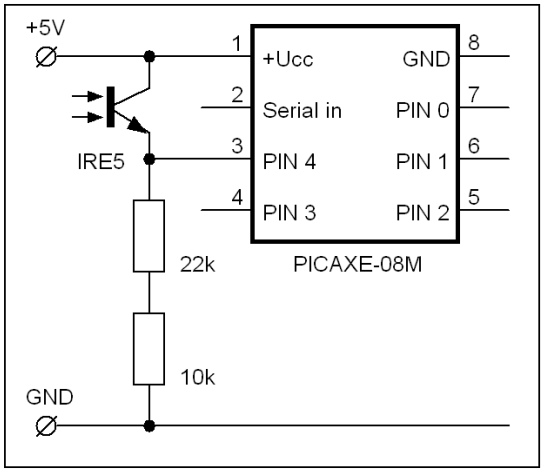
Zapojení doplníme třemi LED připojenými na Piny 0, 1 a 2 podle schématu. Připravíme program pro jednoduchý indikátor napětí, jakmile ovšem bude využívána LED na Pin 0, nemůžeme už přenášet data do PC. Indikátor bude hlídat napětí v intervalu zadaných mezí (svítí prostřední LED na Pin 1), při nižším rozsvítí LED na Pin 0 a samozřejmě při vyšším LED na Pin 2.

```
1       REM ANALOG4 pro PICAXE 08M
2       REM jednoduchý indikátor napětí
3       symbol mezd=40                  ;dolní mez
4       symbol mezh=44                  ;horní mez
5       start:                          ;smyčka programu
6         readadc 4,w0                  ;převod AD z Pin4
7         let w0=w0*10                  ;dělení 5,1
8         let w0=w0/51
9         low 0                         ;vypnutí všech LED
10        low 1
11        low 2
12        if w0<mezd then high 0 endif
13        if w0>mezh then high 2 endif
```

```
14      if w0>=mezd and w0<=mezh then
15      high 1 endif
16      goto start                              ;zpět na začátek
```

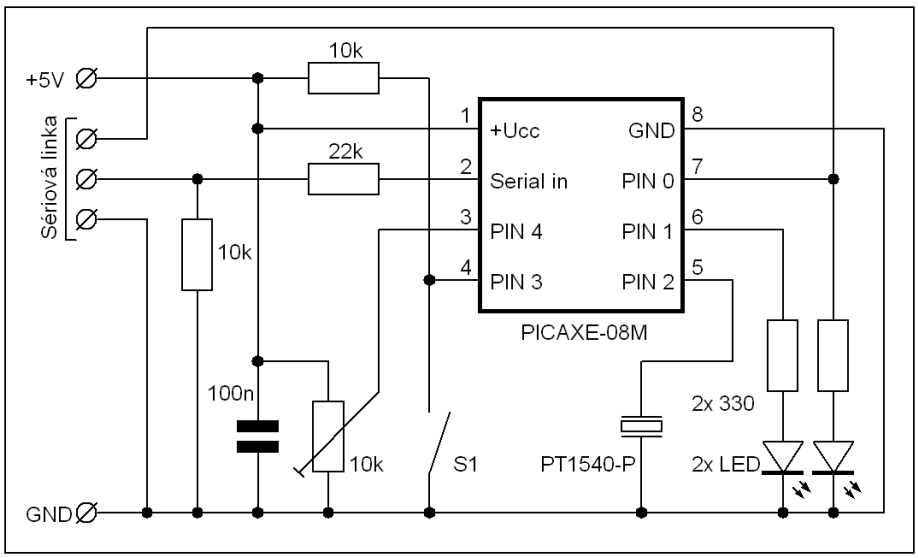
Tato základní funkce by šla realizovat jednodušeji a levněji konstrukcí s jedním IO dvojitého operačního zesilovače, když už ale využíváme mikroprocesor, proč indikaci nerozšířit ze tří na sedm stavů třeba takhle: při přechodech svítí vždy i obě LED současně, výrazně nízké napětí způsobí krátké záblesky na Pin 0, výrazně vysoké napětí dlouhé záblesky na Pin 2. Program pod názvem Analog5 najdete v balíčku souborů k tomuto dílu na internetových stránkách *www.rcrevue.cz* .

Analogové vstupy často nachází uplatnění ve spojení s jednoduchými čidly teploty, světla, nebo polohy. Jeden z takových příkladů si vyzkoušíme. Ze zkušební desky vyndáme trimr a na Pin 4 připojíme fototranzistor IRE5 dalšího schématu. Potřebný větší odpor získáme složením rezistorů 22 a 10 k. K otestování činnosti stačí v programu Analog4 nebo 5 změnit konstanty tak, aby se střed sledované oblasti napětí snížil do okolí 2,5 V. Vznikne tak indikátor osvětlení, který by měl reagovat na přiblížení k žárovce na vzdálenost 10 – 20 cm, případně na dopad přímého slunečního světla.



Zvuk

Než budeme pokračovat dalším větším příkladem, odskočíme si do oblasti generování zvuku. Vrátíme zpátky trimr, odebereme LED připojenou na Pin 2 (i její rezistor) a přímo mezi uvolněný vývod a zem zapojíme piezoměnič PT1540-P. K procesoru lze použít i malý reproduktor pokud možno s vyšší impedancí (nad 32 ohm), ten bychom však museli ještě stejnosměrně oddělit kondenzátorem. Další možností je vyrobit zesilovač s jedním FET tranzistorem, což dovolí nasadit podstatně větší a hlavně výkonnější reproduktory. Na dosud volný Pin 3 umístíme tlačítko. Generování zvuku usnadňují tři příkazy:



SOUND

vytváří sérii pípnutí, má proměnlivý počet parametrů. Prvním je pin, na nějž má výstup směřovat, pak následují dvojice bytů (nejméně jedna) s udáním výšky tónu a délky tónu v desítkách ms. Výška tónu 1 – 127 vytváří čistý zvuk, hodnoty 128 – 255 generují šum, 0 znamená pauzu. Tento příkaz se používá, je-li třeba upozornit obsluhu nebo jako součást zvukových efektů, není vhodný pro reprodukci hudby. Po dobu generování zvuku se procesor nemůže věnovat ničemu jinému. (*27)

TUNE

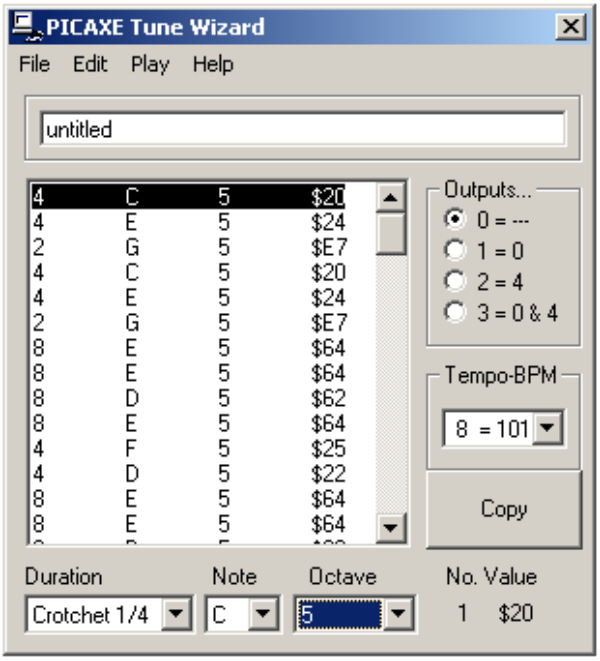
slouží ke generování hudby, na procesoru 08M je výstup pevně směrován na Pin 2, na jiných typech procesorů PICAXE lze výstup určit. První parametr udává chování LED v jistých typických konstrukcích používaných pro výuku, pro nás budou tyto výstupy vždy vypnuty hodnotou 0. Druhý parametr řídí tempo přehrávání skladby a následuje skladba v datech. K pořízení dat pro tento příkaz slouží samostatný program Tune Wizard. (*28)

Zmíněný program můžeme zavolat přímo z editoru (Menu – PICAXE – Wizards – Ring Tone Tunes). Jednohlasá melodie se do něj zadává v tónech (délka, nota, oktáva), 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 programu. Práce s tímto programem je jednoduchá, i když poměrně pracná. Pokud by bylo třeba použít v programu nějakou melodii, nemusíme ji nutně psát, lze si najít a vybrat nejméně z tisíce hotových skladeb. Na obrázku je pracovní okno s písničkou „Ovčáci, čtveráci“, na níž je dobře vidět, že hudba, lze-li tomu ovšem hudba vůbec vážně říkat, nezabírá moc místa, uvedená písnička má zhruba 40 byte.

PLAY

Procesor PICAXE 08M má přímo v sobě čtyři připravené melodie, jež lze volat zjednodušeným zápisem připomínajícím příkaz Tune. Prvním parametrem je číslo melodie (0 - 3), druhý určuje režim ostatních výstupu (pro nás hodnota 0). (*17)

Příklad pod názvem Zvuk1 má za účel pouze předvést, jaké melodie jsou v procesoru připraveny. Po stisku tlačítka se přehraje první, po dalším druhá atd., podobně jako to dělá melodický zvonek. Tyto melodie by šlo použít třeba jako oznámení konce nabíjení, jinak ale jsou pro naše účely prakticky bez významu. Zkusme pouze program modifikovat tak, aby výběr melodie po stisku tlačítka byl náhodný, respektive pseudonáhodný a závisel na době klidu před stiskem tlačítka. Výsledek je v souboru Zvuk2 v balíčku na internetových stránkách. Oba programy lze velmi dobře sledovat i v režimu simulace, jen si musíme uvědomit, že virtuální tlačítko funguje obráceně, tedy zapnutím (rozsvícením) se chod zastaví. V programu Zvuk2 je použit operátor pro zbytek po dělení //.




```
1       REM ZVUK1 pro PICAXE 08M
2       REM ukázky předvolených melodií
3       start:                          ;smyčka programu
4         if pin3=1 then goto start     ;čekání na stisk tlačítka
5         inc b0                        ;posunutí na další melodii
6         if b0>3 then let b0=0 endif   ;4 vyčerpány, znovu od nuly
7         play b0,0                     ;přehraj melodii
8         goto start                    ;hotovo, jdi na další
```

Jednoduché zvuky generované příkazem Sound nachází podstatně častější užití třeba jako kliknutí nebo pípnutí po stisku tlačítka, jako sirénka upozorňující na nežádoucí stav a podobně. Příklad Zvuk3 v sobě zahrnuje obojí a vrací se částečně k funkci indikátoru napětí, stisk tlačítka vyvolá kliknutí, vstupní napětí dané otočením trimru je ve střední poloze ukazováno LED, překročení mezí nahoru nebo dolů vyvolá výstrahu zvukovým efektem znějícím až do stisku tlačítka, i když příčina výstrahy již pominula.

```
1       REM ZVUK3 pro PICAXE 08M
2       REM indikátor napětí se zvukovou signalizací
3       REM tlačítko se testuje jen v hlavním programu
4       symbol mez1=10                  ;konstanty mezí
5       symbol mez2=20
6       symbol mez3=30
7       symbol mez4=40
8       REM b4,b5 - ovládání výstrahy, b2 pro cykly
9       start:
10        readadc 4,w0                  ;převod AD z Pin4
11        let w0=w0*10                  ;dělení 5,1
12        let w0=w0/51
```

```
13      if pin3=0 then sound 2,(110,2)                  ;kontrola TL
14      let b4=0 let b5=0 endif                         ;vypnutí signálu
15      cekej: if pin3=0 then goto cekej                ;čekání na puštění TL
16      low 0 low 1                                     ;zhasnutí obou LED
17      if w0>mez4 then let b5=1 endif                  ;překročena mez nahoru
18      if w0<mez1 then let b4=1 endif                  ;překročena mez dolů
19      if w0<mez2 then high 0 endif                    ;nižší stav - spodní LED
20      if w0>mez3 then high 1 endif                    ;vyšší stav - horní LED
21      if w0>=mez2 and w0<=mez3 then high 0            ;střed - obě LED
22      high 1 endif
23      if b4=1 then gosub malo                         ;volej signál nízký stav
24      if b5=1 then gosub moc                          ;volej signál vysoký stav
25      goto start                                      ;konec smyčky programu
26    malo:                                             ;sestupný tón + čekání
27      for b2=110 to 80 step -5 sound 2,(b2,5) next b2
28      pause 1200
29      return
30    moc:                                              ;vzestupný tón + čekání
31      for b2=80 to 110 step 5 sound 2,(b2,5) next b2
32      pause 1200
33      return
```

Má smysl všimnout si dvou věcí, jednak způsobu generování zvukových efektů pomocí FOR cyklu, jednak toho, jak program v praxi reaguje (respektive nereaguje) na stisk tlačítka. Je-li tlačítko stisknuto v situaci, kdy se jen měří a obsluhují LED, je odezva na stisk (pípnutí) prakticky okamžitá a vše je v pořádku. Jestliže je v činnosti zvuková signalizace, pak krátký stisk procesor většinou vůbec nezaregistruje, protože mu „uteče“ v době, kdy se věnuje právě generování zvuku a následnému čekání. Tuto nectnost lze do jisté míry odstranit tak, že místo příkazu Pause napíšeme vlastní čekací smyčku, v níž se bude tlačítko testovat a případně obsluhovat. Výsledek je mnohem lepší, ale přesto lze s trochou cviku stále stisknout tlačítko tak, aby to procesor nezjistil, stačí se strefit do doby generování tónu. Příslušný program pod názvem Zvuk4 je v balíčku.

V podobných situacích se s výhodou 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 %0000000<u>0</u>,%0000000<u>1</u>

Tento příkaz říká, že k přerušení dojde, když na vstupu Pin0 (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 Pin3 (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:

Setint %00001010,%00001111

Tento příkaz očekává na Pin0 hodnotu 0, současně na Pin1 hodnotu 1, současně na Pin2 hodnotu 0 a na Pin3 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 (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 (například mezi řádky) nebo dokonce v průběhu provádění příkazu, to se týká konkrétně příkazu Pause. Po skončení výkonu interruptu se program vrátí na místo, kde byl přerušen, a pokračuje v činnosti.

Nastavení příkazem Setint platí pro jedno volání interruptu, má-li být používán opakovaně, musíme jej vždy znovu „nahodit“ opětovným použitím Setint, třeba přímo na závěr 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.

Ponecháme zapojení tak jak je. Nejjednodušší příklad na funkci interruptu bude změna stavu LED na Pin0 po stisku tlačítka na Pin3, přerušovaným programem bude nekonečná smyčka.

```
1       REM INT1 pro PICAXE 08M
2       setint %00000000,%00001000
3       ;aktivuj interrupt když na Pin3 je 0
4       start:                  ;smyčka programu
5         pause 10              ;čekání
6         goto start            ;skok na začátek
7       interrupt:
8         toggle 0              ;změna LED
9         return                ;návrat z podprogramu
```

Po spuštění je LED zhasnutá, na výstupu je stav 0. Stisk tlačítka vyvolá interrupt a rozsvícení LED, další stisk už ale neudělá vůbec nic, protože tak jak je program napsán, se přerušení aktivuje a funguje jen jednou. Chceme-li opakovaně měnit stav LED, musíme přidat nebo přemístit příkaz Setint, tak jak je to v programu Int2 v balíčku. To ale nestačí. LED už sice reaguje opakovaně, ale během jednoho stisknutí tlačítka se vykoná podprogram vícekrát a záleží spíš na náhodě, v jakém stavu LED skončí. Je třeba zajistit, aby se změna provedla na jedno stisknutí jen jednou, například aby k ní došlo až po puštění tlačítka. Nakonec ještě přidáme zvuk „kliknutí“ a výsledek je v programu Int3. Čekání v příkazu Pause hodně prodloužíme, aby bylo možné jednoznačně vyzkoušet, že interrupt zafunguje i v jeho průběhu.

```
1       REM INT3 pro PICAXE 08M
2       start:                              ;smyčka programu
3         setint %00000000,%00001000
4         ;aktivuj interrupt když tlačítko sepne
5         pause 5000                        ;čekání 5s
6         goto start                        ;skok na začátek
7       interrupt:                          ;obsluha tlačítka
8         if pin3=0 then goto interrupt
9         ;čekej dokud není tlačítko puštěno
10        toggle 0                          ;změna LED
11        sound 2,(110,1)                   ;zvuk kliknutí
12        return                            ;návrat z podprogramu
```

Po úspěšném zvládnutí použití interruptu se vrátíme k programu Zvuk3 a upravíme voltmetr s indikací překročení mezí tak, aby vypínání signalizace tlačítkem bylo obsluhováno tímto novým způsobem. Interrupt „nahodíme“ příkazem Setint jen pokud je k tomu důvod (některá z mezí byla překročena), jinak tlačítko nereaguje. Výsledek je v programu Zvuk5. Dá se názorně vyzkoušet, jak se program vrací z obsluhy interruptu na místo, kde jej volání zastihlo. Stiskneme-li tlačítko v průběhu generování zvukového efektu, ozve se pípnutí, nastaví se proměnné tak, aby volání zvuku skončilo, ale poté se program vrátí a zvukový efekt je dokončen.

```
1       REM ZVUK5 pro PICAXE 08M
2       REM indikátor napětí se zvukovou signalizací
3       REM tlačítko se testuje interruptem
4       symbol mez1=10                      ;konstanty mezí
5       symbol mez2=20
6       symbol mez3=30
7       symbol mez4=40
8       REM b4,b5 - ovládání výstrahy, b2 pro cykly
9       start:
10        readadc 4,w0                      ;převod AD z Pin4
11        let w0=w0*10                      ;dělení 5,1
12        let w0=w0/51
13        low 0 low 1                       ;zhasnutí obou LED
14        if w0>mez4 then let b5=1 endif    ;překročena mez nahoru
15        if w0<mez1 then let b4=1 endif    ;překročena mez dolu
16        if w0<mez2 then high 0 endif      ;nižší stav - spodní LED
17        if w0>mez3 then high 1 endif      ;vyšší stav - horní LED
18        if w0>=mez2 and w0<=mez3 then high 0   ;střed - obě LED
19        high 1 endif
20        if b4=1 then gosub malo           ;volej signál nízký stav
21        if b5=1 then gosub moc            ;volej signál vysoký stav
22        goto start                        ;konec smyčky programu
23      ;konec hlavního programu, následují podprogramy
24      malo:                               ;sestupný tón + čekání
25        setint %00000000,%00001000        ;nastavení interruptu
26        for b2=110 to 80 step -5 sound 2,(b2,5) next b2
27        pause 1200                        ;čekání mezi signalizací
28        return                            ;návrat ze signalizace
```

```
29      moc:                                          ;vzestupný tón + čekání
30        setint %00000000,%00001000                  ;nastavení interruptu
31        for b2=80 to 110 step 5 sound 2,(b2,5) next b2
32        pause 1200                                  ;čekání mezi signalizací
33        return                                      ;návrat ze signalizace
34      interrupt:                                    ;obsluha interruptu (Tl)
35        if pin3=0 then goto interrupt               ;čekání na puštění Tl
36        sound 2,(120,8)                             ;pípnutí
37        let b4=0 let b5=0                           ;vypnutí signalizace
38        return                                      ;návrat z interruptu
```

Vyzkoušíme další modifikaci voltmetru (Zvuk6), LED budou sledovat jen překročení zadané meze nahoru a dolů a po stisku tlačítka se aktuální stav napětí bude signalizovat počtem pípnutí, desítky nižším tónem, jednotky vyšším. Vyjdeme z programu Zvuk3 a pro testování stisku tlačítka využijeme interrupt, aby byl jeho stisk kdykoli okamžitě zaznamenán. Za prohlédnutí stojí i podprogram „pipani“, který zajistí podle hodnoty z paměti w0 zvukovou signalizaci. Tento způsob výstupu čísla není sice tak pohodlný jako přečtení displeje, ale v aplikacích, kde se používá jen občas a prvořadým kritériem je obvodová jednoduchost a malé rozměry, se občas používá. Ostatně není nijak nový.

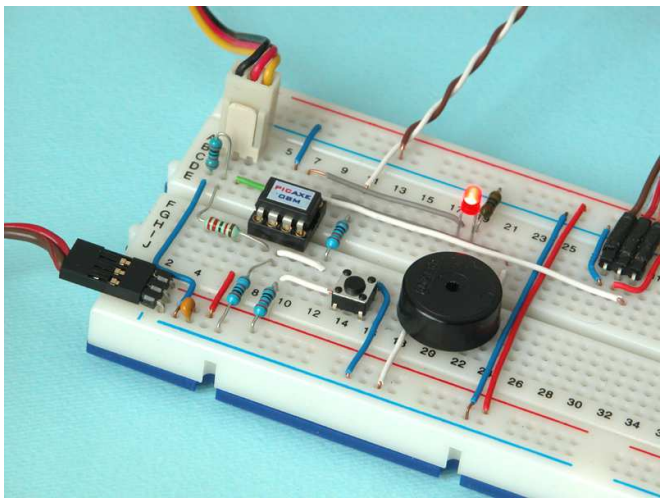
Bylo by logické se o signalizaci postarat přímo podprogramem interruptu, proč je to tedy tak, že interrupt nastaví jen příznak do proměnné b4 a okamžitě se vrací zpět? Interrupt může přijít kdykoli, i v době, kdy se obnovuje stav LED podle okamžitého napětí, a v takovém případě během celé signalizace mohou LED na poměrně dlouhou dobu zhasnout, což neodpovídá žádnému platnému stavu při zapnutí. Interrupt tedy jen zaznamená, že bylo tlačítko stisknuto, a výstup se zavolá s nepatrným zpožděním, když je stav LED jasně definovaný.

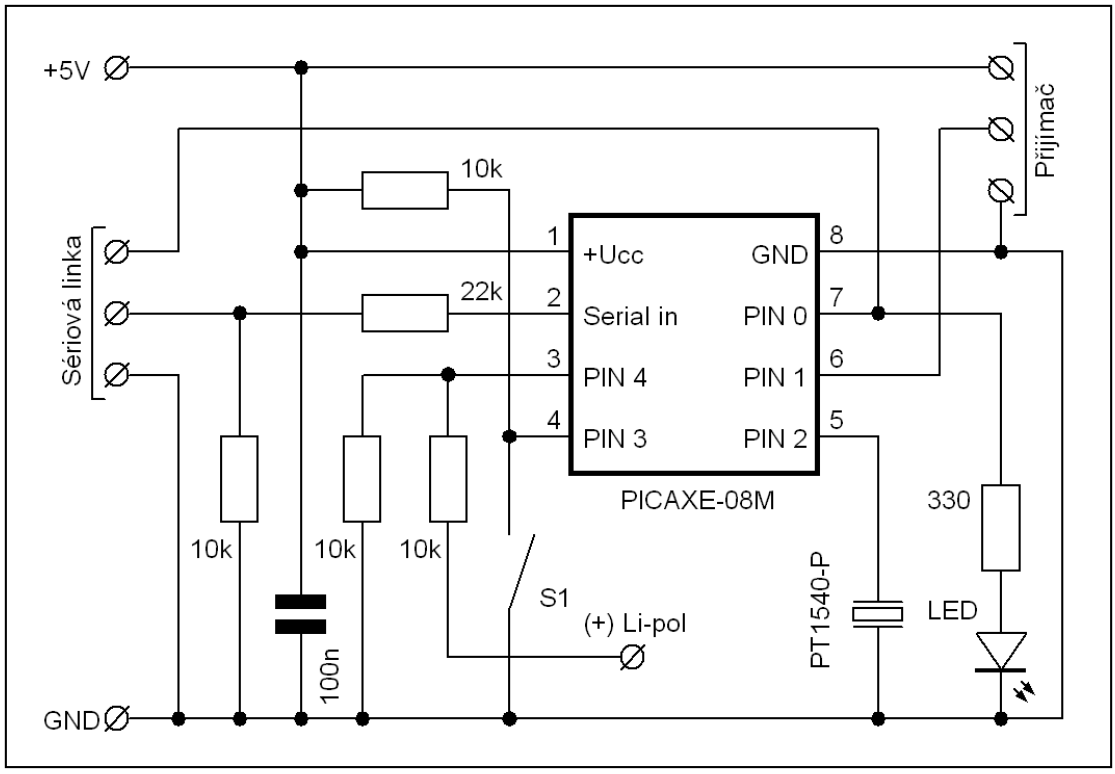
```
1       REM ZVUK6 pro PICAXE 08M
2       REM indikátor napětí s vypípáním hodnoty po stisku Tl
3       symbol mez1=20                                ;dolní mez 2,0V
4       symbol mez2=30                                ;horní mez 3,0V
5       let b4=0                                      ;vypnout pípání
6       REM b2,b3 na rozložení čísla, b4 indikace interruptu
7       start:
8         setint %00000000,%00001000                  ;Tl vyvolá interrupt
9         readadc 4,w0                                ;převod AD z Pin4
10        let w0=w0*10                                ;dělení 5,1
11        let w0=w0/51
12        low 0 low 1                                 ;zhasnutí obou LED
13        if w0>mez2 then high 1 endif                ;překročena horní mez
14        if w0<mez1 then high 0 endif                ;překročena dolní mez
15        if w0>=mez1 and w0<=mez2 then
16        high 0 high 1 endif                         ;střed - obě LED
17        if b4<>0 then gosub pipani
18        goto start                                  ;konec smyčky programu
19      interrupt:                                    ;obsluha tlačítka
20        let b4=1                                    ;nastavit příznak pípání
21        return                                      ;návrat z interruptu
```

```
22      pipani:
23        let b2=w0/10                    ;oddělit desítky
24        let b3=w0//10                   ;oddělit jednotky
25      desitky:                          ;signalizace desítek
26        if b2>0 then
27          sound 2,(80,5)                ;nižší tón
28          pause 300                           ;čekani mezi tóny
29          dec b2                        ;řízení cyklu
30          goto desitky
31          endif
32        pause 600                       ;čekání desítky/jednotky
33      jednotky:                         ;signalizace jednotek
34        if b3>0 then
35          sound 2,(120,5)               ;vyšší tón
36          pause 300                     ;čekání mezi tóny
37          dec b3                        ;řízení cyklu
38          goto jednotky
39          endif
40        let b4=0                        ;vypnout příznak pípání
41        return                          ;návrat z pipani
```

Další příbuznou úlohou může být záznamový modul pro elektrolet napájený dvěma články Li-pol (program Zaznam), který bude za letu sledovat minimum napětí pohonného akumulátoru a počet výpadků řídícího signálu, po přistání tyto hodnoty na požádání odsignalizuje zvukem. Pro zjednodušení předpokládejme, že modul je napájen z BEC a jeho napětí je stabilní a přesně 5V, ke kalibraci měření napětí slouží hodnota konstanty „koef“. V praxi se napětí mění s odběrem serv i teplotou, pro reálné použití bychom museli dát modulu samostatný stabilizátor a napájet jej přímo z akumulátoru.

Úprava zapojení bude minimální a je na schématu. Na Pin1 přivedeme vstup signálu z přijímače, ten nebude nic řídit, jen poslouží ke kontrole, zda přichází regulérní impulzy. I když to není zcela nezbytné, odstraníme také trimr a na vstup zapojíme pevný odporový dělič pro jednoduchost na poloviční napětí (takové rezistory máme v sadě), rozsah měření napětí je asi do 10 V. Vstup sledovaného napětí připojíme na (+) dvoučlánku Li-pol, jeho záporný pól musí být samozřejmě spojen se zemí našeho obvodu. Při prvním pokusu lze spojit vstup napětí s napájením procesoru, změřená hodnota by měla být kolem 4,9 až 5,1 V.





Při činnosti svítí LED polojasem (je pulzně spínaná), po stisku tlačítka zhasne a pustíme-li rychle tlačítko, oznámí modul napětí. Držíme-li tlačítko stále stisknuté a povolíme jej až po bliknutí, oznámí modul počet sekund od zapnutí, v nichž došlo k nějaké chybě řízení. Program je synchronizovaný na vstupní pulzy z přijímače, nicméně není nezbytné, aby pracoval zcela pravidelně, takže například po zjištěné chybě je pauza 1 s , kdy se ani neměří napětí. Za zmínku stojí test délky stisku tlačítka, ten je svou strukturou složitější a pokud by dělalo jeho pochopení problémy, doporučuji odmazat komentáře a rozdělit blok (if .. endif) zhuštěný do pěti řádek na jednotlivé příkazy.

```
1       REM ZAZNAM pro PICAXE 08M - záznam nízkého napětí a počítání chyb
2       REM b2,b3 na rozložení čísla, b4 tlačítko
3       symbol dolnimez=80 symbol hornimez=220     ;meze pulzu (střed 150)
```

```
4     symbol koef=255                                ;kalibrace napětí (/25,5)
5     symbol minimum=w5                              ;proměnná - minimum napětí
6     symbol chyby=b5                                ;proměnná - počet chyb (s)
7     symbol pulz=w4                                 ;proměnná - délka pulzu
8     high 0 pause 200 low 0 pause 1000              ;počáteční blik a čekání
9     let minimum=65535                              ;nastavení min napětí na max
10    smycka:                                        ;začáteh hlavní smyčky
11      readadc 4,w0                                 ;převod 8 bit AD z Pin4
12      high 0                                       ;rozsvítit LED - polojas
13      if w0<minimum then let minimum=w0 endif      ;registrace minima napětí
14      low 0                                        ;zhasnout LED - polojas
15      pulsin 1,1,pulz                              ;čtení z přijímače
16      if pulz<dolnimez or pulz>hornimez then
17        inc chyby pause 950 endif                  ;počítání chyb (jen sekundy)
18      if chyby>99 then dec chyby endif             ;maximum chyb je 99
19      if pin3=0 then do while pin3=0 inc b4        ;test délky stisku tlačítka
20      if b4>100 and b4<110 then high 0 else
21        low 0 endif if b4>250 then dec b4 endif    ;... na mezi bliknout
22      pause 10 loop                                ;... zpomalit testování
23        else let b4=0 endif                        ;... nebylo Tl - délka 0
24      if b4=0 then goto smycka                     ;nebylo stisknuto - zpět
25      if b4<100 then let w0=minimum                ;krátký stisk - do w0 napětí
26      let w0=w0*100                                ;převod na desetiny V
27      let w0=w0/koef
28      gosub pipani else let w0=chyby               ;dlouhý stisk - do w0 chyby
29      gosub pipani endif                           ;volat signalizaci
30      goto smycka                                  ;konec smyčky programu
31    pipani:                                        ;podprogram signalizace
32      let b2=w0/10                                 ;oddělit desítky
33      let b3=w0//10                                ;oddělit jednotky
34    desitky:                                       ;výstup desítek
35      if b2>0 then high 0 sound 2,(80,5) low 0     ;nízký tón s bliknutím
36      pause 300                                    ;čekani mezi tóny
37      dec b2 goto desitky endif                    ;řízení cyklu
38      pause 600                                    ;čekání desítky/jednotky
39    jednotky:                                      ;výstup jednotek
40      if b3>0 then high 0sound 2,(120,5) low 0     ;vyšší tón s bliknutím
41      pause 300                                    ;čekání mezi tóny
42      dec b3 goto jednotky endif                   ;řízení cyklu
43      return                                       ;návrat ze signalizace
```

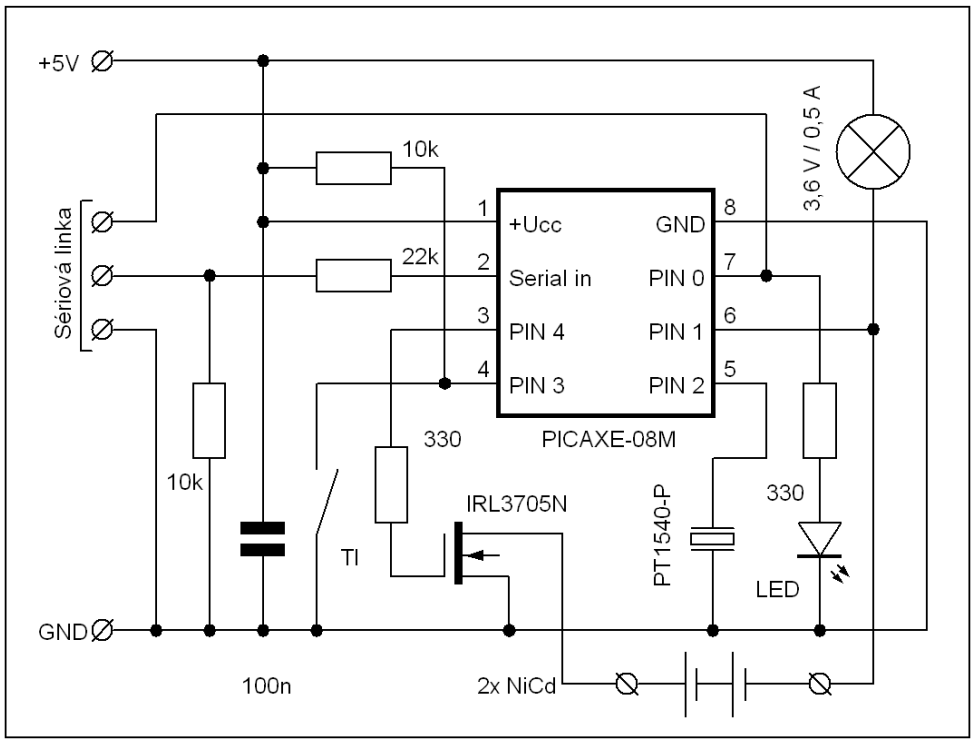
Nabíječ Nixx

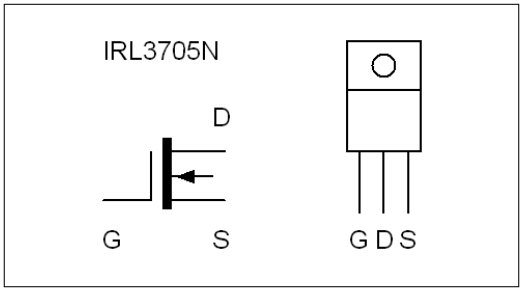
Nelze očekávat, že by si v dnešní době někdo stavěl a programoval vlastní nabíječ NiCd nebo NiMH, když existují levné a kvalitní jednoúčelové obvody, které umí totéž. Jako výukový příklad může jednoduchý nabíječ ovšem velmi dobře posloužit. K následující konstrukci budeme potřebovat součástku, jež v původní sadě nebyla, a to žárovku 3,6 V / 500 mA. Nabíječ je navržen pro dva články NiCd s malou kapacitou, takové, jaké napájejí volné elektrolety prodávané jako hračky. Při zkouškách byly použity akumulátory 180 mAh,

lze použít i běžné články velikosti AAA, v tom případě ovšem bude ještě potřeba opatřit na ně pouzdro. Přípravek lze stále napájet ze čtyřčlánku o kapacitě alespoň 800 mAh, musíme však častěji kontrolovat jeho nabití, budeme-li používat akumulátory AAA, vyplatí se spíše stabilizovaný zdroj 5 V dimenzovaný na trvalý odběr nejméně 0,5 A.

Důležité upozornění: Nabíjený akumulátor vždy připojujeme až k zapnutému přípravku a odpojíme jej před vypnutím napájecího napětí procesoru!

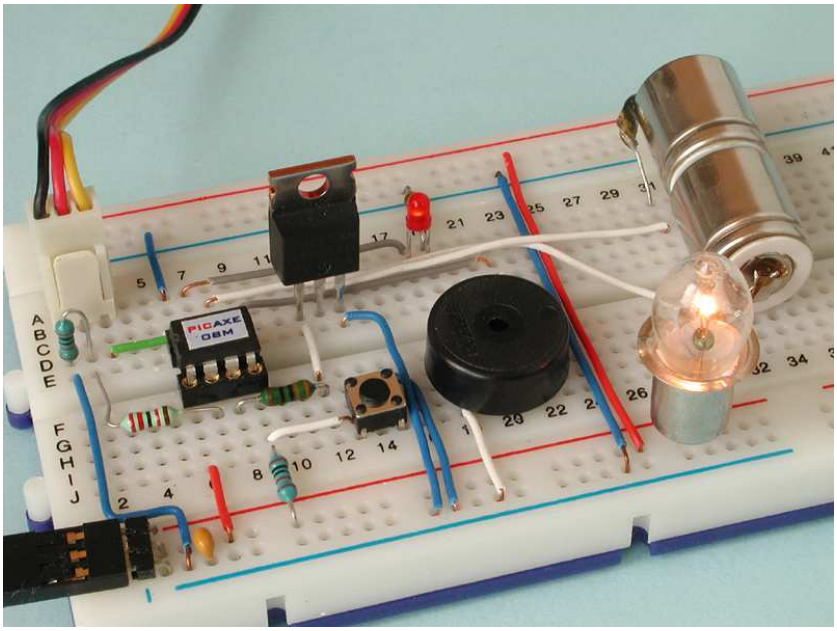
Zapojení využívá maximum z předchozích příkladů, pin 4 bude spínat výkonný FET tranzistor, zapojení vývodů je na nákresu. Kompletní katalogové údaje tranzistoru lze nalézt v balíčku dat k tomuto dílu na internetových stránkách *www.rcrevue.cz* . Před osazením tranzistoru do kontaktního pole je nezbytné pilníčkem nepatrně zúžit jeho vývody. Pin 1 snímá napětí na nabíjeném akumulátoru při sepnutém tranzistoru, úbytek na tranzistoru se neuvažuje a není pro vyhodnocení poklesu napětí v závěru nabíjení podstatný. Žárovka plní dvě funkce, jednak omezuje proud do nabíjeného akumulátoru (částečně funguje jako zdroj proudu 300 - 400 mA), jednak indikuje činnost nabíječe. LED zapojená na Pin 0 se k indikaci používá před začátkem nabíjení a po něm, v průběhu práce bliká při odesílání dat do PC.





V programu Nabijec1 začneme nejjednodušším nabíjením s detekcí poklesu napětí, pak budeme postupně přidávat pomocné funkce, kontroly a ochrany. Připomenu, že akumulátory, s nimiž budeme nabíječ zkoušet, by měly být funkční, ne nějaké vyteklé ze dna zásuvky. Budeme také pravidelně kontrolovat teplotu článků a při jejím výrazném vzrůstu pokus ukončíme.

Po zapnutí program čeká na stisk tlačítka, pak se rozsvítí žárovka a přes ni se akumulátory nabíjí. Nejdřív je nutné nechat určitý čas na proběhnutí dějů v počátku nabíjení, kdy se nebude napětí sledovat, nastavených 10 s je použitelné minimum. Protože měření AD převodníkem je zatíženo chybami, měří se v každém cyklu 10x a naměřené hodnoty se sčítají, maximum součtu je poznamenáno do proměnné. Jakmile naměřená hodnota klesne významně pod maximum, odpojí se nabíjecí proud, měnič zahraje jednu z uložených melodií a nabíječ se připraví na opětovné spuštění nabíjení. Při rozlišení 10 bitů na rozsahu 5 V odpovídá přibližně změna 1 bitu napětí 5 mV. Články jsou 2, měří se 10x, takže v programu nastavená mez poklesu napětí (60) odpovídá přibližně 60/20*5=15 mV/článek. Menší hodnota zpravidla vede k předčasnému (často náhodnému) ukončení nabíjení, při hodnotě větší než 100 se již články silně zahřívají.




```
1       REM NABIJEC1 pro PICAXE 08M
2       REM nabíječ NiCd 2,4 V, proud 300-400 mA, základní verze delta peaku
3       symbol prevod=w0                ;načítání desetibitového AD převodu
4       symbol maximum=w1               ;maximum součtu 10 převodů napětí
5       symbol soucet=w2                ;aktuální součet 10 převodů napětí
6       symbol pokles=b6                ;aktuální pokles součtu napětí
7       REM piny 0-LED, 1-napětí, 2-zvuk, 3-tlačítko, 4-nabíjení
8       start:
9         high 0                        ;rozsvícení LED - připravenost
10        if pin3<>0 then goto start    ;stisk tlačítka spustí nabíjení
11        high 4                        ;zapnout nabíjecí proud
12        low 0                         ;zhasnout LED - bude signalizovat přenos
13        pause 10000                   ;prodleva na náběh nabíjení 10s
14      nabijeni:
```

```
15      let soucet=0                        ;vynulování součtu napětí
16      for b7 = 1 to 10                    ;10 měření napětí
17      readadc10 1,prevod                  ;vlastní měření (1 dil = cca 5 mV)
18      let soucet=soucet+prevod            ;nasčítání hodnot
19      pause 90                            ;odstupy měření asi 0,1 s
20      next b7
21      if soucet>maximum then let maximum=soucet endif         ;nové maximum?
22      let pokles=maximum-soucet           ;pokles součtu aktuální
23      debug                               ;vyslání proměnných do PC (LED blikne)
24      if pokles<60 then goto nabijeni     ;rozdíl p<60 (cca 15 mV/čl) - dál
25      low 4                               ;vypnout nabíjecí proud
26      let maximum=0                       ;vynulovat maximum
27      play 3,0                            ;zahrát melodii
28      goto start                          ;připravit na další nabíjení
```

Uvedený program zabírá necelou čtvrtinu paměti procesoru, je tedy dost prostoru pro jeho rozšíření. Nabíječ by měl zvládnout alespoň v jednoduché formě i ochranné a doplňkové funkce. Nejprve začne nabíjet menším proudem ve třech stupních po 20 s, zkontroluje, zda odpovídá počet článků (napětí), začne nabíjet plným proudem a čeká na pokles napětí po dosažení nabitého stavu. LED svitem signalizuje funkci, žárovka plné pouze funkci rezistoru. Úspěšný konec nabíjení oznámí melodií a trvalým blikáním LED, přejde do udržovacího nabíjení proudem o průměrné velikosti asi tak 5 % původního. Je limitován čas nabíjení, v případě nesrovnalostí je hlášena chyba a akumulátor odpojen. Tlačítkem jde běh programu kdykoli (s využitím interruptu) ukončit.

Program Nabijec2 již nebudeme uvádět v plném rozsahu, je obsažen v balíčku dat na stránkách www.rcrevue.cz, doporučuji si jej však prohlédnout. Za zmínku stojí použití příkazu „setint 0,0“ , který vypne nastavení (nerealizovaného) interruptu, aby případný následující pokus o spuštění nabíjení bez připojeného akumulátoru neskončil zablokováním nabíječe. Celý program má 205 byte, stále ještě zůstává asi 20% kapacity paměti procesoru v rezervě. Závěrem znovu připomínám, že cílem tohoto příkladu nebylo vyrobit nabíječ konkurující komerčním výrobkům, ale vyzkoušet si zvládnutí problematiky, jež je známá většině modelářů.

Na posledním řádku programu Nabijec2 byl použit příkaz, jehož význam asi každý intuitivně chápe, ale jímž jsme se zatím nezabývali. Podívejme se na možnosti ukončení programu jinak než nekonečnou smyčkou.

END

způsobí kdykoli v průběhu programu jeho ukončení a přechod procesoru do stavu „spánku“ s nepatrnou spotřebou. Tento režim zastaví i činnost interního časovače (např. příkazu Servo). Obnova činnosti programu je možná jen vypnutím a zapnutím napájecího napětí, resetem nebo opětovným zavedením programu z PC.

STOP

funguje podobně, ale procesor nepřechází do režimu s minimální spotřebou, takže příkaz Servo probíhá dál

Spící procesor

U příkazu End jsme se zmínili o možnosti procesor „uspat“, k čemu je to ale dobré? PICAXE 08M odebírá v základním zapojení za chodu proud přibližně 1 mA, k tomu se přidává proud do zátěže (LED, piezoměnič, …). Někdy nám stačí, aby procesor něco vykonal jen občas a krátce, třeba jednou za několik sekund nebo minut, a mimo tuto dobu vlastně jen čeká. Čekání příkazem pause je ovšem pro procesor plnohodnotnou prací a jeho spotřeba se nemění, i když z našeho pohledu nedělá nic. Pokud procesor na dobu, kdy není potřeba jeho činnost, souvisle uspíme, klesne jeho spotřeba až na 0,1 mA. Rozdíl se může zdát malý, nicméně jde-li o činnost dlouhodobou a napájení z baterií, znamená to, že baterie vydrží 10x déle, a to už zanedbatelné není.

NAP

má jeden parametr v rozsahu 0 až 7, jímž se určuje doba přechodu do režimu s nízkou spotřebou. 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 ms. V průběhu uspání se interrupt nevyhodnocuje.

SLEEP

slouží stejným způsobem 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.

Je samozřejmé, že když se procesor umí ze spánku sám po předem dané době probudit, musí v něm něco běžet a počítat čas. Časovač, který toto zajišťuje, má omezenou přesnost, takže na uvedené doby se nelze úplně spolehnout, nicméně má-li procesor zareagovat třeba každých 10 sekund na podnět, který vyvolá interrupt, a jeho reakce je hotová do 0,1 s, lze klidně po provedení akce příkazem „sleep 4“ přejít do režimu spánku na 9,2 s, pak aktivně počkat na příchod interruptu, udělat co je třeba, a zase procesor uspat. Spotřeba se tím významně sníží.

V průběhu spánku nereaguje procesor ani na pokus o zavedení nového programu. Potřebujme-li program změnit, odpojíme napájení, spustíme v programu režim přenosu programu a vzápětí připojíme napájení. K přerušení a zapsání programu dojde po resetu procesoru (po zapnutí napájení) ještě před tím, než jej stihne stávajícím programem znovu uspat.

Program (Spanek) slouží k jednoduchému vyzkoušení uspání procesoru. Stiskneme-li za běhu krátce tlačítko, v naprosté většině případů se nestane nic. Pokud tlačítko podržíme, blikne LED jednou za 4,6 s, když se po cyklickém probuzení procesoru dostane ke slovu přerušení. Při pokusu o nové přenesení programu bude PC většinou viditelně čekat na probuzení procesoru.

```
1       REM SPANEK pro PICAXE 08M
2       start:
3         setint 0,8                ;interrupt na tlačítko
4         sleep 2                   ;uspání na 4,6 s
5         goto start                ;skok na začátek
```

```
6       interrupt:              ;výkon přerušení
7         high 0                ;bliknout LED
8         pause 100
9         low 0
10        return                ;návrat
```

Ukládání dat do EEPROM procesoru

Každý program, který jsme zatím použili, vždy při zapnutí napájení začínal znovu za stejných „startovních podmínek“. Cokoli, co program uložil do proměnných, bylo vypnutím napájení definitivně zapomenuto. To platilo i v případě, kdy by se vysloveně hodilo poznamenat získané údaje tak, aby byly dostupné i po vypnutí napájení. Typickým příkladem byl záznam nejnižšího napětí a počtu výpadků řízení za letu. Nemusí to tak být. PICAXE 08M dovoluje použít 256 bytů určených pro program současně také pro ukládání dat. Data se adresují „z druhé strany“ společného prostoru a musíme zajistit, aby nepřepsala program, ten by samozřejmě přestal fungovat. Budeme potřebovat nové příkazy.

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 programu. Druhý parametr tvoří data, která se mají na danou pozici uložit. Pracuje se tedy 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).

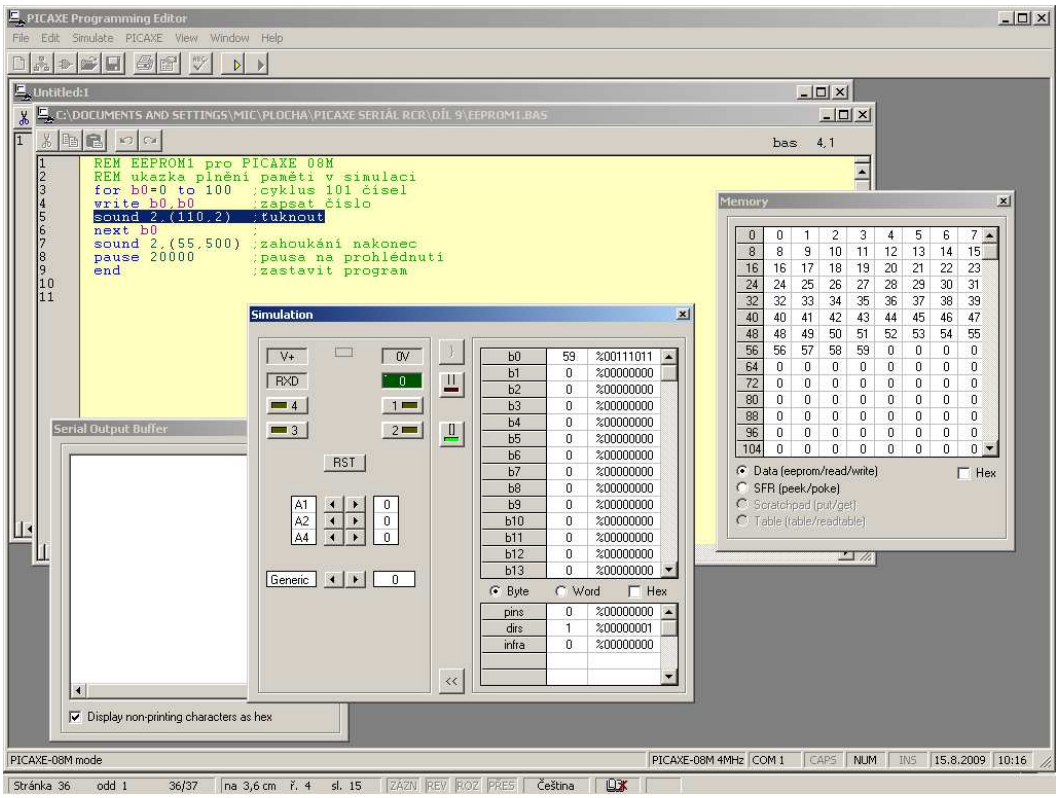
READ

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 velmi vysoký, ale přece jen omezený, takže tuto možnost budeme využívat z hlediska programu zřídka. Poznamenáme-li třeba vždy zapnutí zařízení, pak při 100000 přepisech EEPROM a použití 10x denně dospějeme k deklarované (minimální) životnosti paměti za 27 roků. To je myslím přijatelný výsledek a není důvod takto paměť nevyužít, stejně jako pro uložení parametrů regulátoru nebo testeru serv. Budeme-li bezhlavě zapisovat do EEPROM v cyklu programu, můžeme stejného limitu dosáhnout za několik málo hodin, to asi nebude rozumné. Počet čtení není rozhodující.

Program Eeprom1 se snaží zaplnit 101 bytů paměti čísly, poté vydá hlubší tón a zastaví se. Tento program spustíme nejprve s 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. Pausa na konci programu dává čas 20 s na prohlédnutí obsahu paměti, poté okna zmizí. Jezdcem můžeme posunout zobrazení paměti, dole v tabulce je vidět prostor zabraný programem (písmeno P místo hodnoty), program sám má 27 byte. Změníme-li horní mez cyklu z hodnoty 100 na 250, skončí běh chybou při pokusu přepsat program daty, simulace takovou situaci hlídá.

```
1       REM EEPROM1 pro PICAXE 08M
2       REM ukazka plnění paměti v simulaci
3       for b0=0 to 100                 ;cyklus 101 čísel
4       write b0,b0                     ;zapsat číslo
5       sound 2,(110,2)                 ;ťuknout
6        next b0 ;
7       sound 2,(55,500)                ;zahoukání nakonec
8       pause 20000                     ;pausa na prohlédnutí
9       end                             ;zastavit program
```



Využijeme stále stejné zapojení s tlačítkem, piezoměničem, a LED na kontaktní desce. Nastavíme horní mez cyklu na 100 a přeneseme program do procesoru. Ozve se chrčení složené z jednotlivých ťuknutí po zápisu jednoho bytu (chod procesoru je podstatně rychlejší než simulace) a nakonec souvislý hlubší tón oznamující, že zápis skončil. Nyní změníme horní mez cyklu na 250 a program zapíšeme do procesoru. Jde to, spustí se, ale v jednom okamžiku se program definitivně zastaví, souvislý tón už se neozve. Program byl narušen daty a procesor se „zakousl“, při odpojení napájení a opětovném zapnutí se ozve nejvýš jedno ťuknutí. Naštěstí na možnost nového naprogramování procesoru to vliv nemá.

Druhý jednoduchý demonstrační program Eeprom2 bude sledovat, kolikrát byl procesorem spuštěn, a po zapnutí vždy blikne LED a pípne tolikrát, jaké je pořadí spuštění (počítá se i první okamžitě po zavedení programu). Ukázka slouží jen pro demonstraci možností, počítání většího počtu pípnutí by bylo jistě nepohodlné a po 255 zapnutích by se počítadlo protočilo.

```
1       REM EEPROM2 pro PICAXE 08M
2       REM počítání spuštění programu
3       read 0,b0                       ;čtení z adresy 0
```

```
4       for b1=0 to b0                  ;pípnutí podle dat
5       high 0
6       sound 2,(120,10)
7       low 0
8       pause 200
9       next b1
10      inc b0                          ;zvýšení hodnoty o 1
11      write 0,b0                      ;zápis na adresu 0
12      REM tady pokračuje výkonný program
13      end
```

Přenosy sériových dat

Vrátíme se k programu Eeprom1 a pokusíme se zjistit, kam až se program daty přepsal. Zatím jsme k ladění programu využívali příkaz „debug“, ten vložíme hned za generování zvuku (sound) a program zavedeme do procesoru. Běh je kvůli přenášení všech hodnot proměnných příkazem debug hodně pomalý a musíme chvíli počkat, než se zastaví s hodnotou 236 v proměnné b0. Program měl 27 byte, 27+236=263, což je podstatně víc, než rozměr paměti (256 byte), takže se data „zakously“ do programu docela hluboko. Logicky se program zastavil v okamžiku, kdy byl přepsán příkaz „next“ nutný pro běh cyklu.

Procesor PICAXE je vybaven příkazy pro sériovou komunikaci, ať už s jiným procesorem, nebo s PC. Jeden z těchto příkazů se efektivně využívá k ladění programů, protože využívá zapojený kabel sloužící pro zavádění programů a posílá data v textové formě, takže je můžeme snadno číst prostřednictvím terminálu, který náš editor v PC obsahuje. Někdy bude dokonce nutné program zpomalit, 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 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.

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 zpravidla 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 (CR) vrátí psaní na první pozici v řádku, znak 10 (LF) odřádkuje, řádek tedy končíme sekvencí 13,10. Tento systém pochází ještě z dob dálnopisů a odpovídá i práci mechanického psacího stroje.

Program Eeprom3 v balíčku souborů k tomuto dílu seriálu na internetových stránkách *www.rcrevue.cz* obsahuje kontrolní „tisk“ respektive odeslání hodnoty proměnné b0. Program přeneseme do procesoru a jakmile se začne ozývat „tikání“, uzavřeme okno s hlášením a stiskem klávesy F8 (nebo přes menu - PICAXE - terminal) zavoláme sériový terminál, v němž se zobrazují přijatá data, každé hlášení v samostatné řádce. V příkazu **`sertxd("b0 = ",#b0,13,10)`** se prvním parametrem vypíše „b0 = “, pak hodnota proměnné b0 a nakonec

se dvěma znaky (13 a 10) odřádkuje. Zkusíme si, jak vypadá výsledek bez těchto posledních dvou znaků.

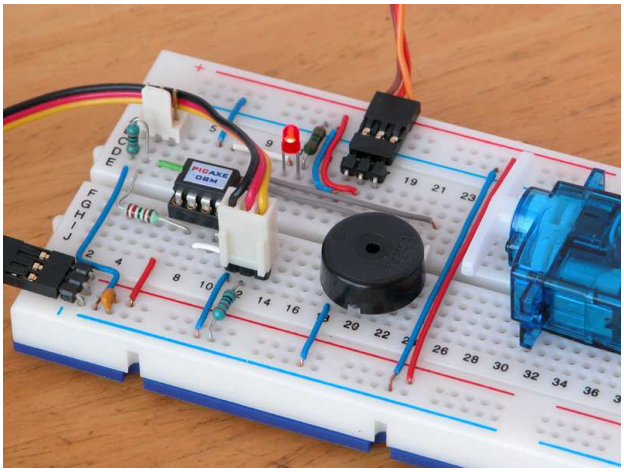
SEROUT

je obecnějším příkazem pro odesílání sériových dat. Jeho prvním parametrem je číslo pinu, na který 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, na PICAXE 08M ne. Všechny rychlosti se týkají hodinového kmitočtu 4 MHz.

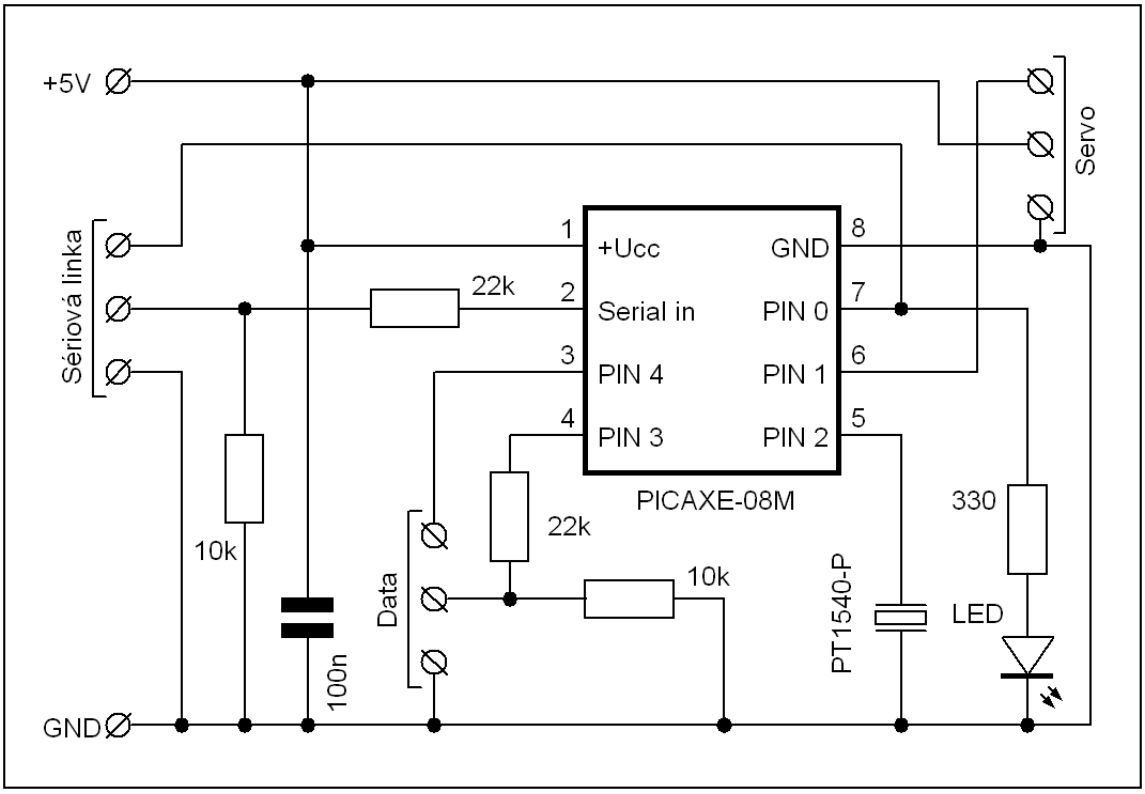
Příkaz SERTXD nahradíme v programu Eeprom3 příkazem (serout 0,N300,("b0 = ",#b0,13,10)) , ten bude využívat stále stejný kabel (s výstupem na pinu 0), ale přepne rychlost přenosu na 300 Bd. Program zavedeme do procesoru, jakmile začne tikat, spustíme terminál a zkontrolujeme, že přenos probíhá dobře, i když samozřejmě mnohem pomaleji, než při rychlosti 4800 Bd. Pokud při těchto pokusech nejde zavést program do procesoru, zkusíme to jednoduše stejně znovu a znovu, komunikace se někdy nestihne „chytit“ včas a editor usoudí, že jsou přerušené vodiče nebo vypnuté napájení.

Přenos z PC do procesoru naráží na problém. Jakmile přijdou nějaká data po standardním kabelu na vývod Serial IN, spustí se vnitřní přerušení procesoru a ten se je snaží uložit jako nový program. Chceme-li posílat svému programu svoje data, musíme to uskutečnit přes jiný pin. Postavíme si tedy zapojení s dalším konektorem pro vstup a výstup dat, použijeme stejný kabel, ale vždy jej přepojíme buď na přenos programu, nebo na přenos dat. K procesoru připojíme jedno servo, LED a zvukový měnič ponecháme.

První program, Prenos1, bude mít za úkol jen prověřit přenos sériových dat tím, že tak jak je bude přijímat bude byte po bytu zase vracet zpátky terminálu. Se sériovým kabelem zasunutým do konektoru pro přenos programů zavedeme program, pak přesadíme kabel do datového konektoru a klávesou F8 spustíme terminál. Zapíšeme znak (nebo více znaků, můžeme k odřádkování používat Enter) a tlačítkem Send je odešleme. Procesor by je měl opakovat beze změny zpět, výpis bude samozřejmě pomalejší, brzdí jej komunikace. Před případnými úpravami nebo zavedením dalšího programu nesmíme zapomenout zase kabel přehodit zpět na „programový“ konektor.



```
1       REM Prenos1 pro PICAXE 08M
2       REM prověření sériového přenosu
3       REM oběma směry na 2400 Bd
4       start:
5       serin 3,N2400,b0
6       serout 4,N2400,(b0)
7       goto start
```

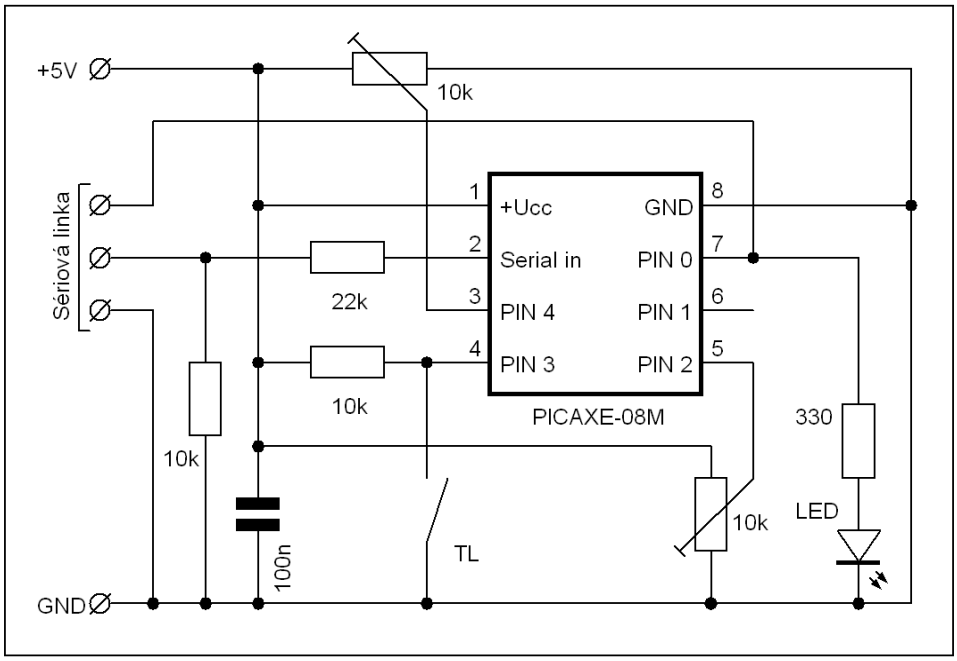


Následující program Prenos2 má předvést možnosti, ale také limity ovládání činnosti procesoru z PC. Jednotlivé akce budeme volit odesíláním číslic, jejich význam je v tabulce. Posíláme vždy jen jednu číslici a další až poté, co procesor svou práci dokončí a ohlásí to zpět. Pokud se pokusíme poslat více znaků, bude procesor akceptovat jen první z nich a další mu mezitím "utečou".

1	Zahraj melodii č. 1
2	Zahraj melodii č. 2
3	Zahraj melodii č. 3
4	Servo do jedné krajní výchylky
5	Servo na střed
6	Servo do druhé krajní výchylky
7	Změň stav LED (rozsviť / zhasni)

```
1       REM Prenos2 pro PICAXE 08M - ovládání LED, zvuku a serva z PC
2       let b1=150                                  ;nastavení serva na střed
3       start:                                      ;hlavní smyčka programu
4         serin 3,N2400,b0                          ;čtení linky do b0, pin3/2400Bd/Neg
5         if b0="1" then play 1,0 endif             ;přehrání melodie 1
6         if b0="2" then play 2,0 endif             ;přehrání melodie 2
7         if b0="3" then play 3,0 endif             ;přehrání melodie 3
8         if b0="4" then let b1=100 endif           ;nastavení serva do 1 krajní polohy
9         if b0="5" then let b1=150 endif           ;nastavení serva na střed
10        if b0="6" then let b1=200 endif           ;nastavení serva do 2 krajní polohy
11        if b0="7" then toggle 0 endif             ;změna stavu LED
12        servo 1,b1                                ;pohyb serva na danou polohu
13        pause 1000                                ;čas pro pohyb serva
14        if b0>"0" and b0<"8" then
15          serout 4,N2400,("Akce ",b0," ukončena",13,10)     ;provedeno
16          else
17          serout 4,N2400,("Akce ",b0," neznámá",13,10)      ;hlášení - jiný kód
18          endif
19        goto start                                ;zpět na začátek
```

Na činnosti tohoto programu je záměrně ukázáno důležité omezení. Normálně stačí zadat příkaz servo, a procesor pravidelnými pulzy sám udržuje servo v dané poloze, program může běžet dál. Tady to najednou neplatí, jakmile skončí pausa, servo se uvolní. Je to proto, že jak sériový přenos tak příkaz servo využívají je své činnosti stejné prostředky procesoru, jakmile se tedy začne vykonávat příkaz pro sériový přenos, pulzy na servo přestanou chodit. Pokud bychom chtěli vytvořit průběžné řízení serva ovládaného z PC, museli bychom například posílat povely každých 20 ms a pulzy generovat přes „pulsout“, tedy synchronizovat celou činnost z PC.



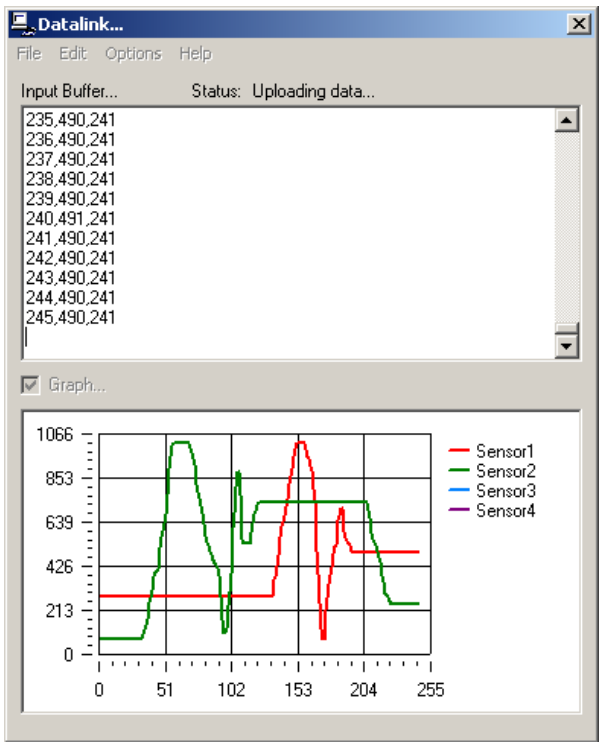
K čemu je to všechno dobré? Kromě výhodného využití zjednodušeného sériového přenosu při ladění programů existuje řada úloh, kterou je problematické řešit jen jedním procesorem, protože ten zkrátka nestihne hlídat vstupy a současně vykonávat to, co má. Jakmile máme dva procesory, musí se spolu nějak domluvit. Propojení sériovou linkou (obvykle jednosměrnou) je jednoduché a často používané řešení. Podobným případem je i napojení procesoru LCD displej kvůli zobrazení dat nebo na GPS a vytvoření jednoduchého systému řízení modelu. V neposlední řadě se sériový přenos do PC (opět zpravidla jednosměrný) využije, když je třeba procesorem získaná data nějak zobrazit, a zařízení samo nemá žádný displej, přece jen „odpípání“ hodnot, které jsme použili u indikátoru napětí, není nejpohodlnější. Editor má pro tyto případy dokonce připravenou pomůcku.

Nejprve modifikujeme zapojení, odebereme zvukový měnič, konektor pro servo i pro přenos dat a přidáme tlačítko a dva trimry, pro začátek je nastavíme přibližně do poloviny. Program Graf1 nejprve poskytne čas na nutné nastavení programu, pak generuje „pilový“ signál a odesílá data do PC, stiskem tlačítka se ukončí. Přeneseme program do procesoru a během jeho čekání v editoru klávesou F9 (nebo přes menu - PICAXE - Datalink) spustíme Datalink. Ve volbách zkontrolujeme, že přenos je nastaven na 4800 Bd, počet senzorů bude 1, zapneme volbu „Send G“ a zatrhneme políčko „Graph“. Dále musíme do menu - file - new a odmáčkneme nabídnuté volby. V grafu se začne zobrazovat „pila“ rychlostí omezenou jen přenosem, můžeme pozorovat změnu měřítka s překreslování grafu. Posílaná data musí mít formát: souřadnice x, data z prvního (druhého, třetího, čtvrtého) čidla, vše odděleno čárkami a ukončeno znaky CR a LF (13, 10). Všechny hodnoty mohou být i z intervalu word (0 - 65535). Po zastavení programu v procesoru tlačítkem Datalink po chvilce vypadne kvůli příliš dlouhé prodlevě v datech.

```
1       REM Graf1 pro PICAXE 08M
2       REM generování grafu - pily
3       pause 8000
4       start:
5         inc w0
6         sertxd (#w0,",",#b0,13,10)
7         if pin3=1 then goto start
8         end
```

Druhý program Graf2 sleduje a zobrazuje napětí na dvou trimrech, jimiž si můžeme „nakreslit“ ukázkový průběh, jen je třeba předem nastavit počet čidel na 2. Tato pomůcka může posloužit opět pro ladění programů nebo pro reálné výstupy měření, jak je potřeba. Námětem pro samostatné vyzkoušení je vrátit se k úloze nabíječe Nixx a doplnit jeho program tak, aby posílal hodnoty napětí akumulátorů do PC a ty se zobrazovaly v grafu.

```
1       REM Graf2 pro PICAXE 08M - graf dvou ručně ovládaných napětí
2       pause 8000                              ;prodleva na inicializaci
3       start:                                  ;začátek smyčky programu
4         inc w0                                ;přírůstek osy x
5         readadc10 4,w1                        ;měření napětí na pin 4
6         readadc10 2,w2                        ;měření napětí na pin 2
7         sertxd (#w0,",",#w1,",",#w2,13,10)    ;odeslání dat do PC
8         pause 100                             ;zpomalení, prodleva 0,1s
9         if pin3=1 then goto start             ;test tlačítka - konec
10        end
```



Řízení motoru

Jednou z úloh, kterou je třeba dost často procesorem řešit, je plynulé ovládání otáček elektromotoru. Budeme uvažovat jen motory stejnosměrné (komutátorové), řízení krokových a střídavých motorů lze najít v aplikační příručce. Pro ovládání otáček (výkonu) motorů se používá pulzně šířková modulace jejich napájecího napětí, to znamená, že na motor je cyklicky připojováno napájení střídané s vypnutím a právě poměrem doby, kdy je motor pod proudem a odpojený, se řídí jeho chod. K tomu slouží dva příkazy.

PWM

Příručka uvádí, že příkaz má tři parametry oddělené čárkou, první určuje, na který z pinů má být příkaz směrován, druhá je konstanta činitele plnění v rozsahu 0 - 255, to je to, čím regulujeme výkon, a konečně třetí parametr (0 - 255) udává počet cyklů, které má procesor vykonat.

Příkaz neběží na pozadí, v jeho průběhu procesor nedělá nic jiného. Probíhá jednorázově, je-li třeba řídit motor průběžně, musí se volat pravidelně. Podléhá samozřejmě nastavení hodinové frekvence procesoru. S parametry je to ve skutečnosti složitější, příkaz negeneruje pulzy s konstantní periodou a proměnným plněním, ale výrazně se mění i perioda. Stejně tak počet cyklů neodpovídá, je jich podstatně více. Budeme-li používat parametry v tom smyslu jak jsou uvedeny, motor se jimi řídit úspěšně dá, ale přesný význam neodpovídá. Tento příkaz se pojí jen s procesory PICAXE 08 (08M).

PWMOUT

je obecnější než předchozí. Má také tři parametry oddělené čárkou, první určuje pin, druhý jednobytový parametr udává periodu pulzů a třetí v rozsahu 0 - 1023 dle návodu odpovídá

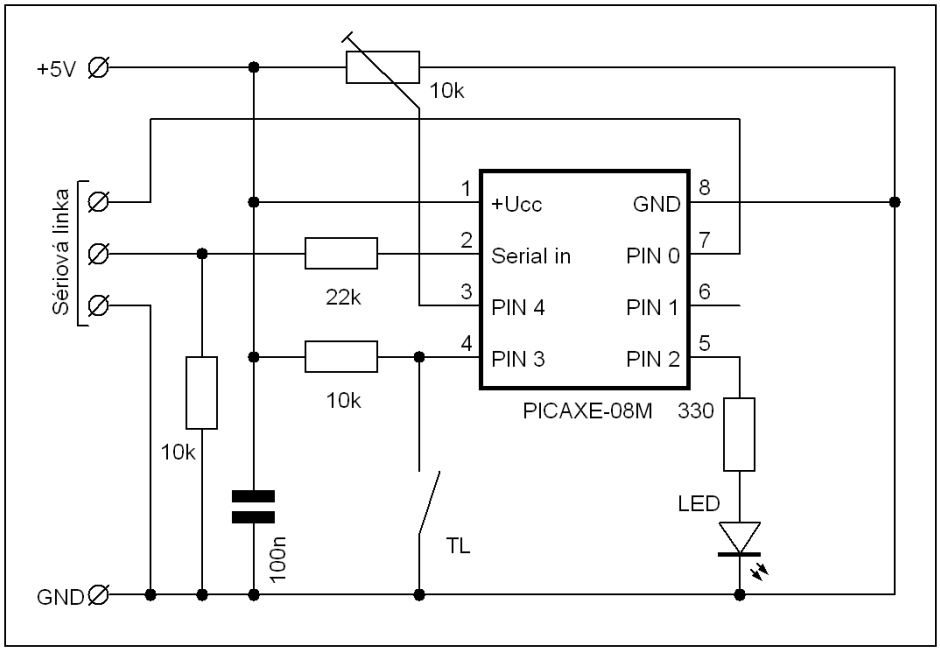
činiteli plnění.

Protože jsou pulzy tentokrát generované časovačem, dá se u procesoru PICAXE 08(08M) použít pouze pin 2, pulzy ale běží na pozadí, procesor může během řízení motoru vykonávat jiné činnosti. Příkaz pwmout nelze použít současně s příkazem Servo, oba využívají stejné prostředky procesoru a oba se deaktivují provedením nap, sleep nebo end.

Perioda PWM signálu je rovna periodě oscilátoru procesoru (při 4 MHz je to 250 ns) vynásobené konstantou 4 a zadanou hodnotou parametru zvýšenou o 1. Vypadá to složitě, příklad ale složitý není. Zadáme-li druhý parametr 100, bude perioda PWM signálu 250*4*(100+1) = 101000 ns = 101 mikrosekund = 0,101 ms (frekvence cca 10 kHz). Nejkratší možná perioda pulzů je 250*4*(1+1) = 2000 ns = 2 mikrosekundy (500 kHz), použijeme-li na místě druhého parametru hodnotu 0, pulzy vypneme. Nejdelší možná perioda je 250*4*(255+1) = 256000 ns = 256 mikrosekund (cca 3,9 kHz).

Třetí parametr je v příručce chybně popsaný jako činitel plnění, ve skutečnosti udává délku aktivního pulzu (výstup je v hodnotě logická 1). Délku aktivního pulzu spočítáme z hodnoty zadaného parametru * perioda oscilátor procesoru. Při zadání třetího parametru 200 bude délka pulzů odpovídat 250*200 = 50000 ns = 50 mikrosekund.

Vždy volíme nejdříve periodu PWM signálu, protože délka aktivního pulzu nemůže být libovolná a vychází z ní. Hodnota třetího parametru nesmí přesáhnout (přibližně) čtyřnásobek druhého parametru, jinak bude výstup trvale sepnutý a žádné pulzy na něm nebudou (tím bychom zadali aktivní pulz delší, než je celá perioda signálu). Pro střídu signálu 1:1 (činitel plnění 50%) je třetí parametr dvojnásobkem druhého.



Chceme-li například vytvořit řízení motoru s frekvencí PWM 8 kHz (periodou 125 mikrosekund), pak vychází zpětně hodnota druhého parametru 125000/250/4-1 = 124 a k tomu odpovídající rozsah třetího parametru (aktivního pulzu) pro ovládání výkonu od 1 do 125000/250 = 500. Zadáme-li nulu, přesně podle významu se pulzy vypnou, výstup bude trvale v nule.

Pro vyzkoušení si nejprve upravíme zapojení podle schématu. Odpojíme z pinu 2 odporový trimr a přepojíme na něj LED s předřadným rezistorem 330 ohmů. První program Rizeni1 bude blikat LED zhruba ve dvousekundových intervalech, to podstatné ale je, že LED bliká omezeným jasem vyvolaným právě PWM signálem. Vyzkoušíme si měnit parametry, druhým řídíme míru jasu, třetím dobu svitu.

```
1       REM RIZENI1 pro PICAXE 08M
2       REM ukazka příkazu PWM
3       start:                  ;smyčka programu
4         pwm 2,20,200          ;nastavení pulzů
5         pause 1000            ;prodleva 1s
6         goto start            ;zpět na začátek
```

Další program Rizeni2 pomalu cyklicky mění jas LED od nejmenšího po největší. Opět můžeme zkusit měnit parametry, všimneme si však především, že ačkoli je změna řízení lineární, změna jasu se na začátku jeví jako poměrně rychlá a v závěru každého cyklu velmi pomalá.

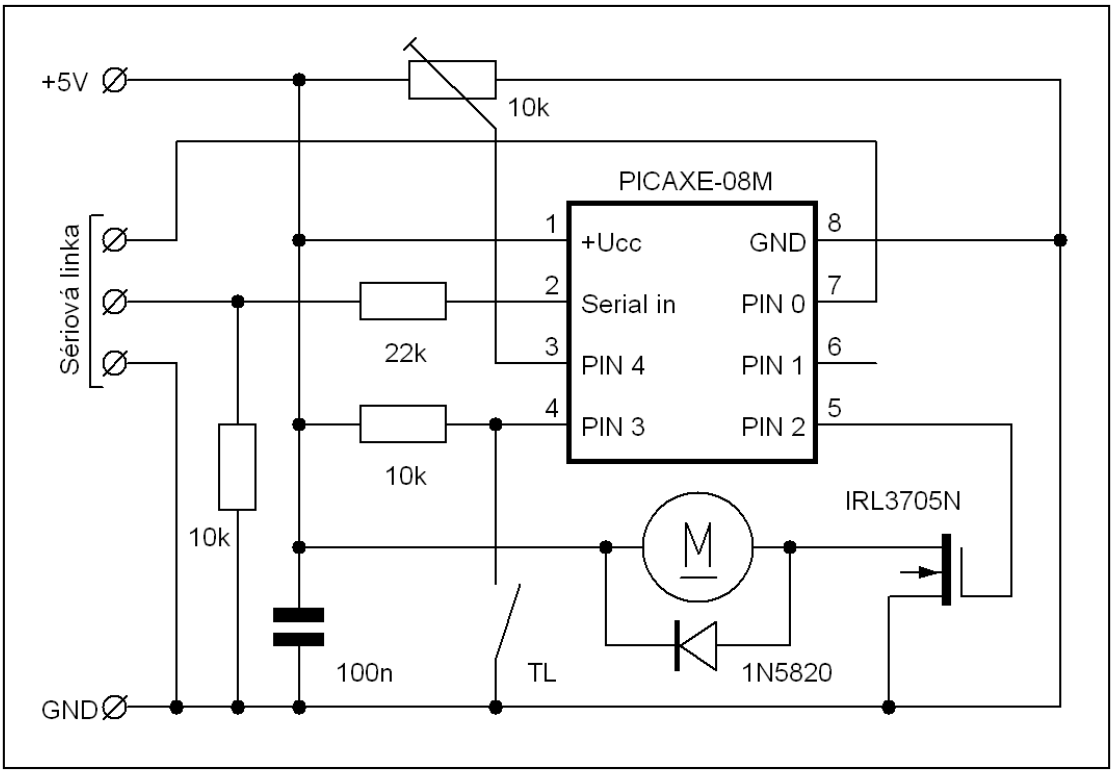
```
1       REM RIZENI2 pro PICAXE 08M
2       REM ukazka příkazu PWM
3       start:                  ;smyčka programu
4         for b0=0 to 255       ;cyklus změny jasu
5         pwm 2,b0,2            ;nastavení pulzů
6         next b0
7         goto start            ;zpět na začátek
```

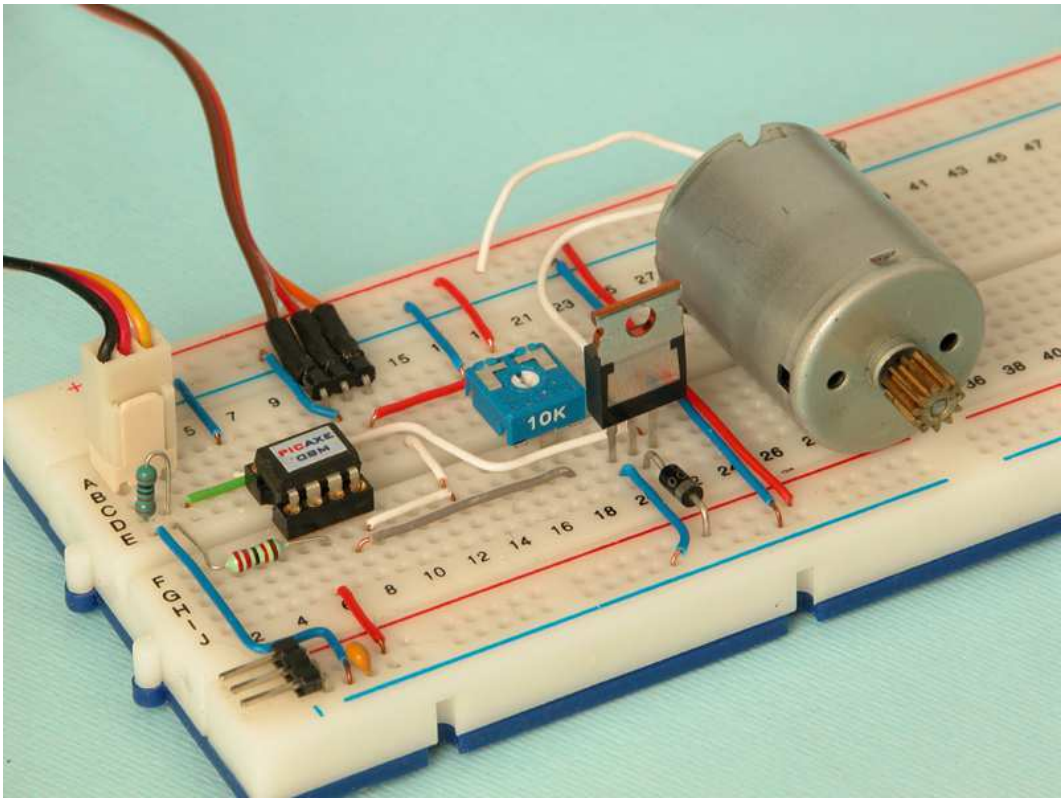
Třetí program využije ADC převodu napětí na pinu 4 pro ruční řízení LED trimem. Jas reaguje s malým zpožděním a jsou rozeznatelné jednotlivé stupně. Můžeme pozorovat i malé kolísání svitu způsobené různým vyhodnocením analogového vstupu.

```
1       REM RIZENI3 pro PICAXE 08M
2       REM PWM řízení LED ovládané ručně
3       start:                  ;smyčka programu
4         readadc 4,b0          ;čtení napětí pin4
5         pwm 2,b0,20           ;nastavení pulzů
6         goto start            ;zpět na začátek
```

Všechny uvedené pokusy můžeme zopakovat i s příkazem PWMOUT. Zajímavější ale bude zkusit zapojit místo LED na pin 2 FET spínač a skutečný elektromotor. Hodí se především malé typy motorů ze serv nebo nejvýše motor třídy 280 bez zátěže, aby bylo možné napájet celek ze stejného zdroje, tranzistor by zvládl mnohem větší zátěž. Upravíme zapojení podle dalšího schématu, odstraníme LED i její rezistor a zapojíme tranzistor IRL3705N a ochrannou diodu 1N5820.

Motor vyzkoušíme nejprve s programem Rizeni3. Na svitu LED to nijak výrazně vidět nebylo, ale chod motoru pravidelným kolísáním výrazně ukáže výpadky v řízení v době mimo provádění příkazu PWM. Další program Rizení4 bude už využívat příkaz PWMOUT, pulzy mají frekvenci kolem 4 kHz. Chod motoru je výrazně pravidelnější.


+5V
10k
PICAXE-08M
Sériová linka
1
+Ucc
GND
8
2
Serial in
PIN 0
7
22k
3
PIN 4
PIN 1
6
4
PIN 3
PIN 2
5
10k
10k
IRL3705N
M
100n
TL
1N5820
GND

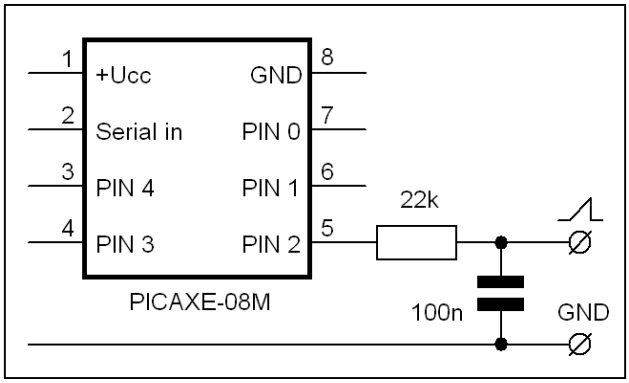

10K

```
1       REM RIZENI4 pro PICAXE 08M
2       REM PWM řízení motoru na pozadí
3       start:                      ;smyčka programu
4         readadc 4,b0              ;čtení napětí pin4
5         let w1=b0*4               ;úprava rozsahu
6         pwmout 2,255,w1           ;nastavení pulzů
7         pause 100                 ;rastr změn 0,1 s
8         goto start                ;zpět na začátek
```

Program lze samozřejmě upravit třeba na ovládání z počítače nebo na jednoduchý RC regulátor, stačí nahradit v zapojení tlačítko na pinu 3 a k němu příslušný rezistor vstupem signálu od přijímače. Tento regulátor lze pak dále rozvíjet, vybavit kontrolou minimálního napětí na vstupu nebo třeba řízenou rychlostí rozběhu a doběhu. Program Rizeni5 obsahuje jen základní funkci pro ověření činnosti a synchronizuje se na řídící signál od přijímače. Měření vstupních pulzů v rozsahu 1 - 2 ms poskytuje hodnoty 100 až 200, ty je nutno přetransformovat přibližně na rozsah 0 - 1023 pro příkaz PWMOUT, meze nejsou ošetřeny.

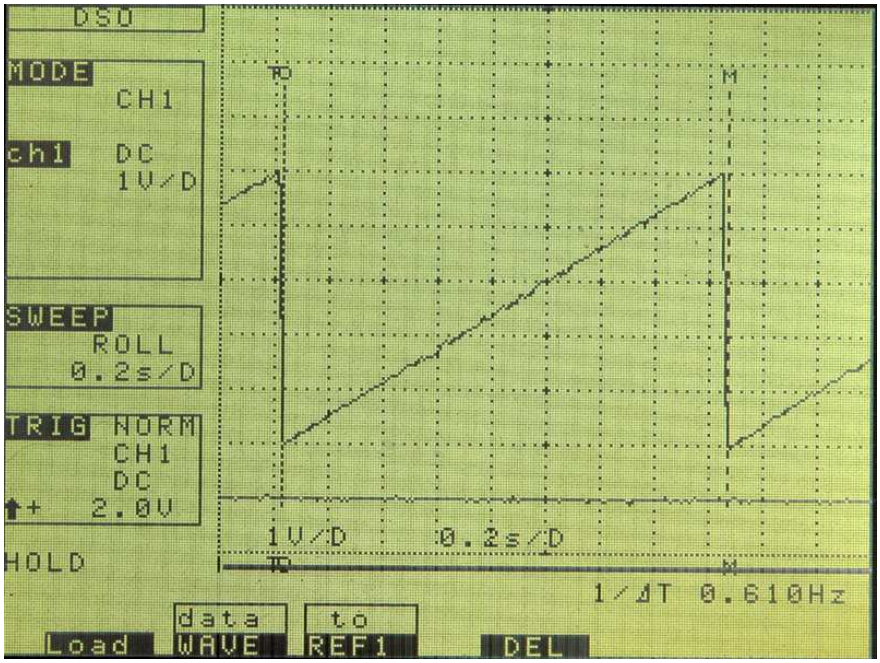
```
1       REM RIZENI5 pro PICAXE 08M
2       REM jednoduchý RC regulátor
3       start:                      ;smyčka programu
4         pulsin 3,1,w0             ;měření RC ovl.
5         let w1=w0-100*10          ;úprava rozsahu
6         pwmout 2,255,w1           ;nastavení pulzů
7         goto start                ;zpět na začátek
```

Příkaz PWMOUT je možné využít i jiným způsobem než k pulznímu řízení motoru. Procesor PICAXE 08 (08M) neobsahuje DA převodník a potřebujeme-li získat výstup ve formě napětí, můžeme na pin 2 zapojit filtr a převést tím PWM modulaci na napětí. Praktické vyzkoušení bude v tomto případě vyžadovat voltmetr. Místo spínacího tranzistoru na pin 2 zapojíme RC článek podle třetího částečného schématu.



Program DA1 mění pomalu střídu a tedy i napětí od nuly do maxima blízkého napájení. Je záměrně zpomalený tak, aby se nárůst dal dobře odečíst na digitálním voltmetru. Že je převod poměrně dobře lineární, to se můžeme přesvědčit na snímku z osciloskopu.

```
1       REM DA1 pro PICAXE 08M
2       REM DA převod pomocí PWM
3       start:                          ;smyčka programu
4         for w0=0 to 400               ;změny střídy
5         pwmout 2,100,w0               ;nastavení pulzů
6         pause 50                      ;zpomalení
7         next w0
8         goto start                    ;zpět na začátek
```



V praxi se poměrně často setkáváme s úlohami, které je obtížné řešit jedním procesorem, výhodnější je úlohu rozdělit na dílčí a nasadit dva nebo i více procesorů, které ovšem pak spolu musí nějak komunikovat. Kromě sériového přenosu dat, s nímž jsme se již zabývali, je jednou z možností právě převod veličiny na napětí prvním procesorem pomocí PWM a druhý procesor si toto napětí odebere (změří) v okamžiku, kdy sám potřebuje. Paměť mezi procesory realizujeme jediným kondenzátorem. Příklad najdeme v programu DA2, jenž převede servosignál z přijímače na napětí.

```
1       REM DA2 pro PICAXE 08M
2       REM převod servosignálu na U
3       start:                          ;smyčka programu
4         pulsin 3,1,w0                 ;měření servosig
5         let w0=w0-100*4               ;změna rozsahu
6         pwmout 2,100,w0               ;nastavení pulzů
7         goto start                    ;zpět na začátek
```

Závěr

Nastíněním možnosti spolupráce více procesorů jsme se dostali ke konci našeho seriálu, jenž měl především ukázat možnosti využití procesorů PICAXE v modelářské praxi, a umožnit zájemcům, aby se s minimem vybavení a poměrně snadno dopracovali k vlastnímu zapojení a naprogramování, které by řešilo jejich specifické problémy. Ohlasy svědčí o tom, že se přinejmenším částečně podařilo. Na tento seriál budeme občas navazovat zveřejňováním konkrétních praktických konstrukcí s procesory PICAXE.

Zdaleka jsme neprobrali všechny dostupné příkazy, od toho je příručka a aplikační dokumentace. Především jsme ale pracovali jen s nejslabším a nejmenším procesorem z řady PICAXE, na reálné úlohy můžeme nasadit procesory s podstatně větším počtem vstupů a výstupů, s mnohonásobně větší pamětí na program i data a také procesory nesrovnatelně rychlejší. Jejich stručný přehled a nabídku najdete na internetových stránkách www.snailshop.cz a v úvodu uvedených stránkách výrobce *www.rev-ed.co.uk/picaxe/* .

Obsah