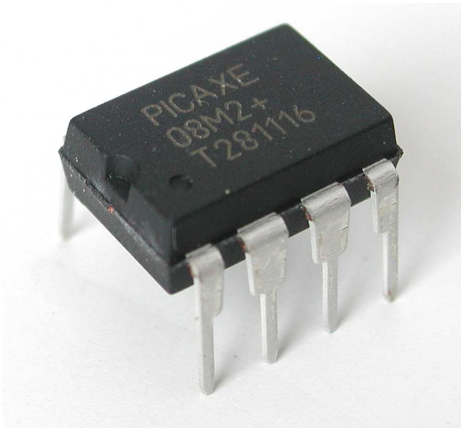
- mikrokontrolér pro začátečníky a snadné použití

Následující text byl poprvé zveřejněn v modelářském časopisu Praktická Elektronika – Amatérské Radio (www.aradio.cz) v roce 2012 jako seriál článků. Seriál, který si vzal za cíl představit na příkladech mikrokontroléry PICAXE a jejich programování v časopise určeném pro amatérské i profesionální elektroniky. Následující text v jiné grafické podobě má stejný obsah, tedy je neaktualizovaný s ohledem na změny v nabídce mikrokontroléru i novějších příkazů. Uvedené příklady volené s ohledem na zaměření časopisu a předpokládanou úroveň znalostí čtenářů nemusí být optimálním řešením daných úloh. O to vůbec nejde, cílem bylo názorně představit novým zájemcům možnosti mikrokontrolérů PICAXE na příkladu typu 08M2 a provést zájemce úvodem při psaní vlastních programů a aplikací těchto mikrokontrolérů.

Ing. Michal Černý

Začínáme

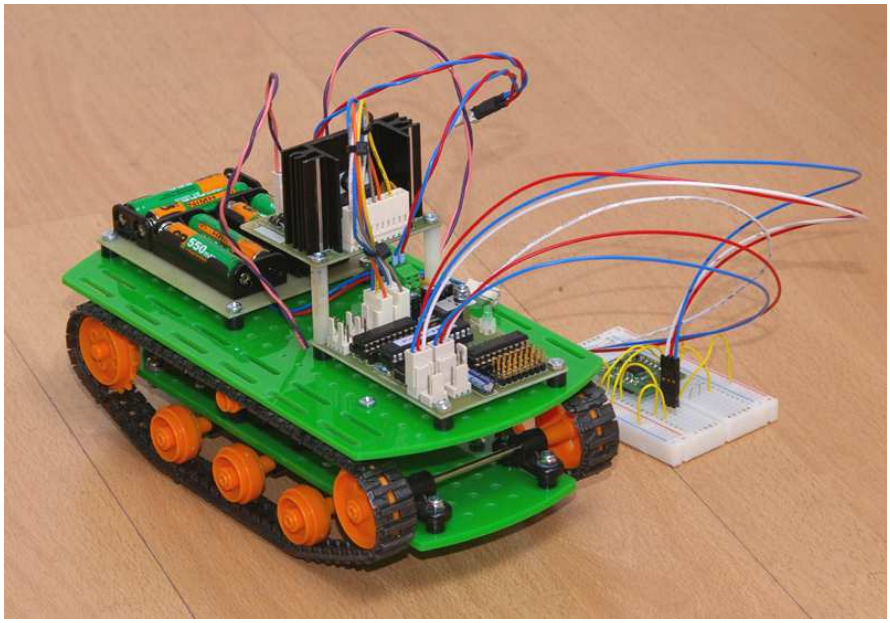
O tom, že mikrokontroléry masivně pronikly i do amatérských konstrukcí, zjednodušily je a přinesly mnoho nových možností i komfort, není myslím pochyb. Zatímco obvodové zjednodušení je objektivně vidět, pro mnohé začátečníky je nový přístup náročnější, než ten předchozí, protože musí současně zvládnout jak dovednosti související s obvodovým návrhem a fyzickou stavbou zařízení, tak programováním. Jedno bez druhého k cíli nevede. Ani pro zkušené konstruktéry není tento přechod snadný a mnozí se k němu neodhodlali. Mikrokontroléry PICAXE byly vytvořeny právě proto, aby maximálně zjednodušily a urychlily vstup do světa mikrokontrolérů a to s minimálními náklady. O tom, že se záměr podařilo splnit, svědčí mnoho příznivců těchto mikrokontrolérů ve všech věkových kategoriích.



Projekt PICAXE vznikl pro výukové účely a konstrukce z oblasti robotiky ve Velké Británii. Hlavní důraz byl kladen na co nejmenší nutné vybavení (a tedy i cenu), jednak na co největší srozumitelnost a „čitelnost“

programování. Záhy se ale mikrokontroléry PICAXE ujaly i mimo oblast výuky a amatérské robotiky. Dá se říci, že je to cesta vhodná pro ty, kdo neumí nebo nechtějí programovat tak, jak se s mikrokontroléry obvykle pracuje, tedy v assembleru nebo různých verzích jazyka C, ale potřebují použít mikrokontrolér a rychle vyřešit svůj problém.

Seriál, který právě začínáme, si bere za cíl přiblížit na praktických příkladech programování mikrokontrolérů PICAXE i těm, kteří předchozí zkušenosti s programováním nemají, případně se trochu rozpomenou na mladá léta a dobu osmibitů, kdy Windows neexistovaly, a práce s počítačem byla totéž, co alespoň základní znalost programování v Basicu. Budeme používat PC se systémem Windows, protože v počítači budeme programy pro PICAXE tvořit. Ke zkoušení uváděných konstrukcí stačí jedno nepájivé kontaktní pole. Smyslem konstrukcí je především demonstrovat jak činnost jednotlivých příkazů a fragmentů programu, tak ukázat meze použitelnosti mikrokontrolérů PICAXE. Za výhody, které přinášejí, je samozřejmě nutné něčím zaplatit, a to je výkon mikrokontroléru. Program efektivně napsaný přímo v assembleru s detailní znalostí možností daného typu obvodu vždy bude pracovat rychleji a mít větší možnosti, pokud ale PICAXE na problém stačí, je napsání programu pro něj naprosto nesrovnatelně rychlejší.



Mikrokontroléry PICAXE vycházejí z mikrokontrolérů PIC vyráběných firmou Microchip, do nichž byl vložen speciální zaváděcí program a předprogramována řada užitečných funkcí. Zaváděcí program dovoluje přeprogramování uživatelského programu z PC bez nutnosti použít speciální programátor, místo něj stačí kabel k sériovému portu počítače. Obvody lze přeprogramovat zhruba 100000x, což celkem neomezuje možnost aplikování „metody pokusů a omylů“. Zavaděč nesmíme z mikrokontroléru 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:

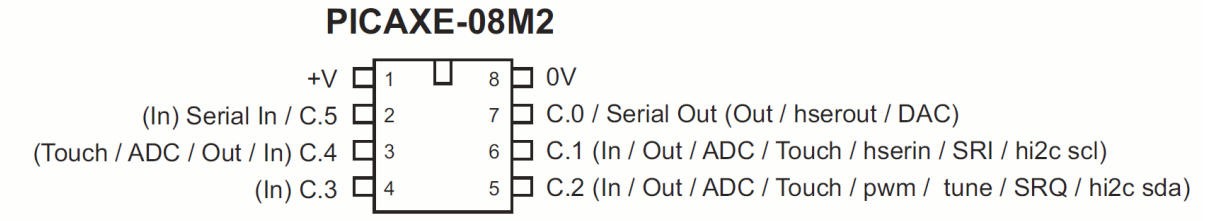
Nikdy nesmíme napájecí napětí mikrokontroléru přepólovat a nikdy nesmíme překročit horní mez napájecího napětí (obvykle 5,5 V). I pokud by sám mikrokontrolér jako součástka obě události přežil, zavaděč se tím pravděpodobně zničí.

Mikrokontrolérů PICAXE je celá řada typů, od malého osmivývodového PICAXE-08M2 po čtyřicetivýodový PICAXE-40X2 s podstatně posílenými funkcemi v oblasti matematiky. Budeme používat především nejmenší a nejlevnější typ z řady M2 (08M2), pokud počtem vývodů nebo možnostmi nestačí, pak budou uvedeny ukázky i s mikrokontrolérem 18M2 nebo 20X2 z řady X2. První pokusy s programováním lze uskutečnit zcela bez finančních nákladů a kdykoli, 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. Používané součástky budou běžně dostupné, speciální díly (mikrokontroléry PICAXE, sériový displej SIC1602AYPLEB20), najdete v nabídce internetového obchodu *www.snailshop.cz* . Můžeme tedy začít.

Vývojové prostředí a první pokusy

Prvním krokem bude stáhnout si z internetových stránek *www.aradio.cz* balíček s vývojovým prostředím verze 5.5.0, mírně zastaralou programátorskou příručkou v češtině (lepší bohužel není) a kompletní aktuální dokumentací v angličtině. Součástí vývojového prostředí, které lze přepnout i do českého jazyka, je editor pro psaní programu i prostředky pro vyzkoušení a simulaci uživatelských programů. Lokalizace do češtiny sice není úplná ve všech položkách nabídky, ale k usnadnění práce určitě stačí. Program nainstalujeme do PC běžným způsobem. Další informace lze najít na domovských stránkách výrobce *www.PICAXE.com* , kde je vždy aktuální verze programu. Protože je důležité, abychom se mohli v celém seriálu odvolávat na stejnou verzi programu, je použité vývojové prostředí uloženo v balíčku. Seriál nemůže suplovat originální dokumentaci, má seznámit se základy, všechny souvislosti ale postihnout nemůže.

Rozmístění vývodů PICAXE-08M2 je na obrázku. Vývod 1 je kladné napájení, mělo by se pohybovat v rozmezí 1,8 - 5V, nikdy nesmí přesáhnout 5,5V. Mikrokontrolér je výborně slučitelný jak s klasickými úrovněmi TTL, tak 3V logikou. Vhodným zdrojem pro pokusy je čtyřčlánek tužkových akumulátorů NiCd nebo NiMH (ne alkalických článků!). Vývod 2 se používá pro programování obvodu z počítače, ale kromě toho jde případně použít i jako univerzální vstup. Není-li vstup připojen k sériové lince z PC ani jej cíleně nevyužíváme, nenecháváme tento vývod volně, ale spojíme jej se zemí. Vývody 3 až 7 jsou vstupy a výstupy obvodu a budeme je značit v souladu s obrázkem Pin C.4 až Pin C.0 nebo jednodušeji jako Pin 4 až Pin 0. Pin 0 je vždy výstupem a slouží spolu s vývodem 2 ke komunikaci s počítačem, může být ale využit i v našem programu jako univerzální výstup. Pin 3 je vždy nastaven jako vstup, zbývající tři (Piny 1,2 a 4) můžeme podle potřeby použít buď jako vstup nebo jako výstup, případně je z programově podle potřeby přepínat. Po zapnutí napájení mikrokontroléru jsou tyto vývody nastaveny jako vstupy, pokud je aktivujeme jako výstupy, ale nezadáme jejich stav, budou v úrovni L. Poslední vývod číslo 8 je zem napájení.



Po spuštění programu jdeme do nastavení (Option), vybereme typ mikrokontroléru 08M2 a v další záložce sériový port, přes nějž budeme mikrokontrolér (časem) programovat. Pokud máme k počítači připojený USB převodník dodávaný výrobcem PICAXE, nastaví se přímo, program ale bez problémů funguje i s běžně používanými převodníky USB/COM. Pro pokusy se simulací není nastavení portu nutné. V záložce Jazyk zkontrolujeme české nastavení, ostatní položky můžeme nechat zatím tak jak jsou.

Můžeme si tedy vysvětlit první příkazy a začít psát. Jednotlivé řádky programu jsou očíslované, číslují a přečíslovávají se samy. Jazyk, v němž se píše, je verzí Basicu, ale čísla řádek se v praxi nijak nepoužívají a jsou spíš atavismem. Budeme-li se odvolávat na českou příručku, bude v závorce číslo strany příručky uvedené hvězdičkou**,** například (*7) je sedmá strana české příručky programátora. V anglické dokumentaci fungují odkazy z obsahu a textové vyhledávání.

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 mikrokontrolérů PICAXE, 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). Příklad: REM Tohle je komentář

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). Příklad: Blok6:

HIGH, LOW

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

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).

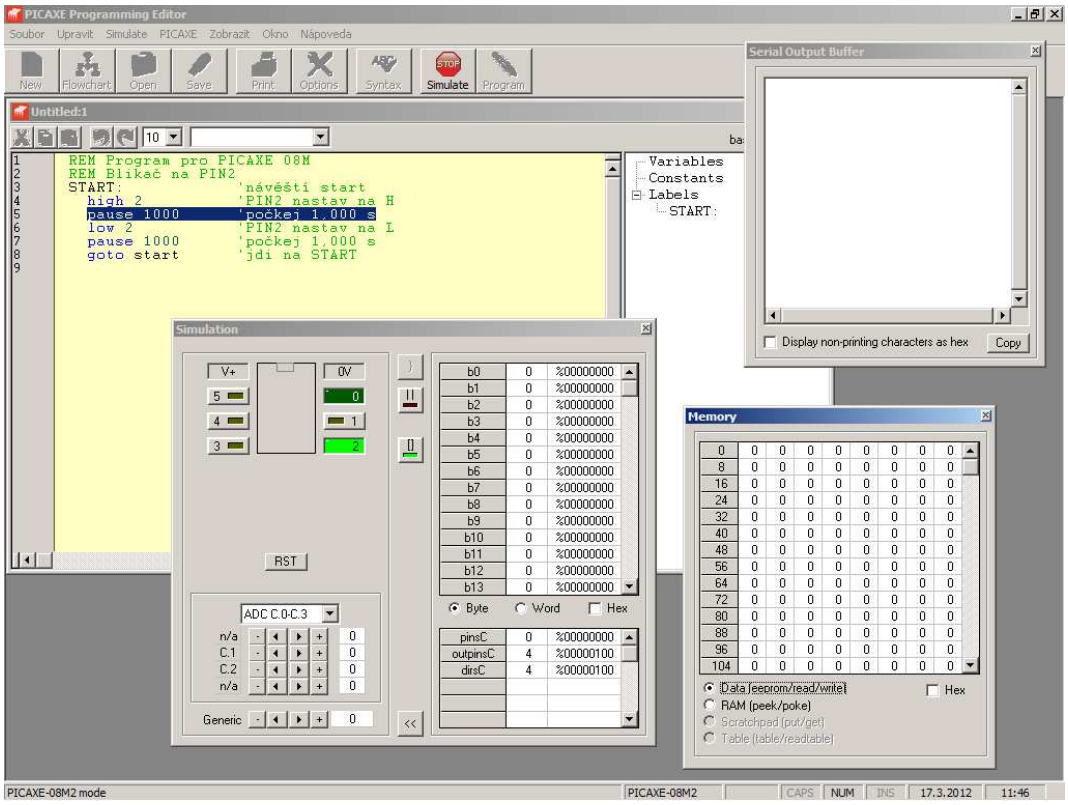
GOTO

Parametrem příkazu GOTO je námi zadané návěští, kam program bez dalších podmínek ihned skočí a pokračuje ve vykonávání příkazů (*9). Návěští se v příkazu GOTO uvádí bez dvojtečky, ta se používá jen když návěští definujeme (jen jednou). Příklad: GOTO Start

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 08M2
2       REM Blikac na PIN2
3       START:                  'náveští start
4         high 2                'PIN2 nastav na H
5         pause 1000            'pockej 1000 ms
6         low 2                 'PIN2 nastav na L
7         pause 1000            'pockej 1000 ms
8         goto start          'jdi na START
```

Jakmile je program napsaný, zkusíme jeho simulaci přes Ctrl+F5 nebo lépe tlačítkem Simulate v horní liště nástrojů. Vedlejší tlačítko Program s obrázkem konektoru zatím nepoužíváme, to slouží k přenesení programu do mikrokontroléru přes programovací kabel. 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, a nemá možnost zjistit, že to není to, co chceme.



Spuštění simulace vyvolá okno, v jehož levé horní části je schématicky naznačené rozmístění vývodů PICAXE-08M2 a příslušné Piny jsou označeny čísly. Vstupy jsou zobrazeny jako tlačítka a kliknutím na ně můžeme přímo nastavit stav na nich (indikuje se „ledkou“ v tlačítku), výstupy jsou jako barevná pole, jasně zelená barva značí úroveň H. Ploška s číslem 2 by se měla rozblikat zelenou barvou s periodou 2 s. Simulaci zastavíme stiskem tlačítka Stop. Tím byl první úkol splněn, blikač je hotový, zatím alespoň jako simulace.

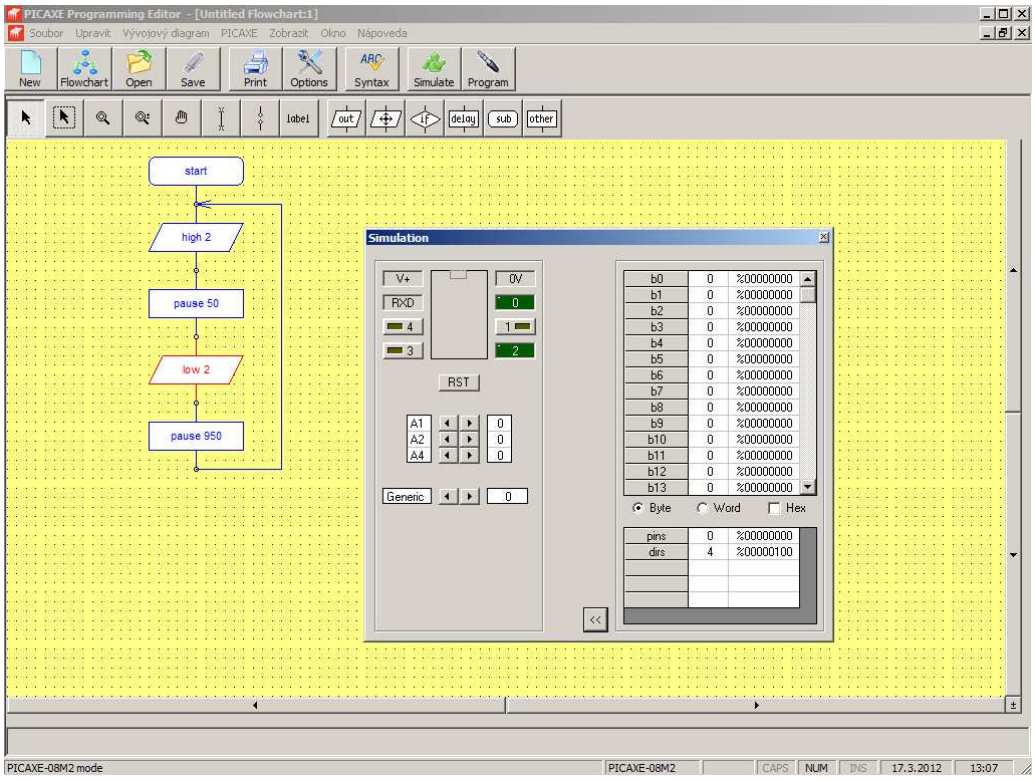
Pro zažití je dobré si vyzkoušet několik úprav, například zrychlit blikání dvakrát, dělat krátké záblesky prokládané dlouhými mezerami nebo naopak a změnit program tak, aby místo Pinu 2 blikal jiný, třeba Pin 4. I takto jednoduchý program je možné zkrátit, a to docela podstatně. Seznámíme se s dalším 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. Příklad: TOGGLE 2

Náš program pomocí nového příkazu zkrátíme o dva funkční řádky, ovšem za tu cenu, že již nebudeme moci měnit střídu, bude vždy 1:1 a do příkazu Pause zadáme polovinu doby odpovídající periodě. Doba potřebná na vykonání ostatních příkazů je zatím zcela zanedbatelná, program tráví naprostou většinu času čekáním. Z programu není vidět, co bude na začátku na výstupu, což ovšem nijak nevadí. Opět si zkusíme modifikovat program tak, aby blikal jiný výstup, třeba PIN 1, zrychlit a zpomalit blikání.

```
1       REM Program pro PICAXE 08M2
2       REM Blikac na PIN2 verze 2
3       START:                  'náveští start
4         toggle 2              'PIN2 zmen stav
5         pause 1000            'pockej 1,000 s
6         goto start            'jdi na START
```

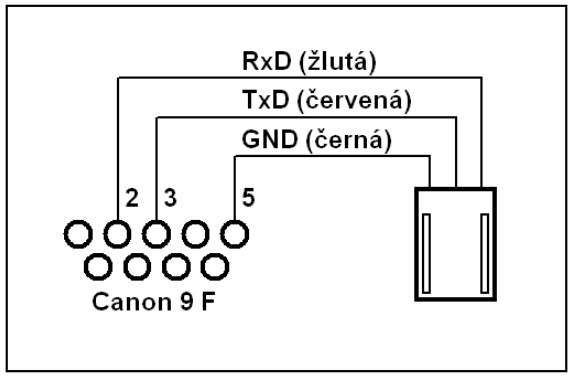


Úlohu jde do editoru zadat i jinak než textově. Když prostřednictvím tlačítka Flowchart otevřeme editor pro kreslení, či spíče sestavování blokových schémat, můžeme z nabídky

nahoře brát jednotlivé příkazy a pospojovat z nich podobný program. Na obrázku je verze programu s konstantami nastavenými na krátké záblesky 0,05 s 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, buď pracujeme jedním způsobem nebo druhým.

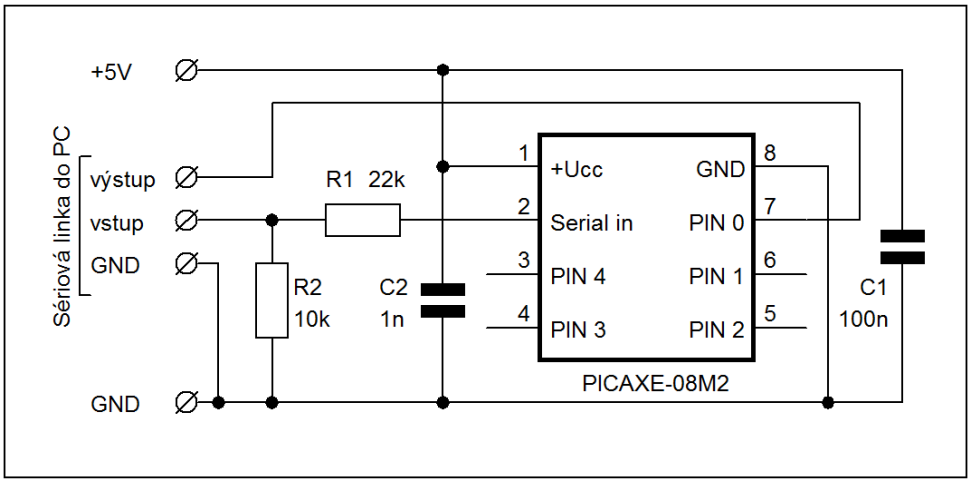
Zapojujeme mikrokontrolér

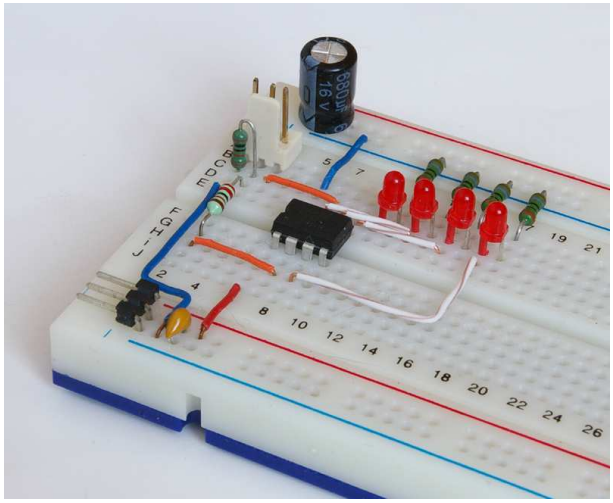
Až do tohoto okamžiku jsme nepotřebovali nic víc než počítač s nainstalovaným vývojovým prostředím. Ve zkoušení příkazů simulací se dá i pokračovat, naším cílem ale jsou hmatatelná a fungující zapojení. Budeme potřebovat zkušební desku s mikrokontrolérem a programovací kabel. Kabel můžeme podobně jako ostatní potřebné součástky buď koupit (*www.snailshop.cz*) nebo vyrobit, stačí k tomu devítikolíkový konektor Canon (protikus sériového konektoru v PC), třížilový kabel a konektor PFH02-03P (GM electronic) s kontakty (případně jiný, který si zvolíme). Zapojení kabelu je ve schématu.



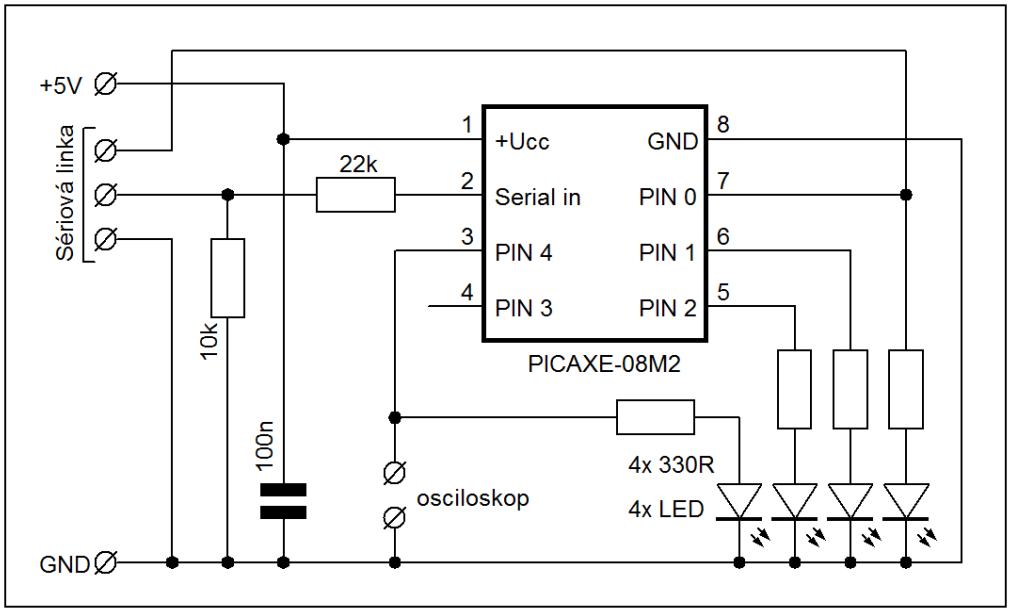
Pokud počítač nemá COM port ale jen USB, musíme ještě opatřit převodník USB/COM nebo originální kabel, který je ale zakončen konektorem Jack 3,5 stereo, jak je zvykem v GB a USA. Tento konektor (jeho protikus ve verzi do DPS) je pravděpodobně lepší pro přípravky a zapojení realizovaná na DPS, pro pokusy je výhodnější konektor s řadovými kontakty.

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





Obvod vytvoříme 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, bohužel nejde použít, má krátké vývody. Mikrokontrolér můžeme osadit do precizní objímky, vzhledem k tomu, že se s ním prakticky nehýbe a přeprogramovává se v zapojení, to však není nutné. Pro napájení použijeme třípinový konektor ze zahnuté propojkové lišty a krajní kontakt odštípneme, takže chybnou orientací nelze napájení přepólovat. Základní zapojení doplníme čtyřmi LED s předřadnými rezistory.



Po kontrole zapojení připojíme sériovou linku a napájecí zdroj, zatím by se nic dít nemělo. V editoru máme připravený program pro rozblikání LED na výstupu Pin 2. Stiskneme tlačítko Program pro přenos do mikrokontroléru. 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 u rozsáhlejších programů. Nepodaří-li se přenos, bývá problém v nepřipojeném napájení nebo napoprvé v chybně nastaveném sériovém portu. 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. Kapacita programové paměti mikrokontroléru je 2048 byte, to odpovídá přibližně 1500 - 1800 řádkům psaného programu s jedním výkonným příkazem na řádku.

Běžící světlo

Různí světelní hadi a běžící světla jsou oblíbenými začátečnickými konstrukcemi, předchozí zapojení je připravené právě na tuto úlohu. Použijeme jen instrukce, s nimiž jsme se zatím seznámili. Je obecně lepší psát jednu výkonnou instrukci na jeden řádek, ale v časopise by to neúměrně prodlužovalo výpisy i jednoduchých programů, proto jsou instrukce, které spolu logicky souvisí, psány někdy za sebe do jednoho řádku. K oddělení instrukcí stačí mezera, přechod na další řádek není pro program podstatný. V zapojení by se po přenosu programu do mikrokontroléru měly postupně rozblikat LED v řadě. Pin 0 slouží také při programování a je to vidět, jeho LED při přenosu dat svítí nebo poblikává.

```
1       REM Svetelný had - PICAXE 08M2
2       start:
3         high 0 pause 200 low 0
4         high 1 pause 200 low 1
5         high 2 pause 200 low 2
6         high 4 pause 200 low 4
7         goto start
```

Programu popisuje všechny změny stavu za sebou tak, jak si je můžeme rozkreslit do sekvence na papír. Je funkční, i když neefektivní, zbytečně dlouhý, a pro větší počet výstupů by to bylo jen horší. Asi nejdůležitější je to, že při pokusu o úpravy rychlosti běhu,

prodloužení nebo zkrácení záblesků a podobně, bychom museli měnit stejné parametry na mnoha místech. Je čas seznámit se s konstantami, symboly a operátory.

KONSTANTY

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

SYMBOLY

respektive symbolické názvy jsme již také poznali, jejich 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 je v (*3). Kromě základních operátorů se často používá ještě MIN a MAX s jedním parametrem, které omezí hodnotu napsanou před nimi na minimum respektive maximum podle parametru. Protože se velmi často používá přičítání nebo odečítání jedničky k obsahu nějaké proměnné (viz dále), mají tyto operace i své vlastní zkrácené (a rychlejší) povely a to DEC a INC. Příklad: INC b0 MAX 55 přičte 1 do proměnné b0, ale výsledek je nejvýše 55.

Nyní se pokusíme celý program napsat znovu. Použijeme jednu symbolicky pojmenovanou konstantu, pomocí níž se pak dá jednoduše na začátku programu nastavit rychlost běhu světla.

```
1       REM Svetelný had - PICAXE 08M2
2       symbol cekani=200
3       start:
4         high 0 pause cekani low 0
5         high 1 pause cekani low 1
6         high 2 pause cekani low 2
7         high 4 pause cekani low 4
8         goto start
```

Pokusíme se program dále zkrátit a zbavíme se i nutnosti vypisovat příkazy pro každý výstup jednotlivě, pro čtyři to ještě jde, ale kdyby jich bylo třeba 8 (s mikrokontrolérem 20X2) bude už rozdíl hodně výrazný. Budeme potřebovat proměnné a cykly.

PROMĚNNÉ a PŘIŘAZENÍ

Proměnná je místo v paměti mikrokontroléru, kam můžeme uložit nějakou hodnotu a používat ji. Po zapnutí napájení mají všechny proměnné hodnotu 0. K dispozici máme celkem 28 proměnných jednobytových (hodnoty 0 až 255) pojmenovaných b0 až b27, samozřejmě si je můžeme označit vlastním symbolickým jménem. Je-li potřeba uložit vyšší hodnoty v rozsahu 0 až 65535, můžeme stejný prostor v mikrokontroléru využít jako 14 proměnných dvoubytových (typ word) značených w0 až w13, 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 si ale pohlídat, abychom se na stejné místo v paměti mikrokontroléru 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, ve většině případů se dokonce nemusí ani slovo LET uvádět.

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

B0	W0
B1	
B2	W1
B3	

....

B24	W12
B25	
B26	W13
B27	

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.

Nová verze světelného hada bude využívat FOR cyklus, protože ale výstupy 0,1,2,4 netvoří postupnou řadu (Pin 3 může být jen vstup u mikrokontroléru 08M2), omezíme se jen na tři LED. Přece jen, v tomto případě jde především o ukázku, ne o výslednou funkci. S mikrokontrolérem 20X2 a jeho osmi výstupy v řadě by byl program stejný, změnily by se jen konstanty.

```
1   REM Svetelný had - PICAXE 08M2
2   symbol cekani=200
3   start:
4    for b0=0 to 2
5    high b0 pause cekani low b0
6    next
7    goto start
```

Podobným způsobem lze řešit různé sekvenční spínače osvětlení pro reklamy, řízení antikolizních světel pro modely letadel, osvětlení vánočního stromku, … aplikací se najde hodně. Zkusíme vytvořit sekvencer který krátce blikne jedním výstupem 3x po sobě, pak druhým a třetím výstupem stejně. Mohla by to být třeba simulace zábleskových výstražných světel. Blikání bude zatím naprosto pravidelné, později se naučíme, jak simulovat, že každé ze světel běží samostatně a nepatrně jinak než ostatní.

Upozorním na jeden obecnější rys. Zařízení s PICAXE jsou tak rychle a snadno

přeprogramovatelná, že jen ty parametry, které se opravdu musí měnit často a za provozu, se mění uživatelsky za pomoci tlačítek, propojek, trimrů a podobně. Parametry, které se nemusí měnit za provozu, spíše se jimi jen občas nastavuje provozní režim, mohou zůstat jako konstanty se symbolickými jmény v programu. To se projeví na dost výrazném snížení počtu nutných vstupů a dalším zjednodušení zapojení.

```
1       REM sekvencer1 - PICAXE 08M2
2       symbol doba1=5              ;délka bliknutí
3       symbol doba2=120            ;prodleva v sérii
4       symbol doba3=800            ;prodleva mezi sériemi
5       start:
6         for b0=0 to 2             ;vnejší cyklus - výstupy
7           for b1=1 to 3           ;vnitrní cyklus - záblesky
8           high b0 pause doba1 low b0 pause doba2
9           next
10        pause doba3
11        next
12        goto start
```

Rychleji, přesněji ...

Trochu odbočíme. Nabízí se otázka, jak rychle náš úplně první program může běžet, jaké nejkratší pulzy může PICAXE 08M2 tímto způsobem programem dělat. Přesměrováním na Pin 4, na němž máme bod pro připojení osciloskopu, odstraněním čekání a všeho zbytečného dojdeme k jednořádkovému programu, který generuje symetrické pulzy. Pulzy jsou rychlé a viditelně se projeví jen snížením jasu LED (a skutečně, takto se dá změnou střídy měnit jas LED i plynule).

```
START: toggle 4     goto start
```

Osciloskop ukáže pulzy s frekvencí 445 Hz (2,32 ms) a je vidět že nejsou naprosto pravidelné, perioda se nepatrně mění, chvěje. K blikání to stačí, ale jinak to není moc dobrý výsledek. Nejdéle trvá skok programu zase na začátek, takže pokud potřebujeme jen několik pulzů a máme dost paměti, můžeme vyrobit pulzy delší sekvencí HIGH 4 LOW 4 HIGH 4 LOW 4 …. V takovém případě mají pulzy 1607 Hz (622 µs). To už je trochu lepší, možnosti PICAXE jsou ale mnohem širší.

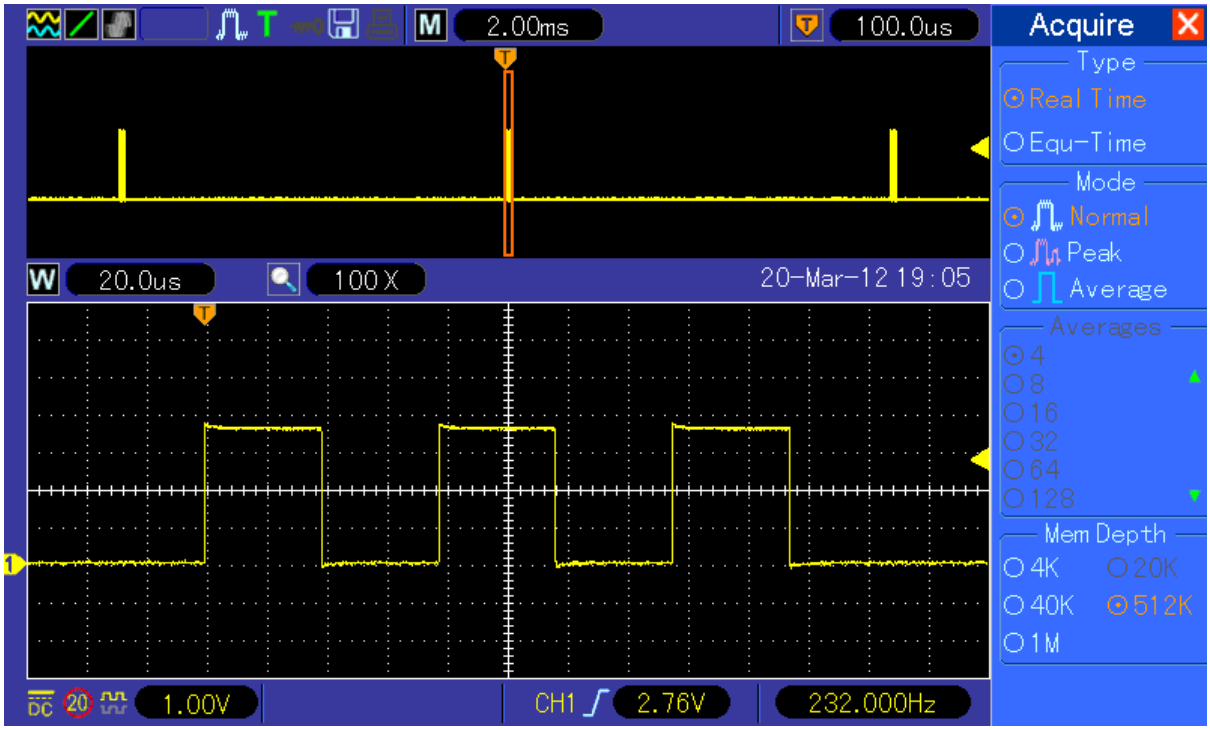
Základní kmitočet mikrokontroléru, který určuje většinu časování, je 4 MHz. Příkazem SETFREQ můžeme tento kmitočet nastavit, v případě PICAXE 08M2 jej zvýšit na 8, 16 nebo 32 MHz nebo také snížit až na 31 kHz. Místo 445 Hz tak získáme přes 3,56 kHz nebo také 3 Hz. Zkusíme vyrobit trojici co nejkratších pulzů opakující se zhruba po 0,1 s.

SETFREQ

může změnit vnitřní hodinovou frekvencí, tím se většina časových údajů zkrátí, třeba parametr PAUSE při zdvojnásobení frekvence by již nebyl v ms, ale násobcích 0,5 ms. Parametr „m4“ nastaví základní obvyklou frekvenci 4 MHz, „m8“ přepne na 8 MHz, „m16“

přepne na 16 MHz, „m32“ přepne na 32 MHz. Analogicky třeba „k250“ odpovídá 250kHz. Změnu hodinové frekvence mikrokontroléru bychom měli používat jen pokud to má opravdu smysl a s rozvahou, může totiž způsobit problémy v provádění některých příkazů, třeba při sériové komunikaci. Všechny příkazy jsou odladěny pro základní frekvenci 4 MHz, případně jinou uvedenou v dokumentaci.

```
1       REM Program trojice - PICAXE 8M2
2       setfreq m32             ;frekvence 32 MHz
3       START:                  ;zacátek smycky programu
4         high 4 low 4          ;1.pulz
5         high 4 low 4          ;2.pulz
6         high 4 low 4          ;3.pulz
7         pause 100             ;prodleva 100 ms
8         goto start
```



Důvodem ke změně pracovní frekvence mikrokontroléru nemusí být jen to, že potřebujeme vyšší výkon mikrokontroléru, rychlejší reakci. Ruku v ruce s tím jde také spotřeba, a ta může být zejména u dlouhodobě pracujících zařízení s bateriovým napájením kritická. Pracovní frekvenci jde kdykoli za běhu programu měnit a tak mikrokontrolér může čekat při velmi líném taktu a nepatrné spotřebě až nastane očekávaná událost (třeba občas měří úroveň hladiny, to nespěchá) a potom kvůli přímému řízení synchronního motoru přepne na rychlý takt. Jak velký je rozdíl v odběru proudu? Necháme mikrokontrolér běžet ve smyčce našeho prvního programu s běžícím světlem s odstraněným čekáním. Odpojíme LED ze zapojení a změříme odběr při napájení 5,0 V. Rozdíl je značný, při 32 MHz naměříme 2,2 mA, při 31 kHz 0,13 mA. Je to ostatně logické, hlavní část spotřeby mikrokontroléru tvoří nabíjení a vybíjení kapacit vnitřních klopných obvodů.

Pracovní frekvenci mikrokontroléru 08M2 určuje vnitřní oscilátor. Jeho přesnost a stabilita na naprostou většinu aplikací stačí, je-li potřeba větší, některé vyšší typy PICAXE mají možnost zapojit vnější oscilátor. U mikrokontrolérů s vnitřním oscilátorem jako je i 08M2 si můžeme pomoci příkazem CALIBFREQ (*6), tím lze frekvenci drobně upravit a to přibližně v intervalu 5%.

Multitasking

Vrátíme se k úloze běžícího světla. Zavaděč mikrokontroléru není jen zavaděčem, plní současně i funkci malého operačního systému, který dovoluje spuštění až čtyř programů běžících (víceméně) nezávisle. Ve skutečnosti nejde o opravdový paralelní běh programů, nemáme k dispozici více jader, ale o jejich velmi rychlé střídání. Musíme obětovat možnost změny hodinového kmitočtu, mikrokontrolér se sám přepne na 32 MHz a všechny programy poběží přibližně tak, jak by běžely na kmitočtu 4 MHz. To samozřejmě na první pohled nevychází, musíme si ale uvědomit, že střídání procesů je hodně náročné a „ukrojí“ si z výkonu mikrokontroléru pořádný kus, zhruba polovinu. Střídání procesů také způsobí, že jednotlivé programy se prakticky nemohou synchronizovat na stejné události (třeba reagovat na stejný stisk tlačítka). Výměna parametrů mezi nezávislými vlákny se může uskutečnit nejjednodušeji přes proměnné.

Začátky programů jsou uvedeny pevně daným návěštím START0 (nebo START), START1, START2 a START3. To stačí, žádný nový příkaz není třeba. Modifikujeme tedy náš zábleskový program sekvencer1 tak, aby se zdálo, že jednotlivé LED jsou časovány nezávisle na sobě. O jednotlivé LED se starají tři samostatné a téměř stejné programy. Když dosadíme do konstant určujících prodlevu prvočísla, bude trvat pořádně dlouho, než se sekvence blikání zase začne přesně opakovat.

```
1       REM sekvencer2 - PICAXE 08M2
2       start0:                                 ;první z paralelních programu
3         for b0=1 to 3                         ;tri pulzy v sérii
4           high 0 pause 20 low 0 pause 120     ;jeden pulz
5           next
6         pause 1009                            ;prodleva mezi sériemi
7         goto start0
8       start1:                                 ;druhá z paralelních programu
9         for b1=1 to 3 high 1 pause 20 low 1 pause 120 next
10        pause 1097 goto start1
11      start2:                                 ;tretí z paralelních programu
12        for b2=1 to 3 high 2 pause 20 low 2 pause 120 next
13        pause 1193 goto start2
```

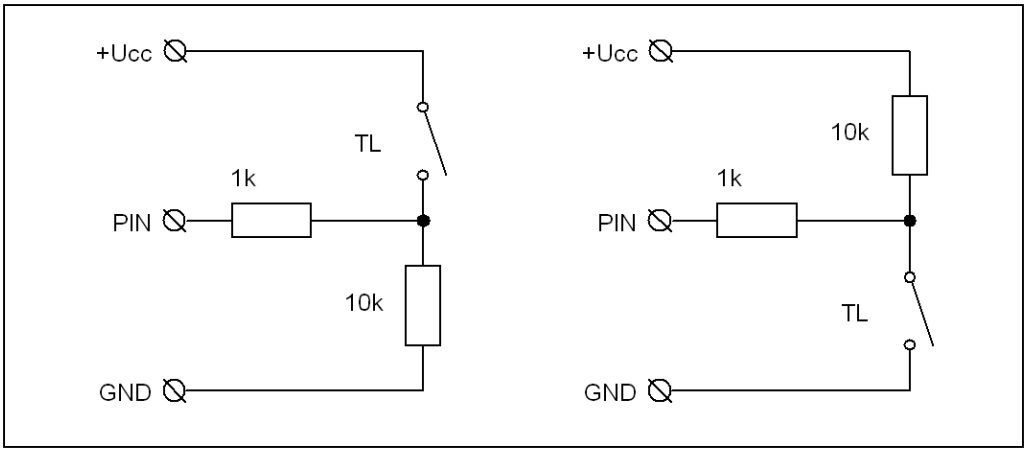
Multitasking je velmi silný nástroj, který nám umožňuje zkrátit a hlavně zjednodušit programy, není ale vhodný pro úlohy, u nichž záleží na přesném časování programu. Technické prostředky mikrokontroléru (třeba časovač) může samozřejmě využívat v jednom okamžiku jen jeden z programů, ty se rozmnožit nedají, takže pokud jedno vlákno třeba odesílá data sériovou linkou, ostatní vlákna se zastaví!

Obsluha tlačítek

Mechanické spínací kontakty patří mezi to nejčastější, co se připojuje ke vstupům mikrokontroléru. Nezapojené vstupy zpravidla vykazují stav L, ale obecně jsou ve stavu vysoké impedance, což znamená, že stačí i nepatrný podnět, aby svůj stav změnily. Můžeme si to názorně vyzkoušet následujícím programem, který čte stav na Pin 3 a výsledek bezprostředně kopíruje na výstup osazený LED. Stav vstupu lze jednoduše přiřadit do proměnné a naopak, stav proměnné poslat na výstupní pin 0, případně jde přiřadit výstupnímu pinu přímo stav vstupního.

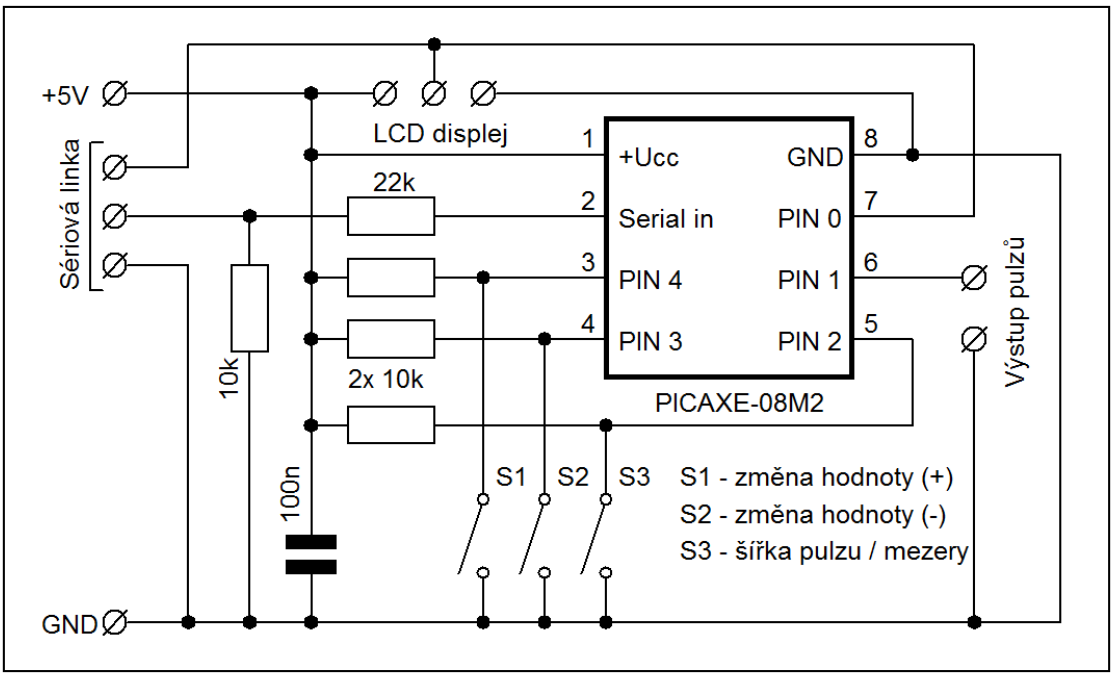
```
start: b0=pin3 pin0=b0 goto start
nebo
start: pin0=pin3 goto start
```

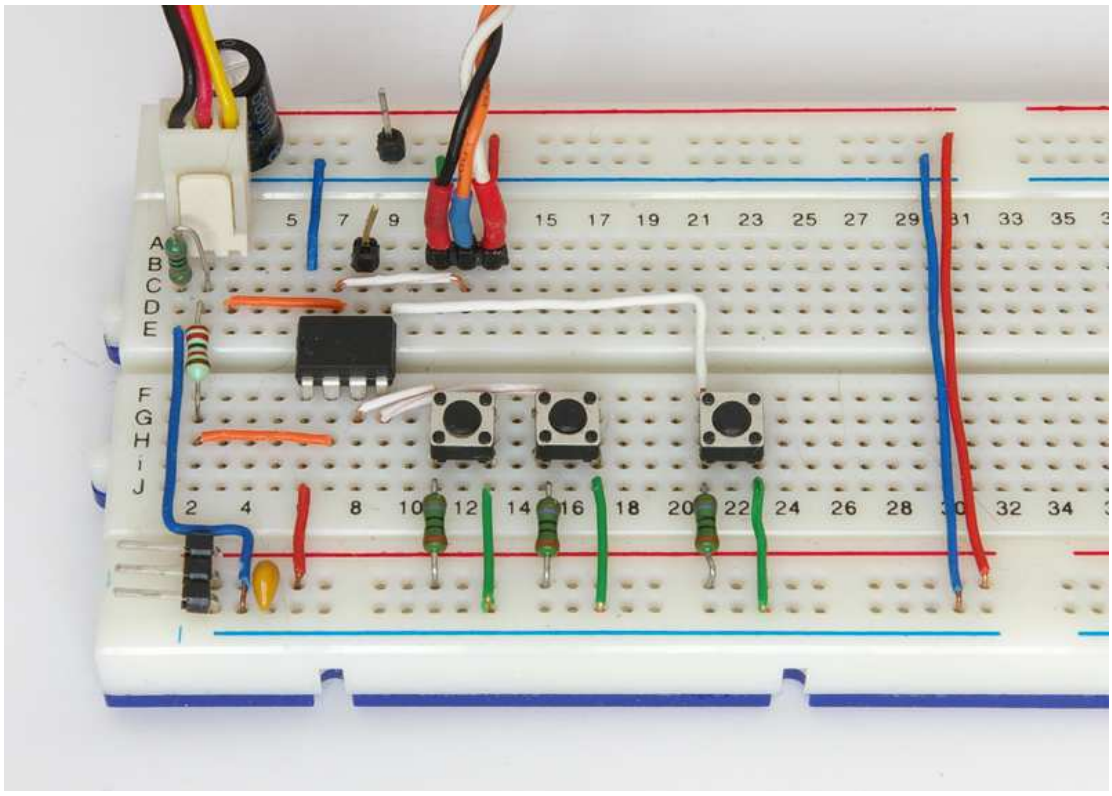
V klidu bude pravděpodobně LED zhasnutá, jakmile se však dotkneme nezapojeného vstupu prstem nebo vodivým předmětem, přenese se na něj rušení a LED se rozsvítí. Většinou dokonce stačí se jen přiblížit, dotek není nutný. Proto vždy musíme jednoznačně definovat úroveň na vývodu, který chceme použít jako vstup. Výrobce doporučuje odpor 4k7 až 10k, menší není na závadu, ale snažíme se, aby ani v případě, že by byl vývod přepnutý do režimu výstupu, nepřekročil proud povolenou mez. Tomu odpovídá asi tak 220Ω. Rezistor 1k sériově zapojený ke vstupu zlepšuje funkci a omezuje proudové špičky při nabíjení nebo vybíjení kapacity vstupu, nutný není a ve většině pokusů jej nebudeme používat. Zákmity tlačítek případně ošetříme programem.



Mikrokontroléry PICAXE řady M2 mají dokonce připravenou funkci Touch, které stačí na vstup připojit vodivou plošku a ta pak může fungovat jako bezkontaktní tlačítko, pro nějž lze kalibrovat citlivost. Vstup reaguje na přiblížení, takže ovládací tlačítka zařízení mohou být vytvořena na plošném spoji a překryta ochrannou fólií s popisem. Tím odpadnou mechanické spínací kontakty, prvek, který má největší problémy se životností a znečištěním.

Směřujeme k tomu, abychom vytvořili jednoduchý generátor pulzů. Připravíme si zapojení podle schématu. Budeme potřebovat další příkazy pro vytváření přesných pulzů a také se naučit, jak z mikrokontroléru vyčíst potřebné hodnoty.





IF … THEN … ELSE … ENDIF

Velmi často je potřeba program rozvětvit a podle hodnoty nějaké proměnné nebo stavu na vstupu pokračovat buď jednou nebo druhou cestou. K tomu slouží příkaz IF (*11) za nímž je

název proměnné (vstupu), 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 české 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, na ten se při psaní programu často zapomíná. Jde tedy napsat „IF w6>150 THEN HIGH 4 ELSE LOW 4 ENDIF“, což znamená: Je-li hodnota w6 větší než 150, nastav Pin 4 na H, v opačném případě nastav Pin 4 na L.

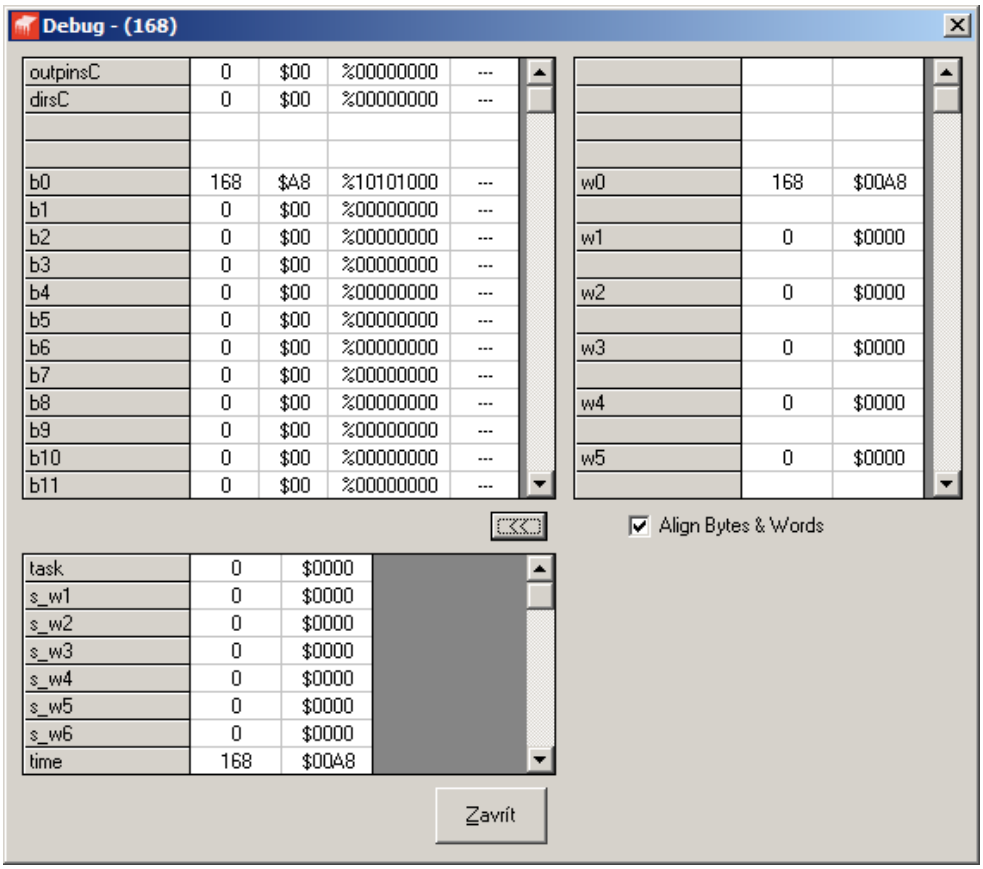
Čtení hodnot z mikrokontroléru, sériové přenosy dat

DEBUG

je univerzálním prostředkem pro ladění programů, někdy ale 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 nikdy 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)

Zapíšeme program s FOR cyklem, do něhož zadáme ladicí příkaz DEBUG. Abychom stíhali sledovat změny, smyčku zpomalíme asi tak na jeden průchod za sekundu. Po spuštění programu se otevře na obrazovce ladicí okno, v němž vidíme stavy všech proměnných v desítkové, šestnáctkové i binární soustavě i jako znakovou reprezentaci. Stačí tedy hodnotu kterou potřebujeme uložit do proměnné a podívat se na ni. V našem případě můžeme sledovat, jak se postupně mění řídící proměnná cyklu.

```
1       REM test debug - PICAXE 8M2
2       START:
3         for b0=0 to 255
4         debug
5         pause 1000
6         next
7         goto start
```



Stále budeme využívat připojený programovací kabel, ale naučíme se přenést z mikrokontroléru jen ty hodnoty, které v daném okamžiku potřebujeme, v podstatě si z počítače dočasně uděláme datový terminál se sériovou komunikací.

SERTXD

má za sebou v závorce seznam parametrů, které jsou brány jako ASCII znaky, a budou vyslány z mikrokontroléru do počítače. Přenos probíhá pevně danou rychlostí 4800 Bd, bez parity (N), 8 bitů s 1 stopbitem, zkráceně zapsáno 4800,n,8,1. Jednotlivé parametry se oddělují čárkou. Chceme-li poslat konkrétní zadaný text, uvedeme jej v uvozovkách ("tohle se pošle"), pokud chceme poslat výpis hodnoty proměnné, zapíšeme před její označení # (například #w0). Když uvedeme číslo, bude se brát jako kód znaku, který chceme poslat, stejně tak jen proměnná se bere jako znak, jehož kód odpovídá obsahu proměnné. Příklad: v proměnné b0 je hodnota 117. Použijeme-li #b0 vypíše se „117“, b0 vypíše znak „u“.

Rychlost přenosu 4800 Bd odpovídá základní frekvenci hodin 4 MHz, pokud ji změníme, úměrně se změní i rychlost přenosu. Snadno tedy můžeme nastavit spolu s vyšší rychlostí mikrokontroléru i přenos na 9600, 19200 nebo 38400 Bd. V našem zkušebním programu nahradíme příkaz DEBUG příkazem SERTXT, který vypíše nejdříve komentář „Průchod “, pak číslo průchodu cyklem z proměnné b0 a nakonec odřádkuje (znaky CR LF podle ASCII tabulky v dekadické reprezentaci 13 a 10).

```
sertxd("Průchod ",#b0,13,10)
```

Po přenesení do mikrokontroléru program poběží a bude vysílat data, my to ale neuvidíme. Musíme tlačítkem F8 nebo přes menu PICAXE - Terminál otevřít okno sériového terminálu a nastavit v něm správné parametry přenosu, hned potom se začnou naše hodnoty vypisovat. Protože se může přenášet jen to nejnutnější (řádově jednotky bytů), tento způsob výstupu hodnot z mikrokontroléru mnohem méně zpomaluje běh programu a dovoluje nám kdykoli bez dalších nákladů třeba na LCD displej vyčítat hodnoty.

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 a za ním je připojena hodinová frekvence v MHz, pro kterou má platit (pro 4 MHz není nezbytné ji uvádět). Způsob nastavení a nabídka rychlostí je v 2. dílu anglické dokumentace. Příklad: SEROUT 7,N2400_8,(b1) - pin7, kladné pulzy (klidová úroveň L) 2400 Bd při hodinách 8 MHz, výstup znaku podle obsahu proměnné b1.

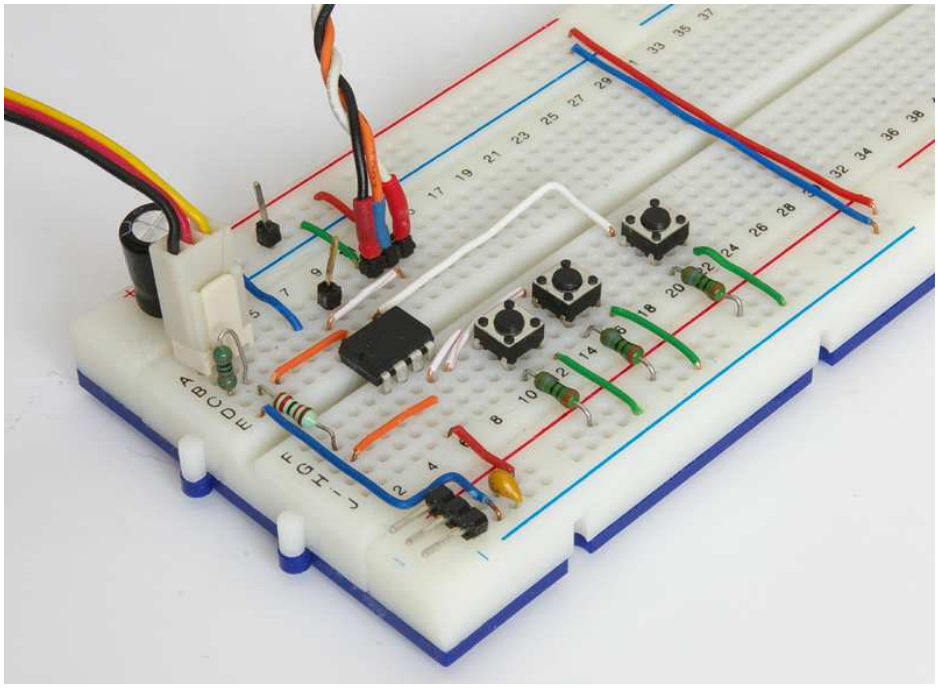
Zatímco příkaz SERTXT použijeme pro kontrolu programu a jako rychlý výstup místo tiskárny nebo displeje, SEROUT může pracovat s námi určeným pinem a použijeme jej ke komunikaci mezi mikrokontroléry, přenosem dat na displej a podobně. Jeden mikrokontrolér může mít připojených několik sériových linek k různým zařízením. Nemá smysl zkoušet terminálem v PC posílat data do mikrokontroléru přes systémový (programovací) kabel, mikrokontrolér by je mohl vzít jako nový program a udělat „čert ví co“. Pokud potřebujeme posílat data z PC do mikrokontroléru, vyhradíme tomu samostatný pin a použijeme příkaz SERIN.

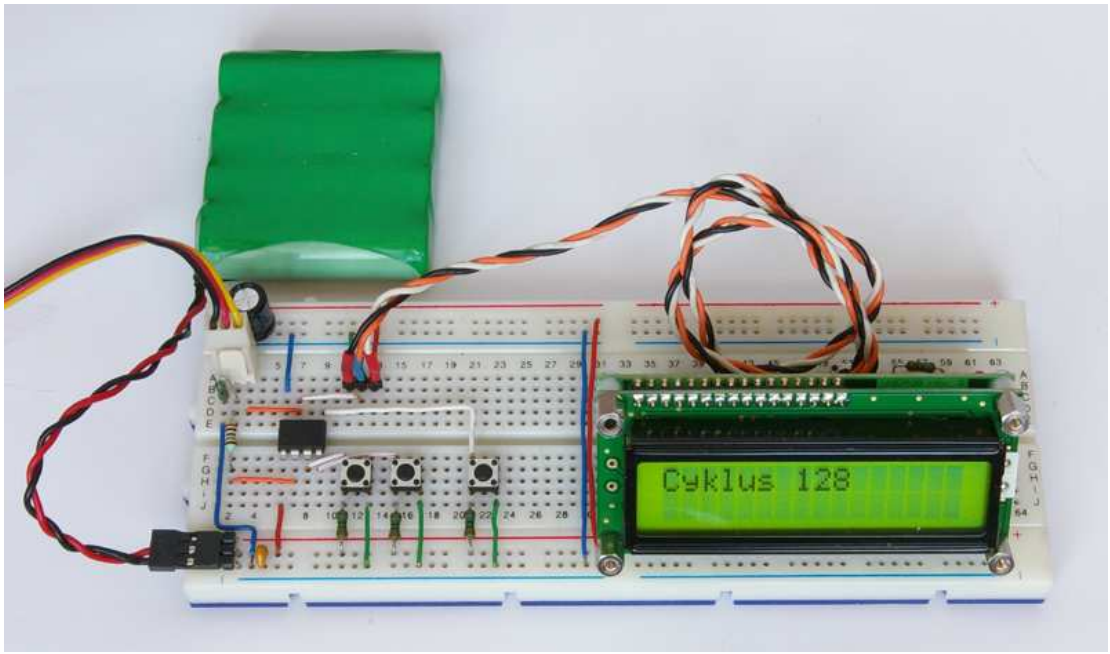
SERIN

je příkazem pro příjem sériových dat. Jeho prvním parametrem je číslo pinu, z něhož se čte, pak následuje způsob přenosu a rychlost, sekvence, na kterou se má čekat (!) a pak seznam proměnných, které se postupně budou přicházejícími daty plnit. 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, opět je přes znak podtržítko připojena frekvence hodin (pro 4 MHz není nutné). Způsob nastavení a nabídka rychlostí je v 2. dílu anglické dokumentace. Příklad: SERIN 1,N2400_16,(“ABC”),b1- čte z pinu 1 kladné pulzy 2400 Bd při hodinách 16 MHz, čeká až přijde text ABC, pak nejbližší následující byte uloží do proměnné b1.

Uvedené příkazy nám už umožní připojit k mikrokontroléru asi nejnázornější výstupní zařízení, LCD displej. Tím se také zbavíme potřeby trvalého propojení s PC. Většina levných

LCD displejů vyžaduje předávání dat prostřednictvím většího počtu vodičů. To není pro nás výhodné, protože se tím obsadí mnoho z vývodů mikrokontroléru, typ 08M2 by na to svými možnostmi vůbec nestačil. Existují ale i sériově ovládané displeje, jeden z nich, SIC1602AYPLEB20 ze sortimentu už zmíněné české firmy na internetových stránkách www.*snailshop.cz* použijeme. Tabulka řídících příkazů je uvedena v dokumentaci displeje. Tento displej si vystačí s jedním vývodem mikrokontroléru. Analogicky by bylo možné připojit i displeje komunikující přes rozhraní I2C. V následujících programech je vždy možné obejít použití displeje a přepsat výstup tak, aby směřoval na sériový terminál PC (a leckdy je to vede k jednoduššímu zápisu), samozřejmě, musí se změnit formátování výstupu. Vyzkoušíme si ovládání displeje připojeného k označeným svorkám, zobrazíme si pořadí prováděného cyklu. Propojky jsou nastaveny na 2400 Bd a invertovanou polaritu.





```
1       REM test displeje - PICAXE 8M2
2       START:
3         for b0=0 to 255
4           serout 0,N2400,($FE,$01)              ;inicializace LCD
5           serout 0,N2400,("Cyklus ",#b0)                ;výpis textu a B0
6           pause 1000                            ;zpomalení
7           next
8         goto start
```

Zatím jsme všechny programy uzavírali skokem na začátek, tedy do nekonečné smyčky. Pokud má zapojení začít pracovat s připojením napájení a s jeho odpojením končit, je to tak obvyklé, nicméně někdy je potřeba program ukončit v definovaném místě a okamžiku, k tomu slouží dva příkazy, END a STOP.

END

způsobí kdykoli v průběhu programu jeho ukončení a přechod mikrokontroléru do stavu „spánku“ s nepatrnou spotřebou. Tento režim zastaví i činnost interního časovače. Obnova činnosti programu je možná jen vypnutím a zapnutím napájecího napětí, resetem (pokud má daný mikrokontrolér reset vyvedený ven) nebo opětovným zavedením programu z PC.

STOP

funguje podobně, ale mikrokontrolér nepřechází do režimu s minimální spotřebou, takže příkazy závislé na interním časovači, které poznáme později (SERVO, PWMOUT,..) probíhají plynule dál.

Vstup a výstup přesných pulzů

Stále směřujeme k tomu, abychom vytvořili generátor pulzů. Zatím jsme generovali pulzy jen programem, můžeme k tomu ale použít i vnitřní časovače mikrokontroléru, což je mnohem přesnější. Naučíme se také měřit délku pulzu a spočítat pulzy, které přijdou v určené době. S touto tématikou souvisí i příkazy sloužící k vytváření zvuku, což není nic jiného než přesné frekvence, ty si ale probereme později.

PULSOUT

má dva parametry (*18). První udává výstupní Pin, s nímž se bude pracovat, druhý délku (jednoho) pulzu, který mikrokontrolér vygeneruje. Pulz je opačný vůči úrovni, v níž je pin na počátku, takže příkaz můžeme použít jak pro kladné tak záporné pulzy. Jednotkou pro délku pulzu je 10 µs při normálním hodinovém kmitočtu 4 MHz (mění se s frekvencí hodin), při 32 MHz je to 1,25 µs. Příklad: PULSOUT 1,4 … generuje na pinu 1 při 32 MHz pulz dlouhý 5 µs.

Vyjdeme z připraveného zapojení. Reálně změřené pulzy na výstupu mikrokontroléru byly při 4 MHz o 4 µs delší, než by teoreticky měly být, při 32 MHz o 0,5 µs. Tento rozdíl by

případně šlo kompenzovat, zatím se tím ale zabývat nemusíme. Nejdelší pulz, který můžeme tímto způsobem vytvořit, je 0,65 s při 4 MHz. Program, který vytvoří nejkratší možné pulzy opakované po 0,1 s se vejde na jeden řádek:

```
ST: pulsout 1,1 pause 100 goto ST
```

PULSIN

měří délku pulzu na zadaném vstupu. 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 daných jednotkách shodných s příkazem PULSOUT. Proměnná může být i typu byte, lépe je ale používat proměnnou typu word. Jestliže pulz nepřijde, skončí měření po 0,65 s (při 4 MHz) a v proměnné vrátí číslo 0, tak můžeme otestovat, že měření bylo neúspěšné. Příklad: 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 (*18).

Příkaz je v podstatě opakem předchozího a také chyba odpovídá. Jako pulz dlouhý 1 jednotku se zaznamená i pulz 10x kratší, než je jednotka času, jinak ale měření funguje slušně přesně. Program, který můžeme považovat za nejjednodušší měřič délky pulzu v jednotkách 10 μs a výsledek posílá do PC, vypadá takto:

```
ST: pulsin 1,1,w0 debug goto ST
```

COUNT

má tři parametry, první udává vstupní pin, s nímž se bude pracovat, druhý dobu v ms, po kterou se budou pulzy počítat, a třetí je proměnná (lépe typu word), do níž se výsledný počet uloží (*6). Jednotka času se mění podle kmitočtu hodin. Příklad: COUNT 1,1000,w0 sleduje po dobu 1 sekundy pin 1 a vrátí počet pulzů, které přišly (tedy přímo frekvenci v Hz).

Vstupní pulzy by měly být alespoň 40 μs dlouhé (při 4 MHz), optimální je, když mají střídu 1:1. I když podle dokumentace je nejvyšší spolehlivě měřitelná frekvence 25 kHz, naměřené hodnoty podle mých zkušeností odpovídají přesně do 10 kHz, výš už roste chyba a PICAXE měří méně, než by měl, jako by některé pulzy vypouštěl. Při hodinách 32 MHz se teoreticky rozsah rozšíří až do meze naplnění proměnné (65 kHz), ale v praxi měření bezchybně fungovalo jen do 50 kHz. Nejjednodušší program pro měření frekvence a čtení hodnoty přes PC tedy můžeme zapsat jako jadnoduše:

```
ST: count 1,1000,w0 debug goto ST
```

nebo

```
setfreq m32
ST: count 1,8000,w0 debug goto ST
```

Generátor pulzů

Nyní už můžeme předchozí střípky poskládat do celku, napsat program pro náš pokusný pulzní generátor a zbavit se i výstupu posílaného po sériové lince do PC. Potřebné údaje budeme mít na displeji.

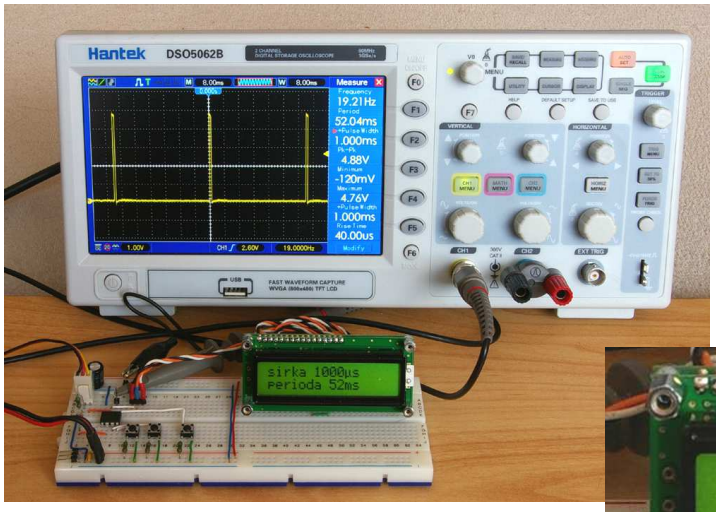
```
1     REM Generátor pulzu - PICAXE 08M2
2     setfreq m8                              ;kmitocet 8 MHz
3     w0=200                                  ;pocátecní šírka 1 ms (*0,005 ms)
4     w1=12                                   ;pocátecní perioda 10 ms (*0,5 ms)
5     pause 200                               ;cas na nábeh LCD
6     serout 0,N9600_8,($FE,$01)              ;vymazání LCD
7     pause 200                               ;cas na vymazání LCD
8     serout 0,N9600_8,("Pulzni generator",$FE,$C0)
9     serout 0,N9600_8,(" verze 1 ")          ;hlavicka
10    pause 4000                              ;zobrazení hlavicky 2s
11    serout 0,N9600_8,($FE,$01)              ;vymazání LCD
12    pause 200                               ;cas na vymazání LCD
13    serout 0,N9600_8,("sirka")              ;popisy
14    serout 0,N9600_8,($FE,$C0,"perioda ") ;popisy
15    gosub vypis                             ;výpis pocátecních hodnot pulzu
16
17    start:                                  ;generování pulzu a obsluha TL
18      if b20=4 then pulsout 1,w0 pause w1 endif
19                                            ;pulzy jen když se nemení nastavení
20      b20=b20-1 min 4                       ;hlídání zmeny nastavení
21      if pin2=0 then                        ;prepínání mezi šírkou a periodou
22        if pin4=0 then dec w0 gosub vypis endif      ;šírka -
23        if pin3=0 then inc w0 gosub vypis endif      ;šírka +
24        else
25        if pin4=0 then dec w1 gosub vypis endif      ;perioda -
26        if pin3=0 then inc w1 gosub vypis endif      ;perioda +
27        endif
28      goto start                            ;konec smycky hlavního programu
29
30    vypis:                                  ;podprogram pro výstup na LCD
31      w0=w0 min 1 max 2800                  ;meze pro šírku v 0,005ms
32      w1=w1 min 1 max 4000                  ;meze pro periodu v 0,5ms
33      w3=w0/2                               ;prevod na 0,01ms
34      w5=w0/100                             ;korekce periody podle šírky
35      w4=w1+w5/2+3                          ;prevod periody na 1 ms a korekce
36      serout 0,N9600_8,($FE,$86)            ;umístení na 1. rádku
37      serout 0,N9600_8,(#w3,"0",234,115," ")       ;výpis
38      serout 0,N9600_8,($FE,$C8)            ;umístení na 2. rádku
39      serout 0,N9600_8,(#w4,"ms ")          ;výpis
40      b20=5                                 ;nastavení promenné hlídání zmeny
41      return                                ;návrat z podprogramu
```

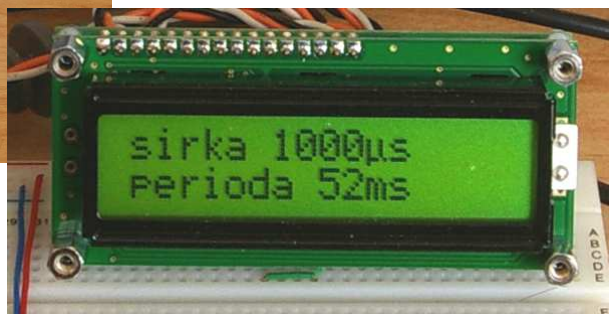
První část programu se stará i inicializaci proměnných na počátku běhu, o výpis hlavičky a prvních hodnot na displeji. Má smysl si všimnout čekání na počátku a také po každém smazání displeje, doba potřebná na smazání LCD je totiž tak dlouhá, že bez prodlevy by první následující byty displeji „utekly“. Proč byl použit kmitočet 8 MHz? Protože je to nejnižší kmitočet, při němž mikrokontrolér umí generovat přenosovou rychlost 9600 Bd, potřebnou pro displej. Pokud by byla použita nižší rychlost 2400 Bd, 4x víc by to zdržovalo běh programu.

V době, kdy dochází k sériovému přenosu, není možné generovat pulzy, a to dokonce ani kdyby byl použit multitasking. Z tohoto důvodu se v době změny parametrů tlačítky pulzy vypínají. Celé generování pulzů je na jediném řádku, jeho šířku vytvoří časovač mikrokontroléru, zbytek do periody je už časován programem. Proto je důležité, aby program běhal stále stejně rychle stejnou cestou ve smyčce, což se také děje. Zbytek střední části programu tvoří obsluha tlačítek, jedním se přidává hodnota, druhým ubírá, třetí tlačítko (nebo přepínač) určuje, zda se bude měnit aktivní šířka pulzu nebo perioda.

Třetí část programu je volána jako podprogram (viz dále) a stará se o zobrazení na LCD a také omezení rozsahu obou nastavovaných hodnot. Protože se pracuje s 2x vyšším kmitočtem než je základní, musí se na celistvé desítky mikrosekund hodnota převést, stejně tak se převádí i perioda na ms. Konstanta kompenzuje dobu, kterou mikrokontrolér stráví proběhnutím programem, o to kratší musí být čekání určují neaktivní část periody. Poslední kompenzace dorovnává periodu podle šířky aktivního pulzu. Aritmetické operace se vyhodnocují vždy postupně zleva doprava, přednosti operací mikrokontrolér nedodržuje!

Výsledný generátor dělá podle měření šířku pulzu od 7 µs do 14 ms (změnou parametru možno rozšířit do 300 ms) a s periodou 4 ms až 2 s. V dalším postupu by bylo možné především zvýšit pohodlí ovládání progresivní rychlostí běhu autorepeatu, šlo by třeba lépe zarovnávat vypisované údaje, zaznamenávat do EEPROM naposledy nastavené parametry, předělat zadávání hodnot na číslicový vstup z klávesnice PC, což umí PICAXE přímo obsluhovat atd. I tak je myslím vidět, že program pro použitelný generátor není nijak dlouhý a pokud jde jen o napsání jednoho konkrétního zdroje signálu bez dalšího ovládání tlačítky a displeje, je to záležitost na tři řádky a jednu minutu. Je zkrátka rychlejší takový generátor napsat než ho postavit. To nejpracnější je zajistit uživatelský komfort a uhlazený projev na displeji.





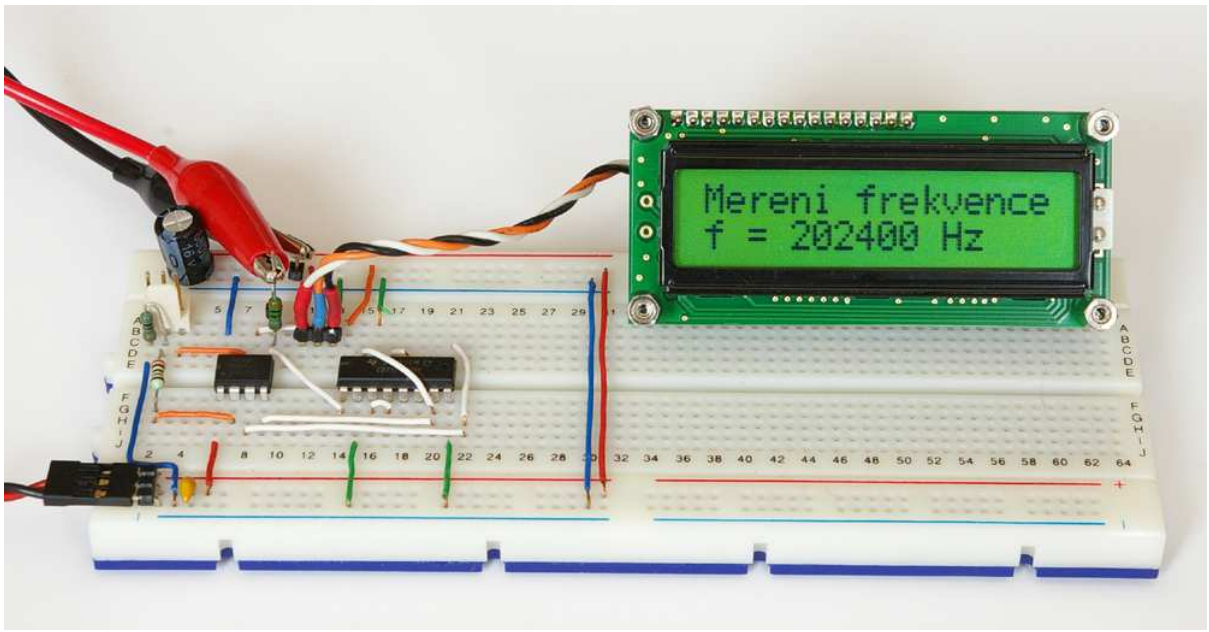
GOSUB ... RETURN

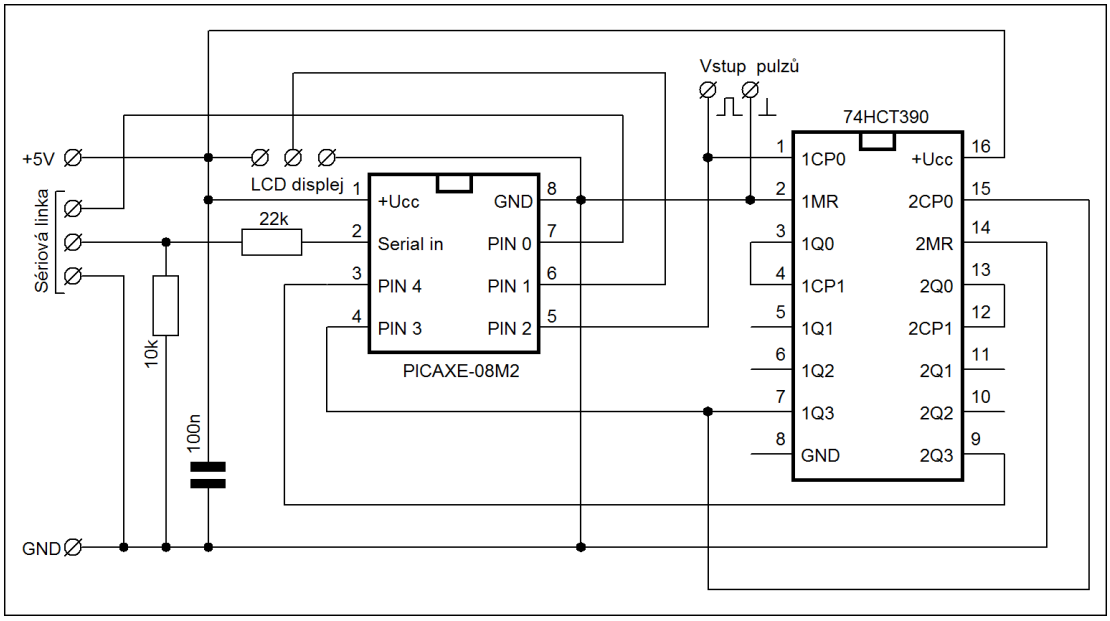
GOSUB je příkaz skoku, ale na rozdíl od GOTO si pamatuje, odkud byl skok proveden, a když program narazí na povel RETURN, vrátí se zpátky za příkaz, z něhož k odskoku došlo. Tu část programu nazývanou podprogram, jež se má na zavolání GOSUB vykonat, musíme samozřejmě označit návěštím. Počet použití příkazů GOSUB je omezen, viz (*9).

Malý měřič frekvence

Postavit a naprogramovat jednoduchý čítač 1 Hz - 1 MHz také není náročné, nebudeme k tomu ani potřebovat nové příkazy. Samotný mikrokontrolér 08M2 má až čtyři použitelné vstupy a měří velmi dobře frekvenci asi do 10 kHz, při vyšších se projevuje výrazná chyba. Zapojení využívá dvojnásobného dekadického děliče 74HCT390 k rozšíření základního rozsahu 100x. K počítání pulzů po dobu jedné sekundy se používá přímo příkaz COUNT, nijak se neřeší vstupní obvody a ochrany, jde pouze o příklad aplikace procesoru. Vstupu procesoru nevadí, pokud na něj přivedeme vyšší kmitočet, zhruba do 50 kHz čte, i když s chybou. Toho se využívá a tři vstupy vnitřním přepínáním nahrazují multiplexer. Nevýhodou je, že pokud z nízkých frekvencí (do 10 kHz) prudce zvýšíme kmitočet třeba k 1 MHz, může dojít k tomu, že procesor bude načítat chybný počet pulzů a nepřepne rozsah. Po zapnutí, kdy začíná číst frekvenci přes dva děliče, toto nastat nemůže. Rezistor, přes který je připojen vstupní signál na fotografii, není zakreslen ve schématu, protože obecně není potřeba, tlumil jen záporné překmity u použitého zdroje signálu.

```
1       REM Malý meric frekvence - PICAXE 08M2
2       setfreq m8                                  ;hodiny 8 MHz
3       b2=4                                        ;na zacátku merení pres 2 delice
4       pause 200                                   ;cas na nábeh LCD
5       serout 1,N9600_8,($FE,$01)                  ;vymazání LCD
6       pause 200                                   ;cas na vymazání LCD
7       serout 1,N9600_8,("Mereni frekvence")       ;hlavicka
8
9       cyklus:                                     ;merení a zobrazení hodnot
10       count b2,2000,w0                           ;cítání 1s
11       serout 1,N9600_8,($FE,$C0)                 ;umístení na 2.r.
12       serout 1,N9600_8,("f = ",#w0)              ;popis a hodnota
13       if b2=4 then serout 1,N9600_8,("00") endif    ;*100
14       if b2=3 then serout 1,N9600_8,("0") endif     ;*10
15       serout 1,N9600_8,(" Hz ")                  ;výpis jednotek
16       if w0<990 then dec b2 endif                ;o delic mín
17       if w0>10100 then inc b2 endif              ;o delic víc
18       if b2=1 then let b2=2 endif                ;nejde mín delit
19       if b2=5 then let b2=4 endif                ;nejde víc delit
20       goto cyklus
```





V dalším kroku by bylo třeba možné udělat na nízkých rozsazích automaticky přepínanou dobu měření na 10 s a zpřesnit údaj (i když už to překračuje možnosti stability vnitřního časového normálu) nebo modifikovat program pro měření periody nebo střídy.

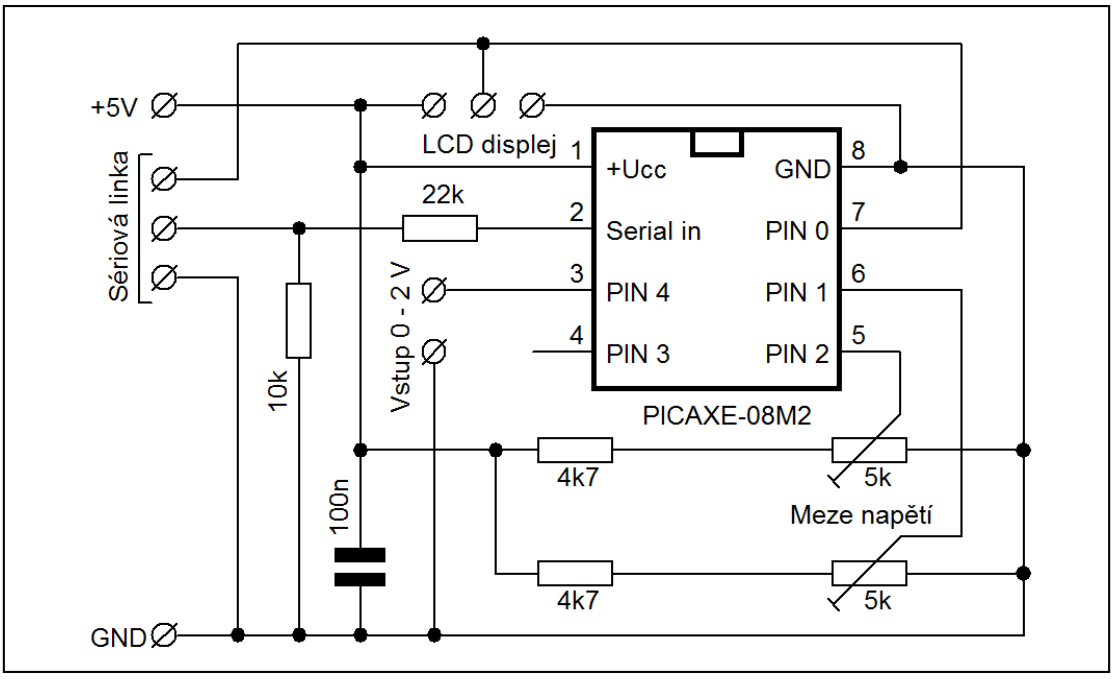
A/D převodníky

Mikrokontroléry PICAXE jsou vybaveny A/D převodníky, takže mohou bez dalších součástek číst napětí na některých ze svých vstupů, konkrétně typ 08M2 má tři A/D vstupy na pinech 1, 2 a 4. Převod může fungovat jako osmibitový s ukládáním výsledku do jednobytové

proměnné nebo jako desetibitový s ukládáním výsledku do proměnné typu word. V nejjednodušší podobě převod pracuje v mezích tvořených zemí a napájecím napětím mikrokontroléru, můžeme ale také programově připojit vnitřní referenční zdroje napětí 4,096 V, 2,048 V nebo 1,024 V, případně přivést vnější referenční napětí na vývod procesoru. Budeme předpokládat, že napájecí napětí mikrokontroléru je přesně 5,12 V a pokusíme se nejprve vytvořit jednoduchý voltmetr, pak voltmetr přepojíme na přesnou referenci napětí a nakonec si postavíme voltmetr, který hlídá a signalizuje překročení nastaveného minimálního a maximálního napětí a pro rychlou orientaci v rámci nastavených mezí zobrazuje bargraf.

READADC, READADC10

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 mikrokontroléru), převede na osmibitové číslo a výsledek uloží do proměnné b0. (*20) READADC10 pracuje stejně, dělá desetibitový převod a proměnná musí být typu word.



Zapojení využívá jako hlavní vstup měřeného napětí pin 4, piny 1 a 2 budou později sloužit k nastavení minimálního a maximálního hlídaného napětí. Tyto vývody jsou pevně určeny tím, že jsou to vstupy A/D převodníků. Protože už žádný volný výstupní pin nezbyl, vrátíme se k použití pinu 0 současně pro komunikaci s PC při programování a současně jako výstupu pro displej, v podstatě tedy nemůžeme při ladění programu používat příkaz DEBUG. V programu stojí za zmínku snad jen použití operátoru zbytku po dělení (modulo, //) a nutnost zarovnání výpisu setin V na dvě místa, protože jinak by při poklesu napětí po údaji 0,10 V následovalo 0,9 V.

```
1       REM Voltmetr 1 (napájení 5,12V) - PICAXE 08M2
2       setfreq m8                              ;hodiny 8 MHz
```

```
3       pause 200                                   ;cas na nábeh LCD
4       serout 0,N9600_8,($FE,$01)                  ;vymazání LCD
5       pause 200                                   ;cas na vymazání LCD
6
7       cyklus:                                     ;merení a zobrazení jednoho napetí
8         readadc10 4,w0                            ;napetí z pin 4 do w0
9         w0=w0/2                                   ;prevod na 0,01V
10        w1=w0/100                                 ;celé V do w1
11        w2=w0//100                                ;setiny V do w2
12        serout 0,N9600_8,($FE,$80)                ;od zacátku 1. rádku
13        serout 0,N9600_8,("U = ",#w1,".")         ;popis, jednotky, DT
14        if w2<10 then serout 0,N9600_8,("0") endif          ;dorovnání
15        serout 0,N9600_8,(#w2," V ")              ;setiny V
16        pause 500                                 ;zpomalení na 2 merení /s
17        goto cyklus
```

FVRSETUP

Jeden parametr vyjadřuje, jaká vnitřní reference se má použít, možnosti jsou FVR1024, FVR2048 nebo FVR4096. Parametr OFF referenci vypíná. Je třeba si uvědomit, že mikrokontrolér neumí vnitřní napětí zvětšit, takže referenci 4,096 V lze použít jen při napájení napětím 5 V (nebo nepatrně vyšším).

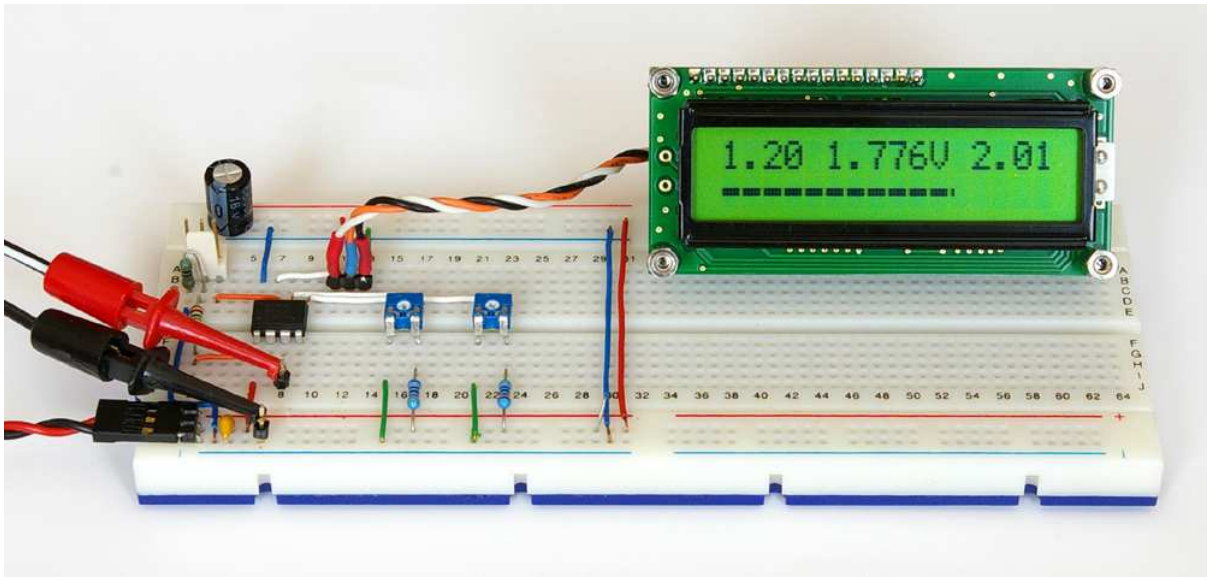
ADCCONFIG

Jeden parametr určuje, jak bude použito referenční napětí pro A/D převodníky. V podstatě stačí nastavit 3 bity. Bit 2 určuje vztažnou úroveň. Je-li bit 2 = 1, bude použit externí vstup referenčního napětí, pro náš procesor 08M2 ale tato možnost dána není. Je-li bit 2 = 0, bude vztažnou úrovní zem napájení procesoru. Bity 1 a 0 určují kladné referenční napětí, při kombinaci 00 (nebo není-li určeno) se bere napájecí napětí procesoru, 01 není povolená kombinace, 10 znamená referenční napětí z externího vstupu (pro náš procesor to je možné a byl by to byl pin 1) a kombinace 11 vyjadřuje, že se vezme napětí z vnitřního referenčního zdroje, velikost napětí je přitom určena povelem FVRSETUP.

Modifikovaný program musí mít v inicializační části povely FVRSETUP FVR2048 a ADCCONNFIG %011, kromě toho se změní přepočet, protože základní rozlišení nyní bude 0,002 V. Převodníky také trpí určitou nepřesností, zejména v okolí nuly (spodního okraje reference) a aditivní chybou, kterou můžeme pro konkrétní procesor po empirickém zjištění z větší části kompenzovat (konstanta korekce).

```
1       REM Voltmetr 2 (reference 2,048V) - PICAXE 08M2
2       setfreq m8                                  ;hodiny 8 MHz
3       adcconfig %011                              ;vnitrní referencní zdroj
4       fvrsetup fvr2048                            ;nastavení reference 2,048 V
5       symbol korekce=62                           ;korekce pro prevodník
6       pause 200                                   ;cas na nábeh LCD
7       serout 0,N9600_8,($FE,$01)                  ;vymazání LCD
```

```
8       pause 200                                   ;cas na vymazání LCD
9
10      cyklus:                                     ;merení a zobrazení jednoho napetí
11        readadc10 4,w0                            ;napetí z pin 4 do w0
12        w0=w0*2+korekce                           ;prevod na 0,002V
13        w1=w0/1000                                ;celé V do w1
14        w2=w0//1000                               ;tisíciny V do w2
15        serout 0,N9600_8,($FE,$80)                ;od zacátku 1. rádku LCD
16        serout 0,N9600_8,("U = ",#w1,".")         ;popis, jednotky, DT
17        if w2<100 then serout 0,N9600_8,("0") endif         ;dorovnání
18        if w2<10 then serout 0,N9600_8,("0") endif          ;dorovnání
19        serout 0,N9600_8,(#w2," V ")                        ;mV
20        pause 500                                 ;zpomalení na 2 merení /s
21        goto cyklus
```



Poslední verze voltmetru využívá všechny tři DA převodníky mikrokontroléru 08M2. Dva z nich slouží pro nastavení mezních hodnot pomocí trimrů. Stejně tak by bylo možné nastavovat mezní hodnoty digitálně třeba dvojicí tlačítek, ale podobným způsobem často šetříme vstupy. Potřebujeme-li například k ovládání přepínač 1 z 8, se dá výhodně použít AD převodník a na přepínač připojit napěťový dělič z rezistorů. Změřenou hodnotu napětí pak mikrokontrolér interpretuje jako volbu jedné z možností na přepínači.

```
1       REM Voltmetr 3 (s bargrafem) - PICAXE 08M2
2       setfreq m8                                  ;hodiny 8 MHz
3       adcconfig %011                              ;vnitrní referencní zdroj
4       fvrsetup fvr2048                            ;nastavení reference 2,048 V
5       symbol korekce=62                           ;korekce pro prevodník
6       pause 200                                   ;cas na nábeh LCD
7       serout 0,N9600_8,($FE,$01)                  ;vymazání LCD
8       pause 200                                   ;cas na vymazání LCD
9       serout 0,N9600_8,($FE,$40)                  ;uživatelské znaky
```

```
10    serout 0,N9600_8,($00,$00,$00,$00,$00,$00,$00,$00)
11    serout 0,N9600_8,($00,$00,$00,$10,$10,$00,$00,$00)
12    serout 0,N9600_8,($00,$00,$00,$18,$18,$00,$00,$00)
13    serout 0,N9600_8,($00,$00,$00,$1C,$1C,$00,$00,$00)
14    serout 0,N9600_8,($00,$00,$00,$1E,$1E,$00,$00,$00)
15    serout 0,N9600_8,($00,$00,$00,$1F,$1F,$00,$00,$00)
16
17    cyklus:                                ;merení a zobrazení napetí
18      readadc10 4,w0                       ;merené napetí z pin 4 do w0
19      w0=w0*2+korekce                      ;prevod na 0,002V
20      w1=w0/1000 w2=w0//1000               ;celé V do w1,tisíciny do w2
21      readadc10 1,w4                       ;minimální napetí z pin 1 do w4
22      w4=w4*2                              ;prevod na 0,002V
23      w5=w4/1000 w6=w4//1000/10            ;celé V do w5,setiny do w6
24      readadc10 2,w7                       ;maximální napetí z pin 2 do w7
25      w7=w7*2                              ;prevod na 0,002V
26      w8=w7/1000 w9=w7//1000/10            ;celé V do w8,setiny do w9
27      if w0<w4 or w0>w7 then               ;hlídání prekrocení mezí
28        serout 0,N9600_8,($FE,$E8)         ;bliknutí displejem
29        pause 100 serout 0,N9600_8,($FE,$E9)
30        else pause 200 endif               ;podobné zpomalení merení
31      serout 0,N9600_8,($FE,$80)           ;od zacátku 1. rádku LCD
32      serout 0,N9600_8,(#w5,".")           ;popis, jednotky, DT
33      if w6<10 then serout 0,N9600_8,("0") endif          ;dorovnej
34      serout 0,N9600_8,(#w6," ")           ;výpis mV
35      serout 0,N9600_8,(#w1,".")           ;popis, jednotky, DT
36      if w2<10 then serout 0,N9600_8,("0") endif          ;dorovnej
37      if w2<100 then serout 0,N9600_8,("0") endif         ;dorovnej
38      serout 0,N9600_8,(#w2,"V ")                         ;výpis mV
39      serout 0,N9600_8,(#w8,".")                          ;popis, jednotky, DT
40      if w9<10 then serout 0,N9600_8,("0") endif          ;dorovnej
41      serout 0,N9600_8,(#w9)                              ;výpis mV
42      serout 0,N9600_8,($FE,$C0)           ;od zacátku 2. rádku LCD
43      if w0>w7 then let w0=w7 endif        ;ošetrení mezí
44      if w0<w4 then let w0=w4 endif        ;ošetrení mezí
45      w10=w0-w4                            ;délka cáry
46      w11=w7-w4                            ;délka celého bargrafu
47      w12=w10*40/w11*2                     ;prevod na císlo 0-80
48      b26=w12/5                            ;pocet plných znaku
49      if b26>0 then                        ;výpis plných znaku
50      for b27=1 to b26 serout 0,N9600_8,(05) next endif
51      b26=w12//5                           ;výpocet zbytku
52      serout 0,N9600_8,(b26)               ;výpis částecného znaku
53      for b27=1 to 16 serout 0,N9600_8,(00) next          ;mazání
54      goto cyklus
```

V programu stojí několik věcí za povšimnutí. V závěru inicializace je do displeje ukládá šest uživatelských znaků, z nichž se později sestaví po bodu rostoucí bargraf. Znaky jsouv podstatě prázdné pole, čárka na 1 bod zleva, čárka na 2 body zleva … až čárka v plné šíři 5

bodů. Potřebné řídící kódy jsou v dokumentaci k displeji, to se týká i možnosti zapnout nebo vypnout podsvícení displeje. Výrazné blikání se zde využívá k signalizaci překročení mezních hodnot.

Nenápadná, ale důležitá věc je na 6. a 5. řádku od konce, kde je pomocí IF ošetřen stav, kdy by se mohly parametry FOR cyklu prohodit, vyšlo by FOR b27=1 to 0. V naprosté většině jazyků by se tělo FOR cyklu v takovém případě vůbec nevykonalo, ale Basic PICAXE vždy projde tělo FOR cyklu alespoň jednou. Domnívám se, že jde o chybu, v každém případě s tím musíme počítat. Než se dostaneme dál k opačnému převodu D/A, podíváme se na řízení motorů, brzy bude jasné, proč.

Řízení motoru

Jednou z úloh, kterou je třeba dost často mikrokontrolérem řešit, je plynulé ovládání otáček stejnosměrného elektromotoru. Řízení krokových a střídavých motorů lze najít v aplikační příručce. Pro ovládání otáček (výkonu) motorů se používá pulzně šířková modulace jejich napájecího napětí (PWM). K tomu jsou připraveny 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 mikrokontrolér vykonat.

Příkaz PWM neběží na pozadí, v jeho průběhu procesor nedělá nic jiného. Probíhá jednorázově jen s omezeným počtem opakování, musí se volat pravidelně. Podléhá samozřejmě volbě hodinové frekvence procesoru. Příkaz negeneruje pulzy s konstantní periodou a proměnným plněním, ale frekvence se mění od 134 Hz do 17,1 kHz, délka kladného (nebo záporného) pulzu je konstantní 29 µs. Počet cyklů neodpovídá ve smyslu počtu period zadaného signálu, ale násobků doby přibližně 7,4 ms, jakýchsi vnitřních cyklů. Budeme-li používat parametry v tom smyslu, jak jsou uvedeny, motor se jimi řídit úspěšně dá, jinak je ale tento příkaz používán zejména k „falešnému“ D/A převodu na napětí.

PWMOUT

je obecnější a používanější příkaz než předchozí. Má také tři parametry oddělené čárkou, první určuje pin, druhý jednobytový parametr udává periodu pulzů a třetí v rozsahu 0 - 1023 odpovídá činiteli plnění, výpočet je ale složitější.

Protože jsou pulzy tentokrát generované vnitřním časovačem, dá se u PICAXE 08M2 použít pouze pin 2, pulzy však běží na pozadí, mikrokontrolér může během řízení motoru vykonávat jiné činnosti. Příkaz PWMOUT nelze použít současně s příkazem SERVO (viz dále), 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, je

potřeba si na to zvyknout. Zadáme-li například druhý parametr 99, bude perioda PWM signálu 250*4*(99+1) = 100000 ns = 0,1 ms (frekvence 10 kHz). Nejkratší možná perioda PWM pulzů je 250*4*(1+1) = 2000 ns = 2 mikrosekundy (500 kHz). Použijeme-li na místě druhého parametru hodnotu 0, pulzy vypneme.

Třetí parametr udává délku aktivního pulzu (výstup je v hodnotě logická 1). Spočítáme ji z hodnoty zadaného parametru * perioda oscilátoru procesoru (250 ns). Při zadání 200 bude délka pulzů odpovídat 250*200 = 50000 ns = 50 µs. Pro vytvoření střídy 1:1 při 10 kHz musíme tedy zadat příkaz PWMOUT 2,99,200 .

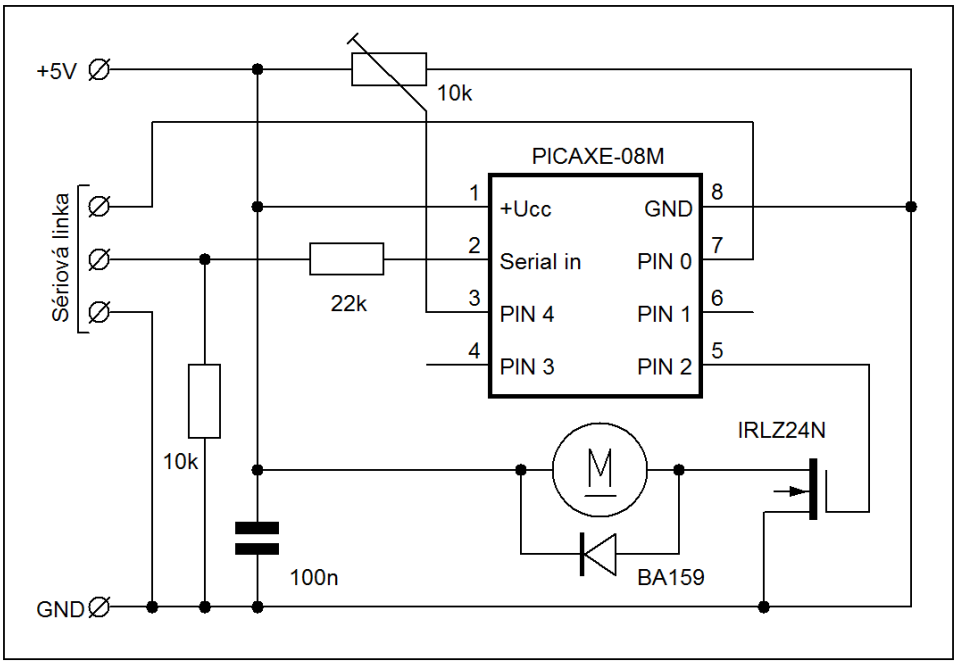
Vždy volíme nejdříve periodu PWM signálu, protože délka aktivního pulzu nemůže být libovolná a vychází z ní. Hodnota třetího parametru nesmí přesáhnout (přibližně) čtyřnásobek druhého parametru, jinak bude výstup trvale sepnutý a žádné pulzy na něm nebudou (tím bychom zadali aktivní pulz delší, než je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

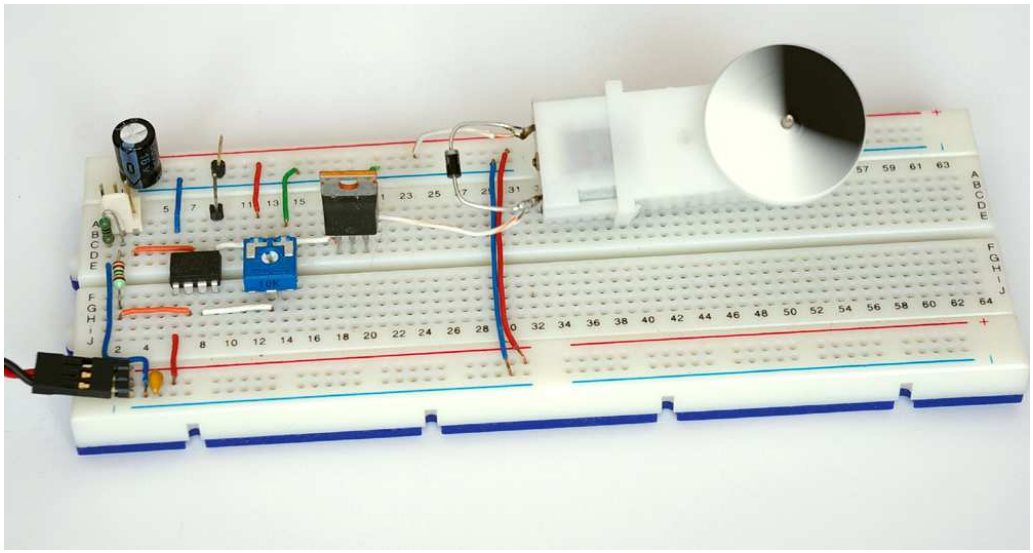
Chceme-li například vytvořit řízení motoru s frekvencí PWM 4 kHz (250 µs), což je poměrně často používaná frekvence pro malé stejnosměrné motory, pak vychází hodnota druhého parametru 250000/250/4-1 = 249 a k tomu odpovídající rozsah třetího parametru (aktivního pulzu) pro ovládání výkonu od 1 do 250000/250 = 1000. Zadáme-li nulu, přesně podle významu se pulzy vypnou, výstup bude trvale v nule.

Vše co bylo napsáno výše platí pro základní hodiny 4 MHz. Vnitřní čítač plně podléhá hodinovému kmitočtu, takže v případě potřeby můžeme signál PWM v širokých mezích upravit jeho změnou, lze dosáhnout frekvence až 4 MHz.

PWMDUTY

Příkaz má dva parametry, první je pin (pro náš mikrokontrolér 08M2 vždy 2), druhý může nabývat hodnot 0 - 1023 a nastavuje střídu pulzů. V podstatě jde jen o zjednodušení příkazu PWMOUT, v němž vynecháváme zadání periody.





První program prověří činnost příkazu PWM. Budeme nechávat cyklicky motor pomalu rozbíhat, zpočátku bude jen „zpívat“ nebo se pohybovat s jemným trháním, pak se rozjede plynule.

```
1       REM řízení motoru PWM1 - PICAXE 08M2
2       start:
3         for b0=0 to 255
4         pwm 2,b0,10
5         next
6         goto start
```

Druhý program dělá prakticky totéž, ale už na pozadí a s definovanou frekvencí 4 kHz. Někdy se stejný princip používá i k řízení jasu LED.

```
1       REM řízení motoru PWM2 - PICAXE 08M2
2       start:
3         for w0=0 to 1000 step 10
4         pwmout 2,249,w0
5         pause 100
6         next
7         goto start
```

Třetí krátký program dovoluje řídit rychlost otáčení motoru připojeným trimrem. Napětí z jezdce se přečte A/D převodníkem na pinu 4 a podle výsledku uloženého do w0 se cyklicky v intervalech přibližně desetiny sekundy upravuje řízení motoru.

```
1       REM řízení motoru PWM3 - PICAXE 08M2
2       start:
3         readadc10 4,w0
4         pwmout 2,249,w0
5         pause 100
6         goto start
```

D/A převody

PICAXE 08M2 má jeden dostupný D/A převodník s pevně určeným výstupem pin 0. Tento převodník ale umí rozlišit jen 32 úrovní, takže se používá zřídka a pro méně náročné účely. Jeho výstup nejde zatížit větším proudem, musí se oddělit například pomocí OZ ve funkci sledovače napětí. Výhodou je, že po nastavení drží výstup sám a nevyžaduje další obsluhu. V případě mikrokontroléru 08M2 nesmíme zapomenout před použitím převodníku odpojit programovací kabel!

DACSETUP

Parametr tohoto příkazu má význam po jednotlivých bitech, bit 0 určuje použití země napájení jako vztažné úrovně (0) nebo externího vstupu napětí (1), bity 3 a 2 definují rozsah napětí - napájecí napětí (00), externí vstup (01) nebo interní referenční zdroj nastavený předem příkazem FVRSETUP (10) , bit 5 připojuje výstup z převodu na pin obvodu a bit 7 povoluje nebo zakazuje použití převodu. Typicky %10100000 povolí převod a jeho výstup v mezích napájecího napětí (respektive 0 až 31/32 napájecího napětí).

DACLEVEL

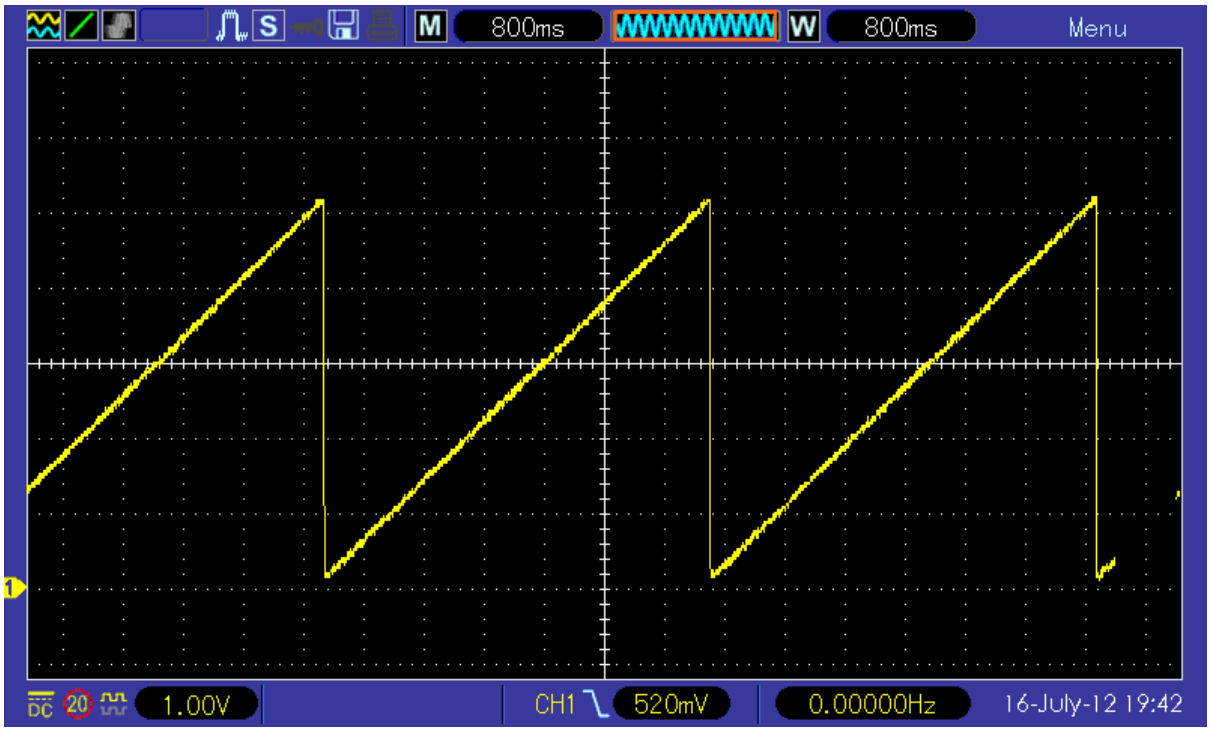
slouží k nastavení výstupní úrovně, parametr nabývá hodnot 0 až 31. Jako příklad vygenerujeme pilovité napětí, které můžeme pozorovat na osciloskopu, případně 100x prodloužíme čekání v cyklu a pak lze výstup sledovat na voltmetu.

```
1       REM Interní DAC1 - PICAXE 08M2
2       dacsetup %10100000
3       start:
4         for b0=0 to 31
5         daclevel b0
6         pause 10
7         next b0
8         goto start
```

Převodu s vyšším rozlišením a stále velmi dobrou linearitou lze dosáhnout malým fíglem, nastavíme PWM pulzy určené k ovládání výkonu motoru, řídíme jejich střídu, ale na daný výstup zapojíme RC filtr (třeba odpor 10k a za něj svitkový kondenzátor 220n), který převede střídu na napětí. V tomto případě ale nemáme možnost využít přesné vnitřní zdroje referenčního napětí, vždy pracujeme v mezích napájecího napětí. Že je převod opravdu slušně lineární se můžeme snadno přesvědčit na snímku z osciloskopu.

```
1       REM DAC2 pomocí PWM - PICAXE 08M2
2       setfreq m32                 ;hodiny 32 MHz
3         start:                    ;smycka programu
4         for w0=0 to 400           ;zmeny strídy
5           pwmout 2,100,w0         ;nastavení pulzu
```

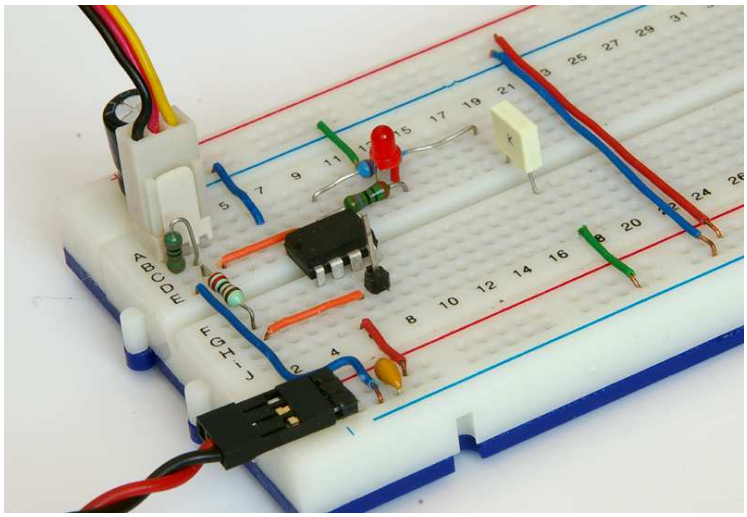
```
6       pause 100               ;zpomalení
7       next w0
8       goto start              ;zpet na zacátek
```

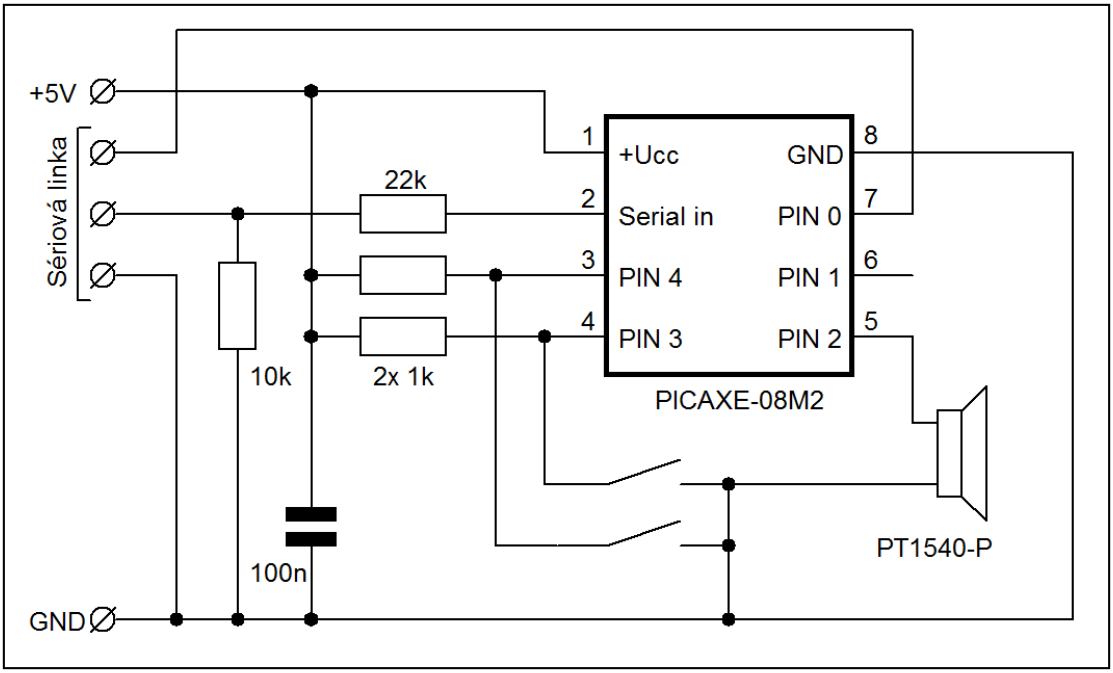


V praxi se poměrně často setkáváme s úlohami, které je obtížné řešit jedním procesorem, výhodnější je úlohu rozdělit na dílčí a nasadit dva nebo i více procesorů, které ovšem pak spolu musí nějak komunikovat. Kromě sériového přenosu dat, s nímž jsme se již zabývali, a který dovoluje přenášet přesné hodnoty v libovolném počtu, ale také vyžaduje určitou synchronizaci, je jednou z možností právě převod veličiny na napětí prvním procesorem (na pozadí) a druhý procesor si toto napětí odebere (změří) v okamžiku, kdy sám potřebuje.

Dotyková tlačítka

A/D převodníky jsou u mikrokontroléru PICAXE řady M2 využity také k podpoře ovládání programů dotykovými tlačítky, respektive jen dotykovými ploškami na plošném spoji. Detekují se oscilace na vstupu obvodu, kalibrace případně nastaví mezní hodnotu pro rozhodnutí. Nejprve si připravíme obvod podle, který bude společný pro několik dalších pokusů. Zatím neosadíme odporový trimr.





TOUCH

má dva parametry, prvním je pin, který ale musí mít současně možnost analogového vstupu, tedy s A/D převodníkem, druhý je proměnná typu byte, do níž se uloží zjištěná okamžitá hodnota úměrná přiblížení ruky.

TOUCH16

je velmi podobný, ale může mít tři parametry, první je konfigurační byte, druhý pin stejně jako v předchozím případě a třetí proměnná typu word. V konfiguračním bytu nejnižší tři bity definují dělicí poměr od 2 do 256, další dva bity určují citlivost a horní tři počet pulzů, které jsou potřeba k překlopení (32-256). Konfigurační byte není povinný, implicitní nastavení je %00001001 (256 pulzů, nízká citlivost, dělicí faktor 4)

Následující program nejjednodušším možným způsobem zkouší obsluhu dotykového tlačítka, pokud je „stisknuto“, svítí LED. Pro pokusy stačí dotýkat se konce vodiče připojeného k pinu nebo se třeba i dotknout přímo vývodu mikrokontroléru kovovým hrotem, pro lepší spolehlivost v praxi je lepší, když dotyková ploška má přibližně plochu 1 cm^2. Mez rozhodnutí (v našem případě 100) je třeba podle konkrétních podmínek upravit.

```
1       REM dotykové ovládání - PICAXE 08M2
2       start:
3         touch 4,b0
4         if b0>100 then high 1 else low 1 endif
5         goto start
```

Práce s EEPROM a RAM

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. Často se hodí, aby si mikrokontrolér do paměti poznamenával, například to, kolikrát byl program spuštěn, nebo aby si zapamatoval parametry nastavené uživatelem pro použití příště. K tomu slouží možnost ukládání dat do 256 bytů EEPROM. Na rozdíl od starších verzí mikrokontroléru je tato oblast paměti samostatná a nemusí se hlídat, abychom nepřepsali program. Budeme potřebovat nové příkazy.

WRITE

má dva parametry, první je adresa (0 - 255) místa, kam se má byte uložit, druhý tvoří data, která se mají na danou pozici uložit. Pracuje se s jednotlivým byty a pokud je potřeba uložit takto proměnnou typu word, uloží se dvěma příkazy jako odpovídající proměnné typu byte, například w0 uložíme jako b0 a b1(na různé adresy).

EEPROM

pracuje podobně, ale ukládá vícenásobná data v jednom příkazu. První parametr je adresa prvního bytu, v závorce oddělená čárkami pak následují data v libovolném počtu.

READ

je podobný příkaz, pracuje obráceně. První parametr určí adresu z níž budeme číst, druhým je proměnná typu byte, 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 zaručovaných přepisech a použití 10x denně dospějeme k deklarované (minimální) životnosti paměti za 27 roků. To je myslím přijatelný výsledek a není důvod takto paměť nevyužít. 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 ukazuje počítání, kolikrát byl program spuštěn (max. 255x). Vždy po zapnutí zvýši obsah bytu o 1 a LED blikne tolikrát, kolik odpovídá hodnotě bytu.

```
1       REM EEPROM1 pro PICAXE 08M2
2       read 0,b0               ;ctení z EEPROM
3       inc b0                  ;prictení jedné
4       write 0,b0              ;zápis do EEPROM
```

```
5       for b1=0 to b0                  ;bliknutí podle poctu
6       high 1
7       pause 600
8       low 1
9       pause 600
10      next b1
11      REM tady pokracuje výkonný program
12      end
```

Práce s RAM se nemusí omezovat jen na připravené proměnné. K dispozici je celkem 256 bytů RAM, z nichž jen 28 spodních je vyhrazeno pro základní pojmenované proměnné, ostatní jsou dostupné přes příkazy POKE a PEEK. Před využitím zbytku RAM je ale dobré seznámit se v manuálu s rozmístěním a funkcí systémových skrytých proměnných.

POKE

má dva parametry, první je adresa v RAM, druhý hodnota, která se tam má uložit. POKE 80,127 tedy uloží hodnotu 127 do registru na adrese 80.

PEEK

pracuje opačně, prvním parametrem je adresa v RAM, druhým proměnná, kam se má hodnota uložit. S registry v RAM se přímo nepracuje, hodnotu třeba pro aritmetické operace musíme vždy přenést do některé proměnné a pak případně zase zpět do registru. I tak práce s RAM výrazně rozšiřuje možnosti, když proměnné dojdou.

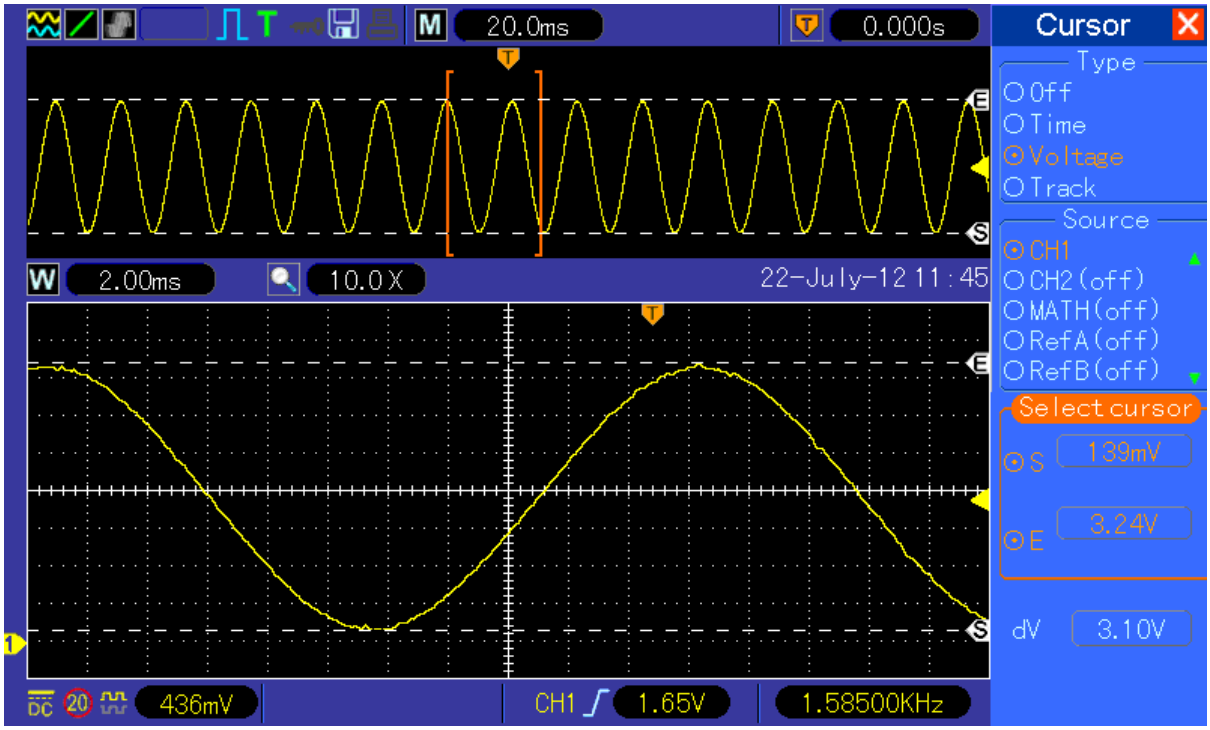
```
1       REM RAM1 pro PICAXE 08M2
2       for b1=40 to 80 poke b1,b1 next b1
3       end
```

Krátký program je nejnázornější spustit v simulaci a přepnout si okno zobrazení stavu paměti na RAM. Průběžně je vidět, jak se mění proměnná b1 a současně jak se paměť plní ukládanými hodnotami.

DDS generátor

Předchozí práci s EEPROM a D/A převodníky respektive převodem pomocí PWM můžeme použít k ukázce vytvoření DDS generátoru harmonického signálu s 60 vzorky na jednu periodu a pevným kmitočtem, který vyjde něco kolem 41 Hz. Program kontroluje obsazení bytu na adrese 60, při prvním spuštění je v tomto bytu pravděpodobně hodnota 0. Program nahraje do EEPROM tabulku vzorků, které odpovídají sinusovce a byly předem spočteny. Při dalších spuštěních už se přenos dat do EEPROM přeskakuje, na 60. bytu je uložena hodnota 255. Výsledek signálu odebíraného na kondenzátoru RC článku ukazuje snímek z osciloskopu.

```
1     REM generátor DDS - PICAXE 08M2
2     setfreq m32                          ;hodiny 32MHz
3     read 60,b0                           ;pgm EEPROM pri 1. spuštení
4     if b0 = 255 then goto start          ;(tabulka už je)
5     eeprom 00,(128,141,153,166,178,189,200,210)
6     eeprom 08,(220,228,236,242,247,251,254,255)
7     eeprom 16,(254,251,247,242,236,228,220,210)
8     eeprom 24,(200,189,178,166,153,141,128,115)
9     eeprom 32,(103,090,078,067,056,046,036,028)
10    eeprom 40,(020,014,009,005,002,001,002,005)
11    eeprom 48,(009,014,020,028,036,046,056,067)
12    eeprom 56,(078,090,103,115,255)
13    start:                               ;smycka syntézy
14      for b0=0 to 59                     ;60 vzorku na periodu
15      read b0,w1                         ;hodnota z EEPROM
16      pwmout 2,100,w1                    ;PWM prevod
17      next b0                            ;jedna perioda hotová
18      goto start
```



Udělat stejným způsobem v jazyce PICAXE výrazně vyšší kmitočet bude problém, při programování stejného typu mikrokontroléru třeba v assembleru by byly možnosti větší. Pokusíme se ale o změnu kmitočtu. Osadíme podle předchozího schématu odporový trimr a využijeme převodník A/D na pinu 4. Při inicializaci se načte napětí, to následně bude určovat zpomalení smyčky syntézy. Bohužel, pokud bychom chtěli odečítat napětí a ladit průběžně, chod už se příliš zpomalí, takto musíme vždy po nastavení frekvence přerušit napájení, aby se načetla nová hodnota. Časem vyzkoušíme lepší způsob, ale nyní se soustředíme na změny kmitočtu. Program DDS2 bude nyní vypadat takto:

```
1     REM generátor DDS2 - PICAXE 08M2
2     setfreq m32                          ;hodiny 32MHz
3     readadc 4,b1
4     read 60,b0                           ;pgm EEPROM pri 1. spuštení
5     if b0 = 255 then goto start          ;(tabulka už je)
6     eeprom 00,(128,141,153,166,178,189,200,210)
7     eeprom 08,(220,228,236,242,247,251,254,255)
8     eeprom 16,(254,251,247,242,236,228,220,210)
9     eeprom 24,(200,189,178,166,153,141,128,115)
10    eeprom 32,(103,090,078,067,056,046,036,028)
11    eeprom 40,(020,014,009,005,002,001,002,005)
12    eeprom 48,(009,014,020,028,036,046,056,067)
13    eeprom 56,(078,090,103,115,255)
14    start:                               ;smycka syntézy
15      for b0=0 to 59                     ;60 vzorku na periodu
16      pause b1                           ;úprava délky periody
17      read b0,w1                         ;hodnota z EEPROM
18      pwmout 2,100,w1                    ;PWM prevod
19      next b0                            ;jedna perioda hotová
20      goto start
```

Výstupní signál jde změnit přibližně od 41 do 0,5 Hz. Další prostředky nám poskytuje změna hodinového kmitočtu hrubě (SETFREQ) nebo velmi jemně (CALIBFREQ). Zatím jsme používali tabulku vypočítanou „ručně“ a zapsanou přímo do programu, ale pokud je signál popsatelný nějakým přijatelně složitým matematickým vztahem, lze tabulku i generovat automaticky bezprostředně před spuštěním syntézy a uchovávat v RAM, ne v EEPROM. Pak lze změnou počtu vzorků jemně nastavit frekvenci při stejné smyčce syntézy. Lze namítnout, že mikrokontrolér není dostatečně vybaven matematickými funkcemi a ani jednoduchý sinusový signál nezvládne jednoduchými prostředky spočítat. Ano, pro PICAXE 08M2 je to pravda, ale třeba procesory PICAXE řady X2 potřebné matematické vybavení mají.

Interrupt

V mnoha případech je potřeba, aby mikrokontrolér reagoval na ovládání nebo obecně na podnět ihned, ne až se příležitostně dostane k otestování očekávaného stavu na vstupech. Ostatně, pravidelné testování by hlavní program významně zdržovalo. Máme k dispozici přerušení, které se vyhodnocuje automaticky vždy mezi příkazy a u typických dlouhých příkazů (například pause) i v jejich průběhu.

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, proto budeme postupovat podrobněji a s příklady. Obsluha interruptu se chová jako standardní podprogram, když skončí, program se vrátí za bod, z něho udělal

odskok.

```
Setint %00000000,%00000001
```

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“.

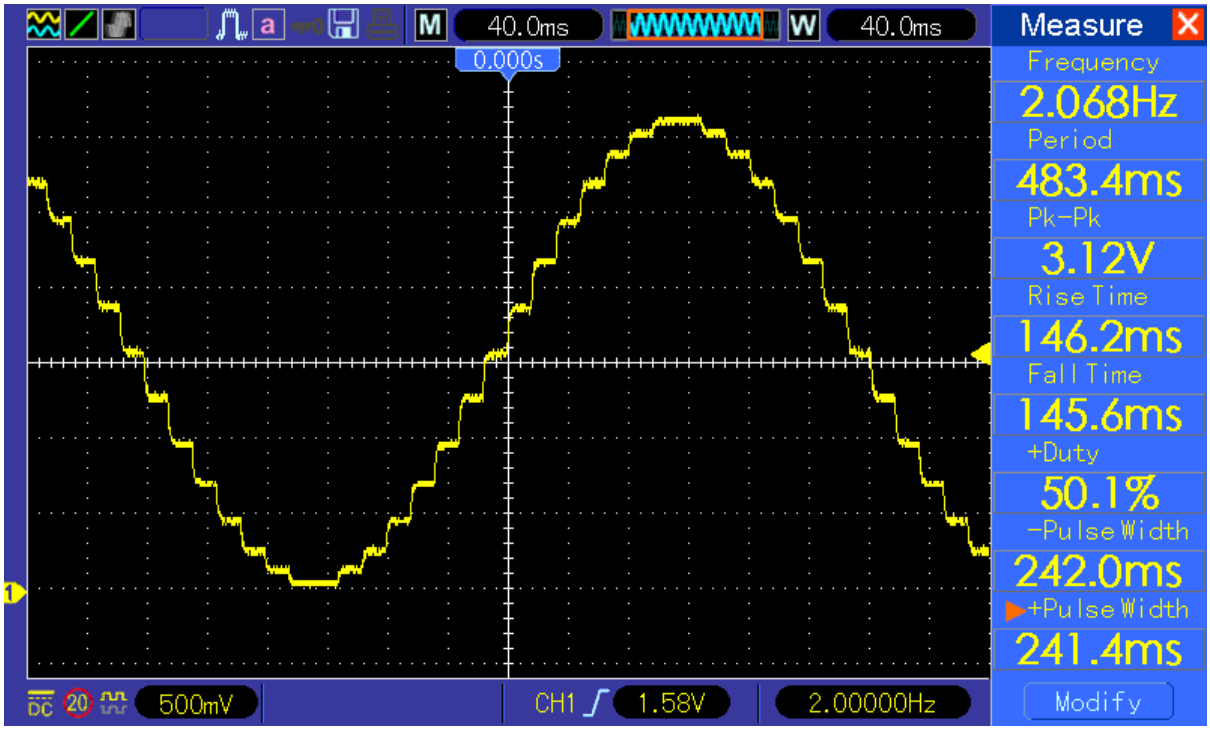
Interrupt je podprogram jako každý jiný, musí být tedy ukončen návratem (RETURN) a na jeho návěští se lze programově odkázat podle potřeby (GOSUB, GOTO). Je tu ale jedna zásadní odlišnost, na rozdíl od podprogramu, který voláme vždy ze zadaného místa a tedy víme (respektive máme vědět), v jakém stavu 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. Používáme-li v obsluze interruptu nějakou proměnnou k zápisu, měla by být použita jen v něm. Po skončení výkonu interruptu se program vrátí na místo, kde byl přerušen, a pokračuje v činnosti.

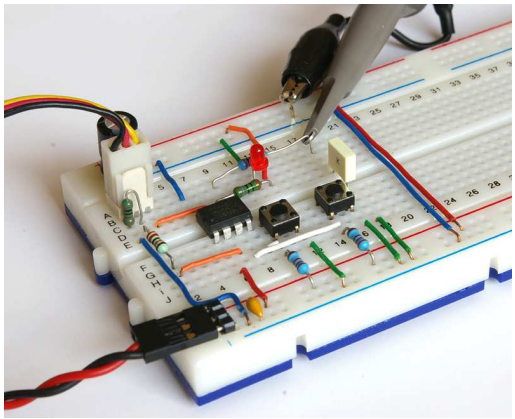
Nastavení příkazem Setint platí pro jedno volání interruptu, má-li být používán opakovaně, musíme jej vždy znovu „nahodit“ opětovným použitím SETINT, třeba přímo na závěr obslužného podprogramu. Lepší je používat SETINT na jiném místě programu a pokud možno se vyhnout možnosti, že výkon podprogramu interrupt bude přerušen jeho dalším voláním.

Využijeme dosavadní zapojení, odstraníme odporový trimr a mezi piny 3 a 4 a zem připojíme tlačítka, ke kladné větvi napájení je spojíme rezistory 1k. Základ programu DDS generátoru zůstane stejný, jeho tabulku zkrátíme na poloviční počet vzorků vynecháním každého druhého a do inicializace doplníme aktivaci přerušení. Vlastní přerušení navázané na pin 3 pak bude měnit frekvenci jemně nahoru nebo dolů podle toho, jestli je nebo není stisknuto tlačítko na pinu 4. Lze samozřejmě naprogramovat jedno tlačítko na zvyšování a druhé na snižování frekvence, ale toto je pro začátek jednodušší. Při frekvencích pod 10 Hz už by bylo potřeba zvětšit hodnotu kondenzátoru, na němž snímáme napětí, začíná se výrazně projevovat „schodovitost“ signálu.

```
1     REM generátor DDS3 - PICAXE 08M2
2     setfreq m32                        ;hodiny 32MHz
3     setint %00000000,%00001000         ;prerušení pin3 v 0
4     read 30,b0                         ;pgm EEPROM pri 1. spuštení
5     if b0 = 255 then goto start        ;(tabulka už je)
6     eeprom 00,(128,153,178,200,220,236,247,254)
7     eeprom 08,(254,247,236,220,200,178,153,128)
```

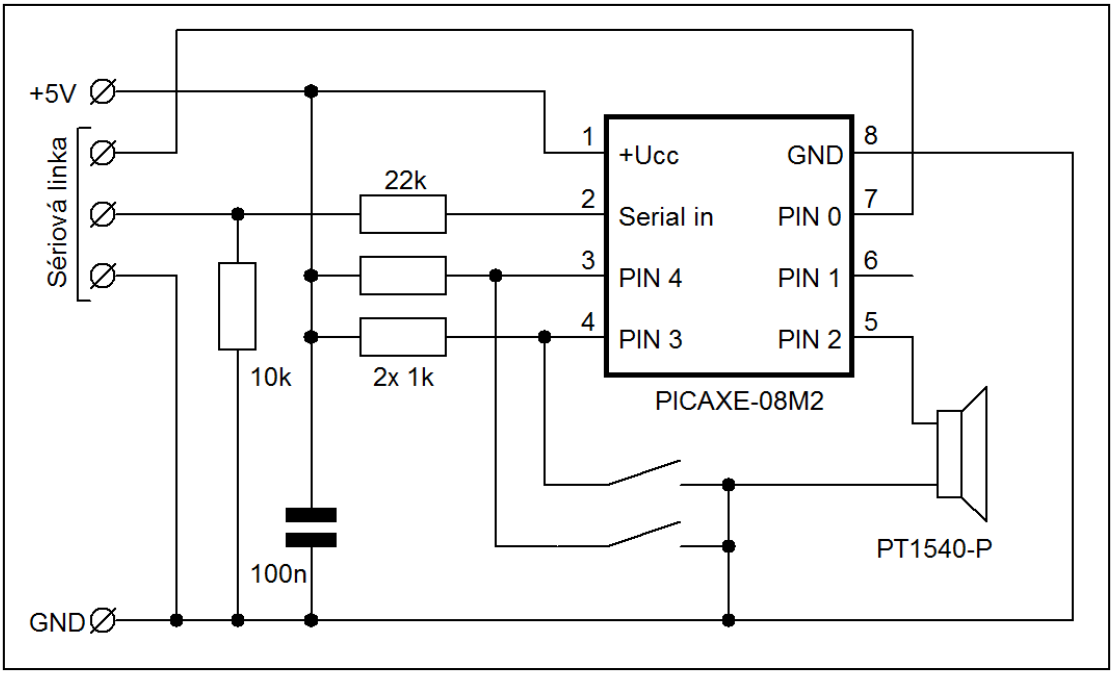
```
8       eeprom 16,(103,078,056,036,020,009,002,002)
9       eeprom 24,(009,020,036,056,078,103,255)
10      start:                                          ;smycka syntézy
11        for b0=0 to 29                                ;60 vzorku na periodu
12          pause b1                                    ;úprava délky periody
13          read b0,w1                                  ;hodnota z EEPROM
14          pwmout 2,100,w1                             ;PWM prevod
15          next b0                                     ;jedna perioda hotová
16        goto start                                    ;skok na zacátek syntézy
17      interrupt:                                      ;obsluha prerušení
18        if pin4=1 then
19        inc b1 else dec b1 endif                      ;zvýšení/snížení
20        b1=b1 max 254                                 ;ošetrení maxima
21        b1=b1 min 1                                   ;ošetrení minima
22        high 1 pause 100 low 1 pause 100              ;bliknutí LED
23        setint %00000000,%00001000                    ;nastavit INT
24        return                                        ;návrat z interruptu
```

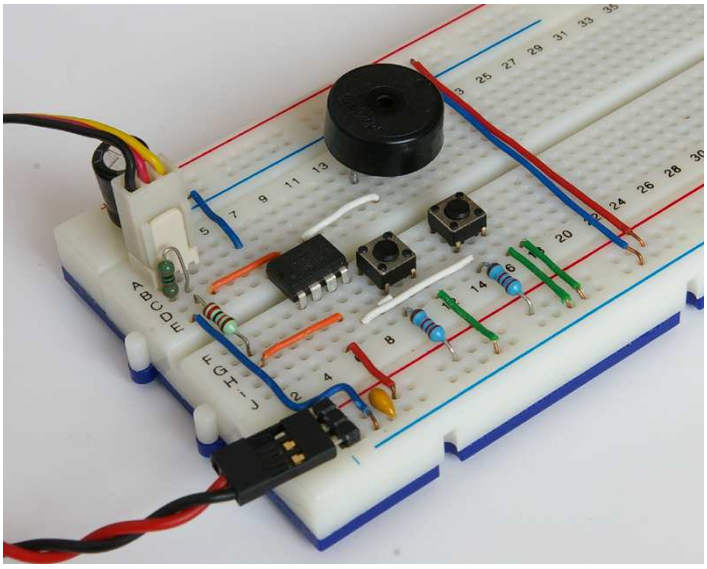




Zvuk

Už jsme pracovali s pulzy a zvuk není nic jiného než opakované pulzy, takže vygenerovat nějaké to pípnutí by nebyl problém ani s příkazy, které si již ukázali. Procesory PICAXE ale jsou poměrně dobře vybaveny na pohodlnější a mnohem komfortnější práci se zvukem. Nejprve si připravíme zapojení podle následujícího schématu. Přímo k vývodu mikrokontroléru místo LED můžeme připojit piezoměnič PT1540-P, to je nejjednodušší způsob vytvoření zvukového výstupu. Lze použít i malý reproduktor 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. Nebudeme se zabývat zvukovým výstupem analogového signálu (z převodníku D/A), ale jen jednoduchým pravoúhlým signálem.





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. (*27) Tento příkaz se používá, je-li třeba upozornit obsluhu nebo jako součást zvukových efektů, není vhodný pro reprodukci hudby. Po dobu generování zvuku se mikrokontrolér nemůže věnovat ničemu jinému. Následující příklad po stisku tlačítka na pinu 4 trojitě pípne, při stisku tlačítka na pinu 3 zazní delší jeden hlubší tón.

```
1       REM Zvuk1 - PICAXE 08M2
2       start:
3         if pin4 = 0 then
4           sound 2,(110,5,0,5,110,5,0,5,110,5)
5           pause 500
6           endif
7         if pin3 = 0 then
8           sound 2,(2,100)
9           pause 500
10          endif
11        goto start
```

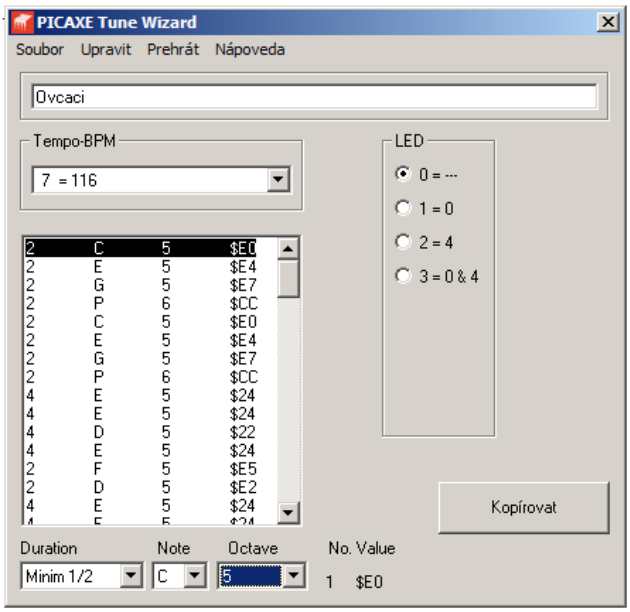
TUNE

slouží ke generování „hudby“, na mikrokontroléru 08M2 je výstup pevně směrován na pin 2, na jiných typech mikrokontrolérů PICAXE lze výstup určit. První parametr udává chování LED v jistých typických konstrukcích používaných pro výuku a podporovaných výrobcem, 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 respektive automatické vygenerování zdrojového textu slouží samostatný program Tune Wizard, který je součástí prostředí, v kterém pracujeme. (*28) Zkušební program po spuštění stále dokola hraje kousek písničky Ovčáci čtveráci.

```
1       REM Zvuk2 - PICAXE 08M2
2       start:
3         tune 0, 7,($E0,$E4,$E7,$CC,$E0,$E4,$E7)
4         tune 0, 7,($CC,$24,$24,$22,$24,$E5,$E2,$24)
5         tune 0, 7,($24,$22,$24,$E5,$E2,$E4,$E2,$E0)
6         pause 3000
7         goto start
```

Zmíněný program TuneWizard 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 hodně zdlouhavá a pracná. Pokud by bylo třeba použít v programu nějakou melodii, nemusíme ji nutně psát, lze si najít a vybrat z knihovny nejméně z tisíce hotových skladeb. Z našeho pohledu asi nepůjde o to generovat skutečné

melodie, jako spíš napsat jednoduché krátké upozorňovací zvuky a znělky, což jde docela dobře.



PLAY

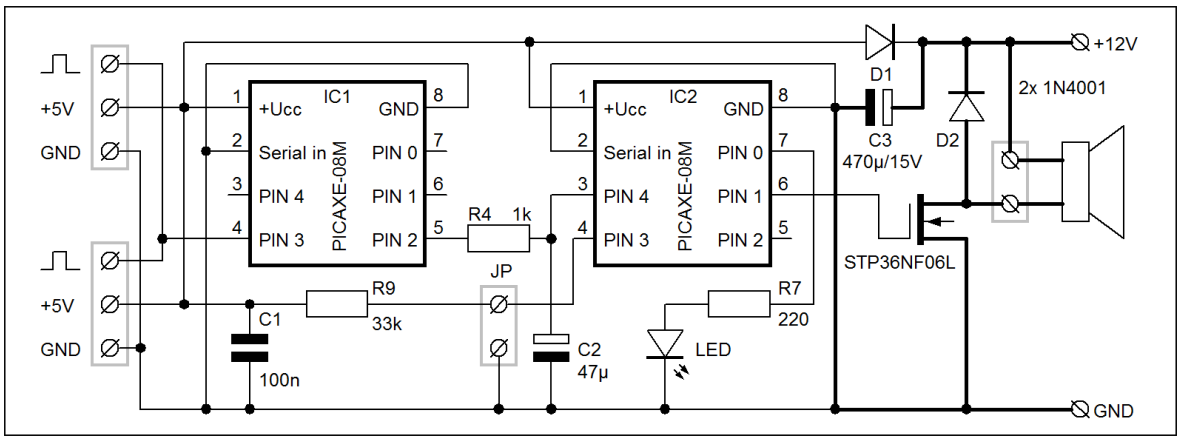
Mikrokontrolér PICAXE 08M2 má přímo v sobě čtyři připravené melodie, jež lze volat zjednodušeným zápisem připomínajícím příkaz Tune. Prvním parametrem je číslo melodie (0 - 3), druhý určuje režim ostatních výstupu LED v přípravcích (pro nás hodnota 0). (*17) Praktická použitelnost kromě předvedení je asi nulová, ale za vyzkoušení to stojí. Následují program stále dokola přehrává všechny čtyři melodie. Generování zvuku je jednou ze situací, kdy program po delší dobu prakticky nemůže reagovat třeba na stisk tlačítka a v praxi se velmi hodí využití interruptu, který jsme si už ukázali.

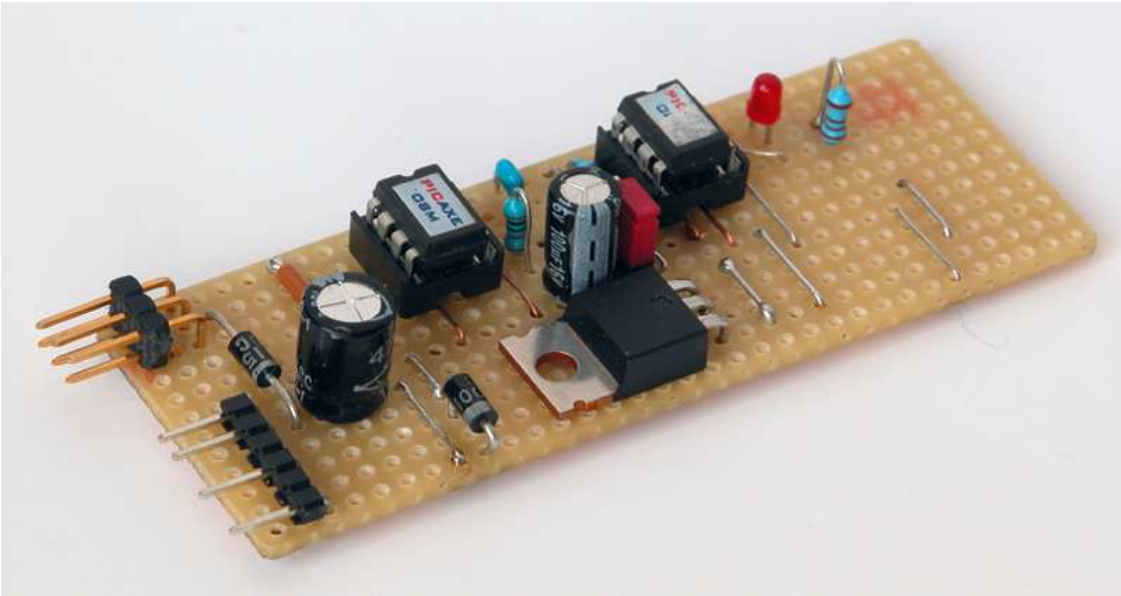
```
1       REM Zvuk3 - PICAXE 08M2
2       start:
3         play 0,0 pause 4000
4         play 1,0 pause 4000
5         play 2,0 pause 4000
6         play 3,0 pause 4000
7         goto start
```

Než ukončíme téma zvuku, rád bych uvedl jeden praktický příklad, který ukazuje možnosti PICAXE, dokonce staršího a méně výkonného typu 08M. Alan Bond z Velké Británie vytvořil zhruba před čtyřmi roky zvukový modul pro lodní modely, který imituje pomaluběžný dieselový velkoobjemový motor, benzínový motor nebo parní stroj. Propojkou jde nastavit počet válců 1 - 6, i ten je ve výsledku rozpoznatelný. Mikrokontrolér napodobí start, nezatížený chod ve volnoběhu, změny otáček a po jisté době nečinnosti kašlavé zhasnutí motoru. Zapojení je na schématu a celé zařízení s konstrukčním popisem u nás vyšlo v časopise RC revue 5/2010. Z našeho hlediska je zajímavé použití dvou mikrokontrolérů. Levý z nich je velmi málo vytížený, program v něm čte pulzy od RC přijímače, které procházejí zvukovým modulem do regulátoru pohonného elektromotoru, a podle pulzů

generuje napětí na svém pinu 2. Ve střední poloze, kdy pohon stojí, a přicházejí pulzy se šířkou kolem 1,5 ms, je generováno téměř nulové napětí, při přidání plynu vpřed nebo vzad se pulzy prodlužují (max. 2,0 ms) nebo zkracují (min. 1,0 ms) a napětí v obou případech roste až k napájecímu. Druhý procesor generuje zvuk, který se mění podle „otáček“ motoru, využívá se tónů smísených se šumy.

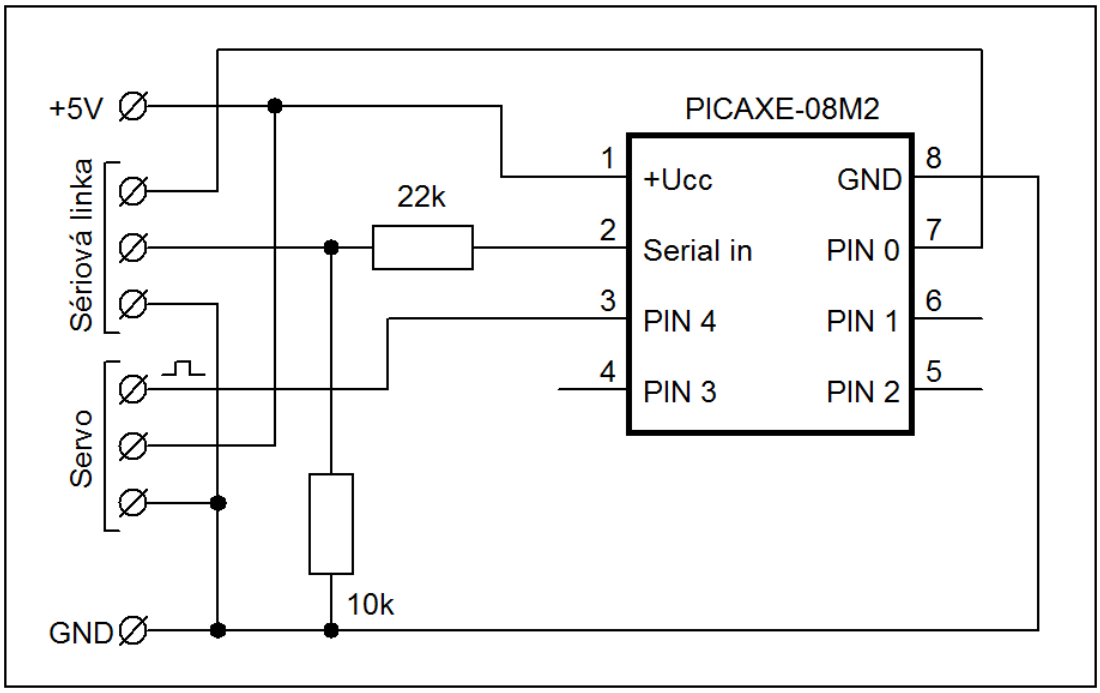
Rozdělení úloh je nutné kvůli tomu, že procesor generující zvuk už nezvládá současně obsluhovat vstupní pulzy, na výsledku by se to projevilo značně rušivými nepravidelnostmi. Napětí, které se zde používá k přenosu informace o otáčkách, i přes omezené rozlišení a přesnost převodu s rezervou vyhovuje, dokonce kondenzátor C2 kromě toho, že integruje výstupní pulzy PWM z prvního procesoru, simuluje i setrvačnost chodu reálného motoru. V balíčku programů k tomuto článku na internetových stránkách www.aradio.cz je uveden plný program do obou procesorů i ukázky zvukového výstupu pro všechny tři druhy motorů. Tomu, kdo to s PICAXE myslí jen trochu vážně, doporučuji si programy projít jako ukázku opravdu dobře odvedené práce a určitě si také poslechnout nahraný výsledný zvuk. K nějakému „škruhlání a pípání“, jaké od mikrokontrolérů obvykle při syntéze zvuku slýcháme, to má docela daleko. Náklady na stavbu tohoto zvukového modulu tvoří asi 1/10 až 1/20 toho, co stojí tovární výrobek s podobnou kvalitou výsledku.





Ovládání pohybu

Protože jsou mikrokontroléry PICAXE původně určeny pro výuku robotiky a amatérská robotika velmi často využívá modelářské díly, jsou pro jednoduché použití připraveny povely k ovládání modelářských serv. Základní informace o způsobu jejich obsluhy mikrokontrolérem byly opakovaně i na stránkách ARadia uvedeny, takže jen heslovitě: napájení je stejnosměrným napětím typicky 4,8 - 6 V, na signálovém vstupu se očekávají kladné pulzy se šířkou 1,0 až 2,0 ms (střed 1,5 ms) opakované typicky po 20 ms, tomu odpovídá na páce serva výchylka asi v rozmezí +/-60 stupňů od střední polohy. Ke zkouškám je lepší používat levné analogové servo než nějaké superpřesné digitální, signál z mikrokontroléru projevuje v některých případech drobné nestability, které pásmo necitlivosti levných serv schová, zatímco na velmi přesných servech můžeme pozorovat neustálé chvění. Na zkušební desce připravíme zapojení podle dalšího schématu.



Malé „spotřební“ servo asi za 100 Kč vyhoví na pokusy lépe než 30x dražší přesné rychlé digitální servo se střídavým motorem.

SERVO

je speciální příkaz pro řízení serv (*25). Podobně jako Pulsout prvním parametrem se udává pin, druhým poloha serva respektive délka impulzu (neutrál odpovídá při základním kmitočtu hodin číslu 150). Zásadní rozdíl je v tom, že příkaz Servo využívá vnitřního časovače mikrokontroléru a automaticky opakuje zadaný pulz s periodou 20 ms až do další změny nastavení, mikrokontrolér přitom nezávisle vykonává program dál. Potřebujeme-li ukončit ovládání serva, pošleme na příslušný výstup úroveň L (např. low 4). S procesorem 08M2 příkaz pracuje korektně jen při hodinové frekvenci 4 nebo 16 MHz.

Vyzkoušíme si program, který bude s odstupem dvou sekund přejíždět z jedné krajní polohy do druhé. Skutečný rozsah pohybu serva je vždy větší, ale ten už bychom museli individuálně zkoušet a nastavit. Pokud změníme hodiny ze 4MHz na 16 MHz, na rozsahu pohybu serva se nic nezmění, ale čtyřikrát se zkrátí doba čekání, příkaz Pause je na hodinách závislý.

```
1       REM Servo1 (prejezdy) - PICAXE 08M2
2       setfreq m4                      ;hodiny 4 MHz
3       start:
4         servo 4,100                   ;jedna krajní poloha
5         pause 2000                    ;cekání 2s
6         servo 4,200                   ;druhá krajní poloha
7         pause 2000                    ;cekání 2s
8         goto start
```

Program upravíme tak, aby servo přejíždělo mezi krajními polohami pomalu přibližně během 10 sekund. Všimneme si, že řízení serva ve 100 krocích na rozsah pohybu není zrovna přesné (kvalitní serva jsou schopna rozlišit zhruba 2000 poloh) a i jednoduché servo sebou škube, viditelně krokuje.

```
1       REM Servo2 (zpomaleni) - PICAXE 08M2
2       start:
3         for b0=100 to 200
4         servo 4,b0
5         pause 100
6         next b0
7         for b0=200 to 100 step -1
8         servo 4,b0
9         pause 100
10        next b0
11        goto start
```

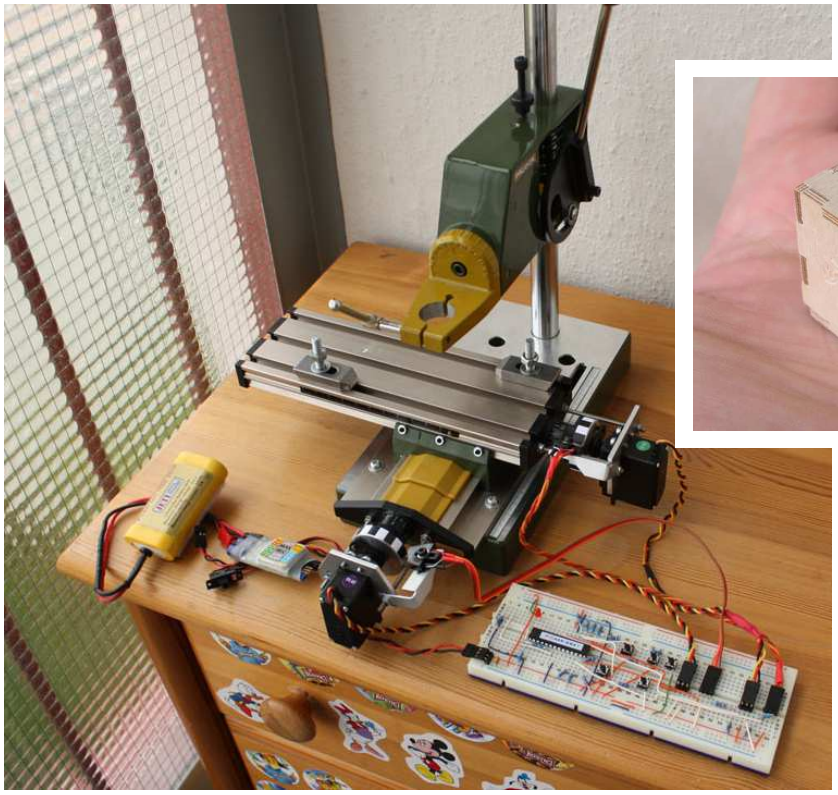
Výhoda příkazu Servo je chodu na pozadí bez obsluhy, jinak ale bývá praktičtější použít starý známý příkaz Pulsout a vyšším kmitočtem hodin jeho práci zpřesnit. Kromě toho můžeme pak obsloužit jedním procesorem více serv, ovšem za tu cenu, že procesor už může dělat něco jiného jen v mezidobí mezi pulzy a musíme čas, který je k dispozici, sledovat.

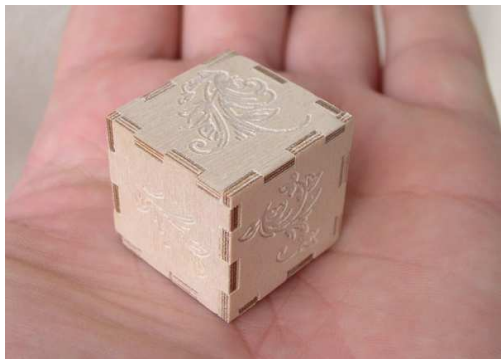
Zkusíme předchozí program přepsat tak, aby pohyb byl plynulý:

```
1       REM Servo3 (zpomaleni) - PICAXE 08M2
2       setfreq m32                         ;hodiny 32 MHz
3       start:
4         for w0=800 to 1600                ;800 kroku
5           pulsout 4,w0                    ;výstup 1 pulzu
6           pause 160                       ;cekání cca 20ms
7           next w0
8         for w0=1600 to 800 step -1
9           pulsout 4,w0
10          pause 160
11          next w0
12        goto start
```

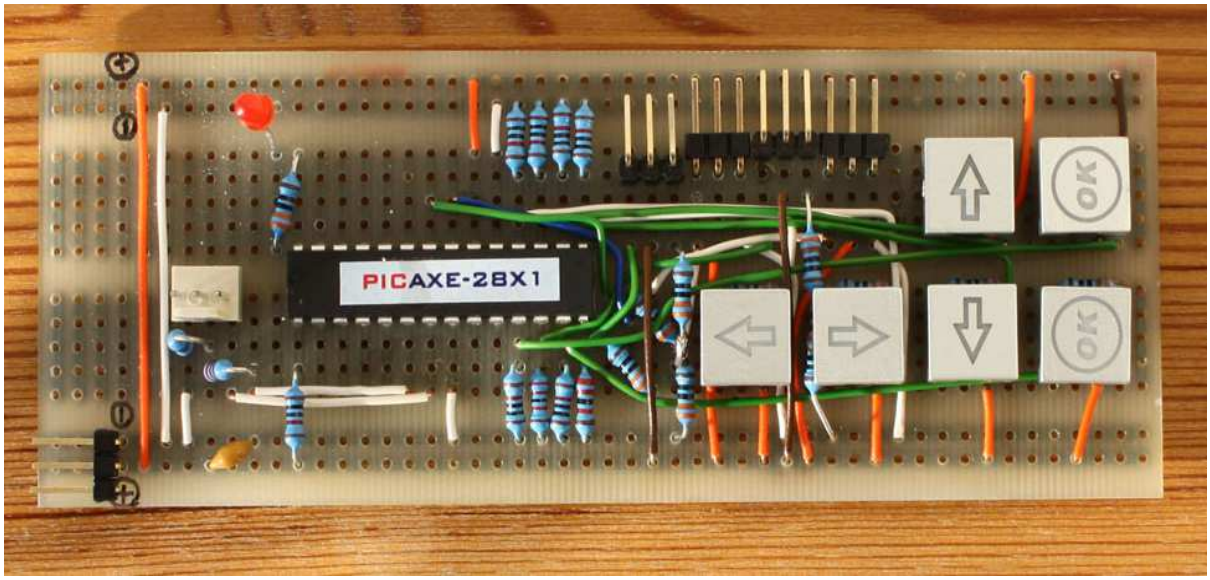
Pokud ještě zjistíme nějaké krokování, půjde pravděpodobně o důsledek širšího pásma necitlivosti serva, 800 kroků na rozsah by už nemělo být jinak znát. Zbývá se zamyslet na otázkou, kolik serv takto můžeme přímo vývody mikrokontroléru obsloužit. Budeme-li počítat s nejdelšími možnými pulzy 2,0 ms, pak při zachování periody opakování 20 ms se dá ovládat 10 serv, máme-li dostatek výstupů. Reálně ale potřebujeme ještě nějaký čas mezi pulzy a na přípravu, takže 8 serv se považuje za praktické maximum. Na druhou stranu není nezbytné dodržet přesně odstup 20 ms, při mírně kratším se dokonce vlastnosti serva trochu zlepší, při mírně delším klesá postupně moment a rychlost pohybu serva.

Serva se nemusí použít jen tak jak jsou, ale dají se i jednoduše upravit, aby se mohl jejich výstup otáčet dokola a šířce ovládacího pulzu neodpovídala výsledná poloha, ale směr a rychlost otáčení. Co se s takto upravenými servy, která stále z hlediska mikrokontroléru mají stejné ovládání, dá dělat, může ukázat příklad přestavby malé stolní frézky Proxxon na CNC ovládání, které po krátkém seznámení s mikrokontroléry PICAXE udělal před několika léty Luboš Hort z Prahy.





K pohonu posuvů stolku byla použita dvě upravená serva Hitec HS-625MG, která sloužila v modelu letadla přes dva roky a měla už značné vůle, při snímání výstupní polohy optickými kotouči a infrasenzory tyto vůle nevadí. Držáky serv z duralového plechu jsou připevněné ke stolku pomocí kousků závitové tyče, kousky tyče sloužily i k přenosu otáčení na původně ručně ovládané kličky. K řízení byl použit mikrokontrolér PICAXE-28X1 a dva infrasenzory QRB1114. Procesor má dostatečnou kapacitu na jednoduché výrobky, pojme přibližně 1000 řádků programu a dává k dispozici 0 až 12 vstupů a 8 až 16 výstupů, díky tomu lze snadno zapojit i ovládání třetí osy např. u Proxxonu MF-70. Čtyři tlačítka slouží pro ruční ovládání posuvu a páté pro spouštění naprogramovaného frézování. Úprava frézky včetně naprogramování zabrala týden a dva víkendy práce po večerech, celkové náklady činily něco přes 800 Kč bez serv, ta byla z modelářského použití víceméně už vyřazena.



I když v současné době už frézka vypadá jinak a dostala lepší krokové motory, řízení pomocí mikrokontroléru PICAXE zůstalo. Původní verze se servy byla velmi rychle hotová a fungovala, jak lze posoudit i z přiložených fotografií. Podrobné informace o tomto projektu a postupných úpravách včetně schémat najdete na internetových stránkách http://luboshort.cz/ v sekci ostatní.

Spící mikrokontrolér

Už jsme se setkali s příkazy pro ukončení programu END a STOP, kromě ukončení je ale také možné program na omezenou dobu uspat. K čemu je to dobré? Mikrokontrolér odebírá v základním zapojení za chodu proud řekněme 1 mA, k tomu se přidává proud do zátěže (LED, piezoměnič, …). Někdy nám stačí, aby mikrokontrolér 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 mikrokontrolér plnohodnotnou prací a jeho spotřeba se nemění, i když z našeho pohledu nedělá nic. Pokud na dobu, kdy není potřeba činnost, souvisle mikrokontrolér uspíme, klesne jeho spotřeba třeba na 0,1 mA. Rozdíl se může zdát malý, nicméně jde-li o činnost dlouhodobou a napájení z baterií, znamená to, že baterie vydrží téměř 10x déle, a to už zanedbatelné v žádném případě není.

NAP

má jeden parametr v rozsahu 0 - 14 (pro některé starší procesory 0 - 7), jímž se určuje doba přechodu do režimu s nízkou spotřebou. Jednotkou (NAP 0) je 18 ms, výsledná doba se přibližně spočte jako 2^hodnota parametru * 18 ms. „NAP 1“ uspí mikrokontrolér na 32 ms, „NAP 2“ na 72 ms atd. Nejdelší spánek, který lze takto vyvolat, trvá přes 4 minuty. V průběhu uspání se interrupt nevyhodnocuje a doba není závislá na nastaveném hodinovém kmitočtu. Doba uspání není přesná, takže třeba používat NAP místo PAUSE k časování programu zpravidla nepůjde.

SLEEP

slouží stejným způsobem k uspání, ovšem obvykle na delší dobu. Má jeden parametr v rozsahu 0 až 65535, jednotkou doby je 2,3 s (2,1 s u procesorů řady X1 a X2). „Sleep 2“ uspí mikrokontrolér na 4,6 s, maximální hodnota parametru na téměř 42 hodin.

Je samozřejmé, že když se mikrokontrolér 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 mikrokontrolér 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 mikrokontrolér uspat, spotřeba se tím velmi významně sníží.

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

Program (Spanek) slouží k jednoduchému vyzkoušení uspání mikrokontroléru. Stiskneme-li za běhu krátce tlačítko zapojené mezi pin 3 a zem (s pull up rezistorem), v naprosté většině případů se nestane nic, protože obvod spí. Pokud tlačítko podržíme, blikne LED připojená na pin 0 (k zemi přes rezistor) jednou za 4,6 s, když se po cyklickém probuzení mikrokontroléru dostane ke slovu přerušení. Při pokusu o nové přenesení programu bude PC většinou viditelně čekat na probuzení mikrokontroléru. Schéma přípravku tentokrát už neuvádíme, v této fázi by si jej každý již měl zvládnout zapojit sám.

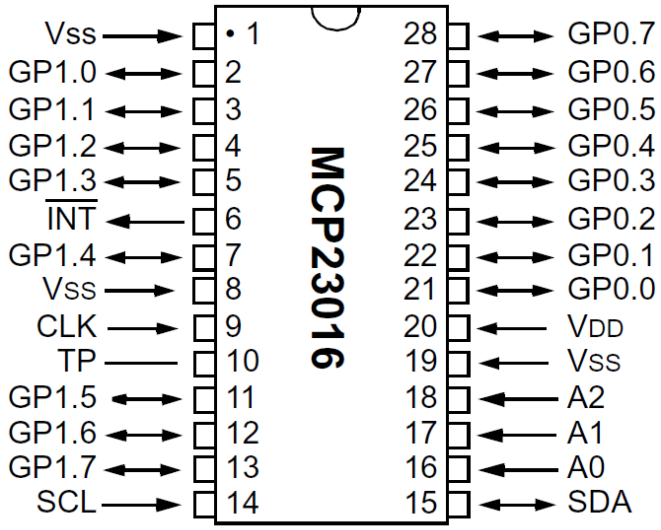
```
1       REM SPANEK pro PICAXE 08M2
2       start:
3         setint 0,8                    ;interrupt na tlačítko
4         sleep 2                       ;uspání na 4,6 s
5         goto start                    ;skok na začátek
6       interrupt:                      ;výkon přerušení
7         high 0                        ;bliknout LED
8         pause 100
9         low 0
10        return                        ;návrat
```

Sběrnice I²C

Velmi silným prostředkem pro obousměrné spojení jak více mikrokontrolérů PICAXE mezi sebou tak s jinými mikrokontroléry, mikroprocesorovými systémy i s čidly (teploměry, akcelerometry, čidly tlaku, A/D a D/a převodníky …) a dalšími obvody je obsluha standardní sběrnice I²C (Inter-Integrated Circuit). Tato sběrnice využívá dva vodiče označované SCL (hodiny) a SDA (data) a k tomu samozřejmě vztažnou zem jako třetí vodič. Všechna zařízení jsou na sběrnici připojena paralelně, jejich výstupy mají otevřený kolektor a na jednom místě jsou vodiče spojeny s kladným napájením přes rezistory (zpravidla 4k7). Délka sběrnice je omezena tím, že její kapacita nesmí přesáhnout 400 pF, což pro běžné vzdálenosti desítek cm až jednotek metrů zpravidla stačí.

Komunikace je simplexní (v jednom okamžiku může probíhat jen jedním směrem) s detekci kolize (začít komunikovat může každé zařízení, pokud je sběrnice volná). Zařízení rozdělujeme na „master“, ta generují i signál hodin a vyzývají ostatní, a „slave“, ta zjednodušeně řečeno „poslouchají“ případně „odpovídají na vyzvání“. Jednotlivá zařízení mají na sběrnici svou (zpravidla jednobytovou) adresu, jejíž jedna část bývá pevně daná typem zařízení, část je volitelná a nastavitelná adresovými vývody. Jeden (nejnižší) bit funguje jako rozlišení čtení nebo zápisu. Ve výsledku může na jedné sběrnici teoreticky celkem pracovat maximálně 127 zařízení, ale máme-li k dispozici (obvykle) tři adresové vývody, pak těchto stejných obvodů může být nejvýše 8. V praxi se ale setkáváme s mnohem menším počtem obvodů, spíše narazíme na maximální kapacitu sběrnice. Z našeho hlediska bude stačit se zabývat jen sestavou s jedním masterem (PICAXE) a jedním nebo několika málo obvody slave.

Jako příklad použijeme opravdu hodně užitečný obvod MCP23016, což je expandér umožňující rozšířit možnosti našeho mikrokontroléru 08M2 celkem o 16 jednotlivě konfigurovatelných vstupů/výstupů. Obvod může pracovat i při sníženém napětí (2,0 - 5,5 V) a jeho výstupy lze zatížit proudem až +/- 25 mA, takže může třeba přes rezistory budit LED.



Expandér má dva osmibitové porty (GP0 a GP1) s jednotlivě nastavitelnými bity. Potřebuje a generuje si vlastní hodinový kmitočet 1 MHz, k tomu slouží RC článek na vývodu CLK. Hodinové pulzy můžeme sledovat na výstupu TP, jinak se tento vývod nevyužívá. Nejprve je nutné nastavit adresu obvodu pomocí tří vývodů A0 až A2, je-li obvod na sběrnici jeden,

volíme zpravidla kombinaci 000. Celá adresa bude sestavena z horní pevné části (0100) dané dohodami respektive výrobcem, tuto část najdeme v katalogovém listu, naší volby adresy (000) a nejméně významný bit R/W pro zápis dat bude mít hodnotu 0: výsledek je binárně 01000000.

Před použitím je nutné obvod inicializovat, v první řadě určit, které vývody budou vstupní a které výstupní pro oba porty. Řídící slova IODIR0 = 06H a IODIR1 = 07H následují osmibitové masky, v níž je pro každý vstupní bit 0 a pro každý výstupní 1. Řídící slova GP0 = 00H a GP1 = 01H pak umožní zápis/čtení jednotlivých portů. Obvod toho umí podstatně více včetně generování přerušení, nicméně my si vystačíme s těmito základními funkcemi.

Ještě jednu věc je vhodné mít vždy na paměti. Řekněme, že chceme zapsat hodnoty do portu GP0 a pak GP1. Logika říká, že pošleme řídící kód pro GP0, hodnotu na GP0, pak řídící kód pro GP1 a hodnotu pro GP1. Takto to také funguje, ale jen pokud jde o dva samostatné příkazy v Basicu PICAXE, mezi nimiž necháme komunikaci „spadnout“ a druhým příkazem ji opět navážeme. Pokud použijeme jeden jediný příkaz s více parametry, první byte bude brán jako řídící kód pro GP0 (dle předpokladu), druhý jako data pro GP0, ale třetí bude automaticky považován za data pro GP1 (mělo to být řídící slovo pro GP1) a čtvrté opět automaticky za data pro GP0 (měla to být data pro GP1). Vznikne zmatek. V tomto případě řídícím slovem určujeme jen „odkud se začne“ v rámci jednoho příkazu, další jsou už byty střídavě pro jednu a druhou polovinu dvoubytového registru. Obdobně to platí nejen pro nastavení výstupů, ale pro všechny registry. V podstatě to dost urychluje a zjednodušuje práci, tedy pokud na to nezapomeneme. Z hlediska programování PICAXE nám stačí tři příkazy:

HI2CSETUP

má obecně dva nebo čtyři parametry oddělené čárkami. První určuje, jestli je mikrokontrolér brán jako master nebo slave (I2CMASTER / I2CSLAVE). Je-li master, pak následuje adresa slave zařízení, s nímž se bude nyní pracovat, potom rychlost komunikace (I2CFAST / I2CSLOW) a nakonec délka adresy (I2CBYTE / I2CWORD). Je-li mikrokontrolér na pozici slave, což ale řada M2 jednoduše neumožňuje, na to musíme mít mikrokontrolér z řady X2, pak následuje jen adresa zařízení. Příklad: HI2CSETUP I2CMASTER, %01000000, I2CFAST, I2CBYTE . Toto znamená, že mikrokontrolér je master, bude pracovat se zařízením %01000000 (pokud chceme zařízení střídat, musíme vždy před změnou použít znovu příkaz HI2CSETUP), komunikace poběží nejvýš na 400 kHz (snížená rychlost je 100 kHz) a adresa má jeden byte. Je-li třeba, přiřazení lze ukončit příkazem HI2CSETUP OFF.

Uvedené parametry rychlosti 400/100 kHz platí po základní kmitočet procesoru, pro zvýšený (např. 8 MHz) je třeba modifikace jako je I2CFAST_8 a podobně. Je to obdoba příkazů nastavení sériového přenosu.

HI2COUT

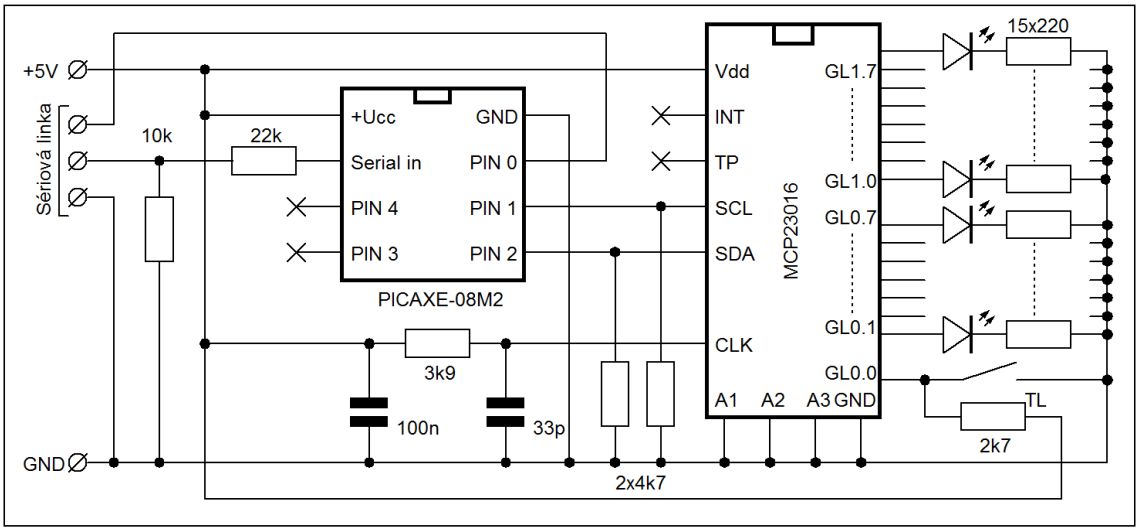
zapisuje hodnoty respektive hodnoty proměnných uvedených v závorce, ostatní parametry jsou určeny předchozím HI2CSETUP.

HI2CIN

načte hodnoty do proměnných uvedených v závorce, ostatní parametry jsou určeny předchozím HI2CSETUP.

Jako ukázku si uděláme efekt běžícího světla, ale „světelný had“ bude pořádně dlouhý, využijeme 15 výstupů na LED. Jeden z vývodů nasměrujeme jako vstup a zapojíme na něj tlačítko.

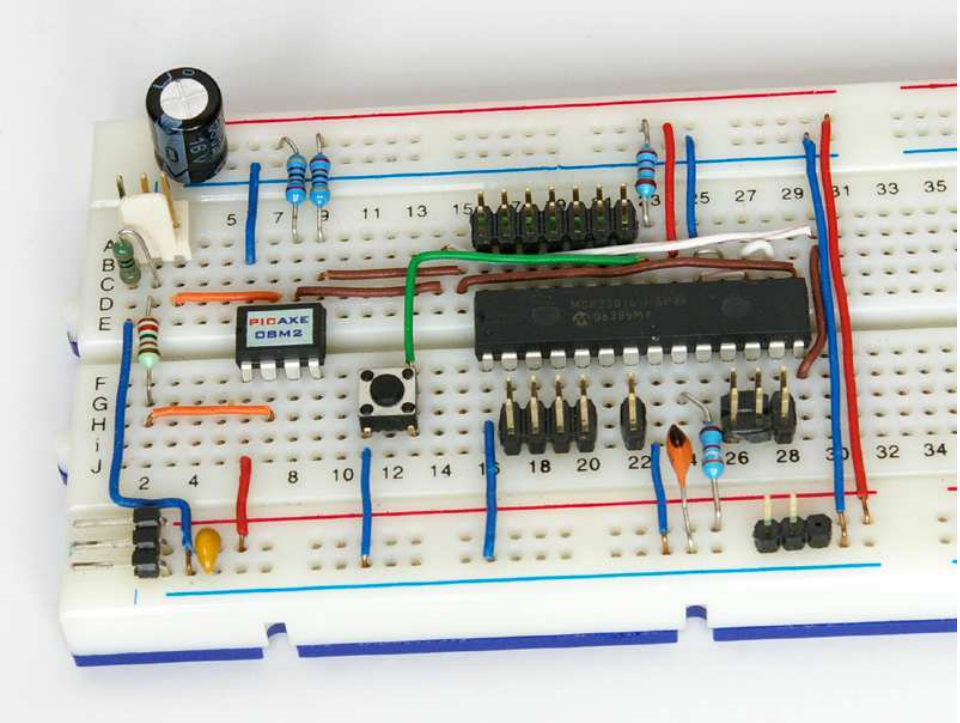
Tlačítko budeme procesorem testovat a pokud bude stisknuto, světlo přeběhne. Stejného cíle by bylo možné dosáhnout pomocí univerzálních obvodů podstatně jednodušeji a levněji, ale berme to jako příklad, na němž se komunikace po I2C sběrnici snadno vyzkouší.




```
1       REM MAXIHAD pro PICAXE 08M2
2       REM s expanderem MCP23016
3       pause 100
4       ;prodleva kvuli resetu MCP23016 (min 70 ms)
5       hi2csetup i2cmaster,%01000000,i2cfast,i2cbyte
6       ;nastavení master, adresa, rychle, 1 byte
7       hi2cout (6,%00000001,%00000000)
8       ;všechny vývody krome GP0.0 na výstup
9       opakovani:
10        ;velký cyklus
11        pause 80
12        ;doba svitu poslední LED a zhasnutí všech
13        hi2cout (0,0,0)
14        ;zacít portem GP0,zhasnout vše
15        w0=1
16        w1=2
17        ;nastavení pocítání cyklu a váhy bitu
18        hi2cin 0,(b4)
19        ;nacíst port GP0 respektive jeho bit 0
20        b4=b4 and 1
21        ;maskovat bit 0 (není nutné)
```

```
22      if b4<>0 then goto opakovani
23      ;v klidu vyckává, stisk TL rozbehne svetlo
24      serie:
25      ;jedno prebehnutí svetla - malý cyklus
26      pause 80
27      ;doba svícení
28      hi2cout (0,b2,b3)
29      ;na oba porty (GP0 první) pošli w1(b2 a b3)
30      w1=w1*2
31      ;posunutí aktivního bitu o jednu vlevo
32      inc w0
33      ;pocítání posunutí
34      if w0<16 then serie
35      ;prebeh dokoncen
36      goto opakovani
```

K programu je třeba dát jedno důležité vysvětlení. Nebyl použit FOR cyklus, protože v té verzi obslužného programu, v níž pracujeme, se FOR a použití povelů pro ovládání I2C sběrnice „nesnáší“ a hlásí chybu, přestože je třeba FOR cyklus v úplně jiné části programu a věcně spolu nesouvisí. Je to pravděpodobně další z odhalených chyb prostředí. Snímek je ze zapojení před připojením LED.



Měření teploty

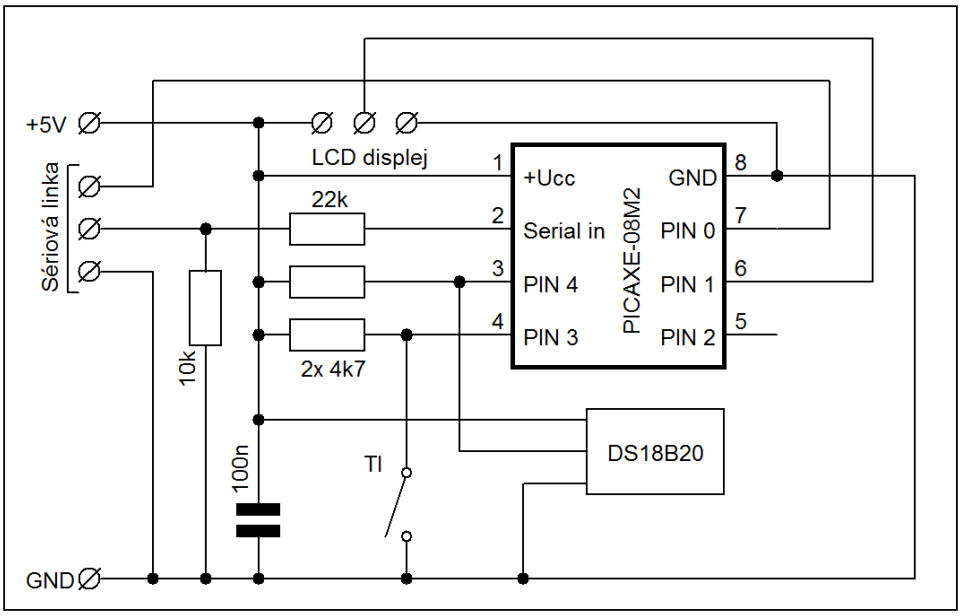
Měření teploty je potřeba poměrně často a může k němu posloužit třeba termistor a snímání napětí na něm AD převodníkem nebo jednoduchý teplotně závislý astabilní klopný obvod a měření doby jeho aktivního signálu nebo periody. Tyto způsoby mají jedno společné, jsou

nelineární, budeme muset snímání teploty nějak adjustovat a vytvořit buď převodní tabulku nebo funkci, která se bude používat k přepočtu. Je to cesta možná, ale na rychlé použití zbytečně složitá a pracná. Každý mikrokontrolér PICAXE 08M2 obsahuje v sobě čidlo teploty, toho můžeme pro méně náročné účely využít.

READINTERNALTEMP

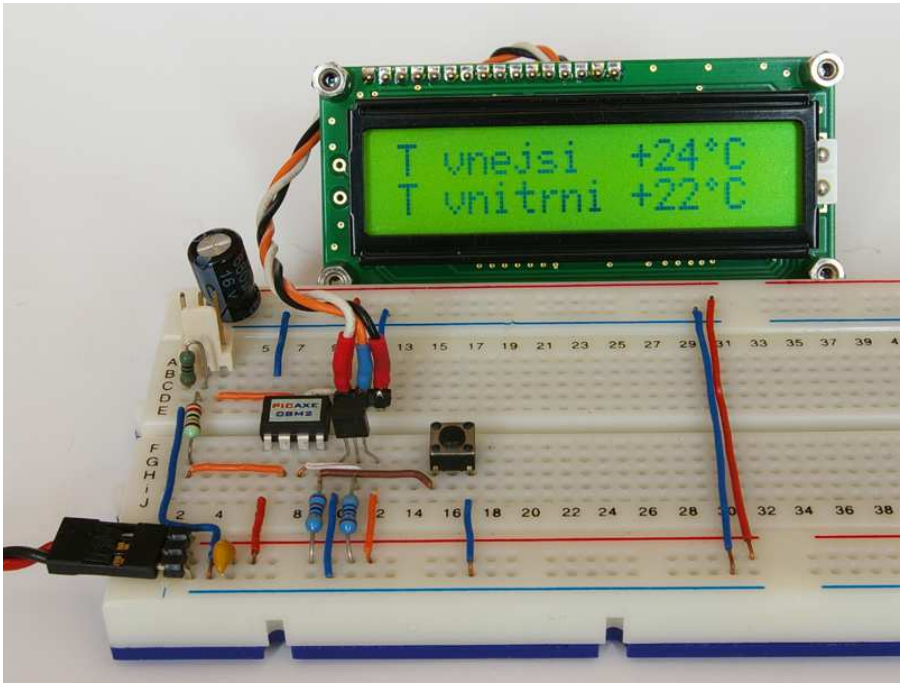
je příkaz, jímž se přečte vnitřní čidlo teploty. Má tři parametry, v prvním se musí zadat napájecí napětí mikrokontroléru, aby se dala provést korekce závislosti na napětí, v druhém se může zadat korekce výstupu měřené teploty (číslo se přičte k výsledku a může být i záporné) a třetí je proměnná typu byte, do níž se teplota přímo ve stupních C přenese. Například READINTERNALTEMP IT_4V5,0,b0 říká, že se má teplota uvnitř čipu měřená při napájecím napětí 4,5 V zapsat bez korekce do proměnné b0. Nesmíme zapomínat, že se měří uvnitř obvodu, který je zahřívaný svými vlastními výkonovými ztrátami, a že změny teploty pronikají dovnitř poměrně pomalu. Pro orientační měření a třeba jako upozornění před přehříváním desky s plošným spojem, na níž je i mikrokontrolér, nebo pro řízení výkonu chladicího ventilátoru, toto vnitřní čidlo postačuje.

Pro přesnější měření teploty je připravena spolupráce s čidlem Dallas DS18B20, které má rozlišení 0,0625°C a v rozsahu -10 až +85°C přesnost lepší než +/- 0,5°C. Výsledek se přenáší do mikrokontroléru sériově jediným vodičem. DS18B20 je sice dražší než třeba obyčejný termistor, ale jeho hlavní výhodou je, že údaj o teplotě získáme jediným příkazem bez nutnosti zabývat se nastavováním nebo přesností. Schéma obvodu, na němž si vyzkoušíme práci s teplotou, je na dalším schématu.



READTEMP

má dva parametry, první určuje vstupní pin, k němuž je připojen signál z čidla DS18B20, druhý bytovou proměnnou, do které se teplota ve stupních C uloží. Kvůli konstrukčním omezením nejde v případě PICAXE 08M2 použít vstupní pin 3 a samozřejmě také pin 0, který je výhradně výstupní. Kladné teploty se uloží přímo, záporné v podobě čísla 128+číselná hodnota teploty.



READTEMP12

je obdobou předchozího příkazu, ale pracuje s proměnnou typu word a ukládá do ní teplotu v 1/16 °C (1 bit = 0,0625°C).

Připravíme si nejprve program, který na sériovém LCD displeji zobrazí v horní řádce (vnější) teplotu snímanou čidlem DS18B20 a v dolní (vnitřní) teplotu mikrokontroléru, u té nebudeme předpokládat, že by mohla být záporná. Na práci s LCD není nic nového s čím bychom se už nesetkali v předchozích příkladech, obě teploty jsou vypisovány na celé stupně C, předpokládá se napájení ze stabilizovaného zdroje 5 V.

```
1       REM Teplomer1 - PICAXE 08M2
2       start:
3         readtemp 4,b0                         ;nactení teploty do b0
4         serout 1,N2400,($FE,$01)              ;inicializace LCD
5         if b0>127 then                        ;záporná teplota
6           b0=b0-128                           ;prepocet
7           serout 1,N2400,("T vnejsi -",#b0,$B2,"C")
8           else                                ;výpis záporné teploty na LCD
9           serout 1,N2400,("T vnejsi +",#b0,$B2,"C")
10          endif                               ;výpis kladné teploty na LCD
11        readinternaltemp IT_5V0,0,b1          ;vnitrní teplota, 5V
12        serout 1,N2400,($FE,$C0)              ;na druhý rádek
13        serout 1,N2400,("T vnitrni +",#b1,$B2,"C")
14        pause 5000                            ;výpis vnitrní teploty + cekání
15        goto start
```

Druhý zkrácený příklad předvádí použití 12ti bitového údaje z čidla DS18B20 k tomu, aby bylo možné teplotu vypsat s rozlišením 0,1°C. Odříznutím dolních čtyř bitů (dělením 16)

získáme celé stupně, pro získání desetin použijeme zbytek po dělení. Desetinné místo není zaokroulované. Pro jednoduchost program opět předpokládá jen kladné teploty.

```
1       REM Teplomer2 - PICAXE 08M2
2       start:
3         readtemp12 4,w0                    ;nactení 12bit teploty do W0
4         serout 1,N2400,($FE,$01)           ;inicializace LCD
5         w2=w0/16                           ;celá část čísla
6         w3=w0//16*10/16                    ;desetinná část
7         serout 1,N2400,("teplota ",#w2,",",#w3,$B2,"C")
8         pause 2000                         ;výpis na LCD a cekání
9         goto start
```

Náhodná čísla

Program by se obecně vzato neměl chovat náhodně, ale někdy je žádoucí, aby náhodu co nejlépe simuloval. Využijeme stále stejný přípravek a pokusíme se vytvořit „hrací kostku“, která by po stisku tlačítka generovala náhodná čísla v rozsahu 1 až 6.

RANDOM

je příkaz, který má jeden parametr, proměnnou typu word. Použitím se do proměnné zapíše pseudonáhodné číslo v rozsahu 0 až 65535. Obsah proměnné by se neměl nijak měnit, protože současně slouží jako výchozí stav pro určení dalšího pseudonáhodného čísla. Výsledek do požadovaného rozsahu nejjednodušeji (i když ne úplně korektně) dostaneme ze zbytku po dělení.

```
1       REM Kostka1 - PICAXE 08M2
2       start:
3         inc w2                             ;pocítání hodu
4         random w0                          ;náhodné číslo do w0
5         w1=w0//6+1                         ;prevod na interval 1..6
6         serout 1,N2400,($FE,$01)           ;inicializace displeje
7         pause 100                          ;pockat na dokoncení
8         serout 1,N2400,("Hod ",#w2,": ",#w1)  ;výpis na LCD
9         cekani:                            ;stisk Tl - další hod
10        if pin3=1 then pause 10 goto cekani endif
11        goto start
```

Čísla, která tato „hrací kostka“ generuje, jsou sice (pseudo)náhodná, ale protože proměnná w0 vychází vždy z hodnoty 0 po zapnutí mikrokontroléru a algoritmus je také stejný, budou vždy stejná, setkáme se stále se stejnou posloupností 3-6-6-6-6-5-4-1-1-1-2-…. I když to tak ze začátku posloupnosti asi nevypadá, je generátor poměrně dobrý. Program „Nahoda“, který zde už uvádět nebudu, ale je v balíčku souborů k tomuto dílu, vygeneruje zhruba během minuty mikrokontrolérem 60000 náhodných čísel a roztřídí je podle hodnoty, výsledek zobrazí přes debug mód. Ideální výsledek by byl po 10000 v proměnných w1 až w6, reálný

se od něj příliš neliší.

Pokud chceme, aby náhodná čísla byla opravdu náhodná (v praxi nepředvídatelná), stačí před prvním použitím příkazu RANDOM vložit do jeho proměnné hodnotu, kterou neumíme přesně zopakovat, a bude vždy jiná. V našem přípravku se nabízí vyzvat uživatele ke stisku tlačítka a změřit dobu tohoto stisku nebo třeba vzít nejvyšší možné rozlišení teploty z teplotního čidla. První z uvedených možností bude současně posledním příkladem, který si v tomto seriálu uvedeme.

```
1     REM Kostka2 - PICAXE 08M2
2     serout 1,N2400,($FE,$01)                ;inicializace displeje
3     pause 100                               ;cekani na inicializaci
4     serout 1,N2400,("Stiskni tlacitko")     ;výpis na LCD
5     serout 1,N2400,($FE,$C0)                ;rádek 2
6     serout 1,N2400,("a okamzik podrz")      ;výpis na LCD
7     setfreq m32                             ;hodiny na 32 MHz
8     cekani1:                                ;cekání na stisk Tl
9     if pin3<>0 then goto cekani1
10    mereni:                                 ;merení doby stisku Tl
11    if pin3=0 then inc w0 goto mereni endif
12    setfreq m4                              ;hodiny zpet na 4 MHz
13    cyklus:                                 ;cyklus házení kostkou
14      inc w2                                ;pocítání hodu
15      random w0                             ;náhodné císlo do w0
16      w1=w0//6+1                            ;prevod na interval 1..6
17      serout 1,N2400,($FE,$01)              ;vymazání displeje
18      pause 100                             ;pockat na vymazání
19      serout 1,N2400,("Hod ",#w2,": ",#w1)  ;výpis na LCD
20      cekani2:                              ;stisk Tl - další hod
21      if pin3=1 then pause 10 goto cekani2 endif
22      goto cyklus
```

Závěr

Tento seriál neměl za cíl probrat všechny dostupné příkazy a suplovat příručku nebo dokonce učebnici, od toho je dokumentace vydávaná výrobcem PICAXE a řada specializovaných publikací a internetových stránek, které se danému tématu věnují. Záměrem bylo vzbudit zájem, na příkladu nejmenšího a nejlevnějšího člena řady mikrokontrolérů PICAXE ukázat jak relativní jednoduchost programování těchto mikrokontrolérů při využití připravených funkcí, tak omezení, na která při tom můžeme narazit. Znovu zdůrazňuji, že hlavní výhodou PICAXE je rychlost a přehlednost řešení jednodušších úloh, nikoli maximální dostupný výkon. Sortiment mikrokontrolérů PICAXE lze najít buď na britských internetových stránkách *www.picaxe.com* nebo v českém internetovém obchodě *www.snailshop.cz*. .

Obsah