ZAVADĚČ

(speciální program pro PIC16F877/876)

Úvodem

Tento dokument se snaží sblížit čtenáře s funkcí takzvaného zavaděče, který umožňuje manipulovat s obsahem FLASH a EEPROM paměti PIC16F877 (nebo PIC16F876) pomocí sériové komunikace, bez nutnosti ICSP rozhraní. Zde se zaměřuji pouze na funkci zavaděče, nikoliv na ovládací program na straně osobního počítače (tento program je též obsahem aktuální přílohy a nese název Loader16. Jedná se již o druhou verzi, která odstraňuje chyby v časování komunikace).

Následující řádky popisují funkci zavaděče, jehož zdrojový kód se stručným komentářem naleznete v souboru "zavadec.asm". Nejsou okomentovány záležitosti, jako je nastavení speciálních registrů, nebo vývody komunikace. To je možno dopátrat na základě manuálu k mikrokontrolérům PIC16F87X.

Hlavní principy zavaděče

Po spuštění (resp. resetování) mikrořadiče se nastaví programový ukazatel na 0000h (vlastnost mikrořadiče). Na adrese 0000h až 0002h je standardní konstrukce, která provede skok na adresu 1E00h (zde je umístěn zavaděč). Manipulace s PCLATH je nutná, protože GOTO adresuje pouze v rozsahu 2048 instrukčních slov. Jakmile je proveden a ukončen zavaděč, dojde ke skoku zpět na adresu 0003h. Odtud může začít uživatelský program. Adresa obsluhy přerušení (0004h) se tedy nemění. Je logické, že adresa mimo oblast 0003h až 1DFF nesmí být přepsána, jinak zavaděč přestane fungovat (je důležité mít na paměti, že zavaděč zabírá 512+3 slov ve FLASH paměti). Naštěstí je zkonstruován tak, že se sám nedokáže přepsat. Mohl by ho však přepsat "nepovedený" uživatelský program, takže si dávejte pozor. Další příčinou přemazání zavaděče může být nesprávné provozování mikrokontroléru (příliš vysoká taktovací frekvence nebo nestabilní oscilátor). Program je potom prováděn chybně a snadno zapíše data i tam, kam nemá (to se týká i funkce zavaděče).

Samotný zavaděč dělá následující věc: chvíli čeká na obdélníkový signál ze sériového portu. Pokud se signál neobjeví, zavaděč končí. Pokud se signál objeví, program podle něj nastaví komunikační rychlost (obdélník je chápán jako znak s kódem 128, tedy spolu se startbitem je to blok osmi stejných bitů). Protože spektrum komunikačních rychlostí mikrokontroléru je omezené, musí být nastavena nejbližší možná rychlost. Poté se program pokusí přijmout obdélníky pomocí sériového portu. Pokud je bezchybně přijat znak s číslem 128, je spojení v pořádku a může být aktivováno komunikační rozhraní. V opačném případě se zavaděč ukončí.

Komunikační rozhraní slouží k přijímání příkazů a obousměrnému posílání dat mezi zavaděčem a druhým zařízením pomocí sériového portu (druhé zařízení je zpravidla osobní počítač). Komunikační protokol bude popsán později.

Navázání spojení

Snažil jsem se o to, aby nebylo nutno k mikrokontroléru připojovat příliš mnoho vodičů a zároveň abych se nemusel omezovat na jednu komunikační rychlost. Jsou zde pouze jeden vstupní a jeden výstupní vodič. Tyto vodiče jsou zprvu konfigurovány jako standardní vstupy (vysoká impedance). Aplikace Loader16 (ve Windows) se snaží navázat spojení tak, že nepřetržitě posílá znak s číslem 128. Společně se startbitem tak vznikají obdélníky s délkou osmi bitů. Zavaděč v mikrokontroléru změří délku tohoto obdélníku a získá tak "relativní" komunikační rychlost. V programovém kódu si všimněte dvou věcí. Obsah TMR0 po změření doby obsahuje (díky vhodnému nastavení předěličky) hodnotu, kterou lze dosadit do registru

SPBRG (komunikační rychlost). Musí se však snížit o jedničku, protože registr SPBRG v podstatě chápe 0 jako 1 a 255 jako 256. Můžete si všimnout (z tabulky v datasheetu), že komunikační rychlosti, které jsou si násobky, nemají soudělné hodnoty v SPBRG. Pokud se tyto hodnoty zvýší o jedničku, potom ano. Vraťme se zpět k měření času pomocí TMR0. Mezi vznikem obdélníkové hrany a spuštěním TMR0 uplyne několik taktů a to je třeba kompenzovat zpožděním konce měření. Zároveň je třeba si uvědomit, že TMR0 počítá jen celé periody (zaokrouhlení dolů). Takže s předděličkou 16 (tu jsem použil) představuje 15 instrukčních taktů stále nulu. Proto jsem kompenzační zpoždění prodloužil tak, aby například hodnota 9 instrukčních cyklů byla změřena jako jeden cyklus TMR0. Bez této kompenzace by se nepodařilo správně rozpoznat některé vyšší komunikační rychlosti (zvláště pak 115200 baudů).

Poté, co se nastaví komunikační rychlost, aktivuje zavaděč hardwarový sériový port a zkusí přijmout znak. Pokud byla rychlost správně rozpoznána, musí přijmout znak s číslem 128. V tom případě může být zahájena komunikace s druhým zařízením (osobní počítač). Této komunikaci bude věnována následující kapitola.

Ještě pár slov k vlastnostem navazování spojení. „Teoreticky“ by mělo být jedno, jakou komunikační rychlost zvolíte na svém počítači a jakou frekvencí taktujete mikrokontrolér. To je však pravda jen částečně. Problém je v tom, že nabízené spektrum rychlostí mikrokontroléru je velmi omezené a prakticky vždy se jedná o jiné rychlosti, než které nabízí sériový port na PC. Pokud například taktujete mikrokontrolér pomocí 4MHz krystalu a pokusíte se komunikovat rychlostí 115200 baudů (nejvyšší rychlost pro PC), nepodaří se vám to. Mikrokontrolér při takové taktovací frekvenci nenabízí žádnou dostatečně podobnou rychlost, tudíž chyba komunikace by byla příliš vysoká. A to i přesto, že stále nabízí rychlosti, které značně přesahují schopnosti portu na PC. Při použití 20MHz krystalu už je možno komunikovat rychlostí, která je zmiňované „115200“ velmi podobná.

Komunikační protokol

Pokusil jsem se vytvořit komunikační protokol, který je bezpečný a snadno odolá chybám. V podstatě se jedná o princip "já pošlu znak a ty odpověz". Každý vyslaný byte ze strany PC je opětován kontrolním bytem ze strany zavaděče. Existuje šest příkazů (resp. pokynů), které mají přesně definovaný účel:

Potvrzení spojení: na znak s číslem 128 zavaděč odpoví hodnotou 229.

Ukončení zavaděče: na znak "Q" zavaděč odpoví hodnotou 0 a ukončí se.

Zápis do FLASH: na znak "W" zavaděč odpoví hodnotou 0 (rozpoznán začátek příkazu). Dále je nutno poslat FLASH-adresu, nejprve nižší byte a potom vyšší byte, na každý z nich zavaděč odpoví vrácením stejné hodnoty (ověřování přenosu). Stejně tak musí být vyslán obsah paměťové buňky (také dva byty, ve stejném pořadí a i s ověřováním). Nakonec je nutno vyslat povel k zahájení zápisu ("G"). Tento povel jsem přidal proto, aby si uživatelský systém mohl ověřit správnost přenosu a poté se rozhodl, zda se zápis uskuteční.

Čtení z FLASH: na znak "R" zavaděč odpoví hodnotou 0 (rozpoznán začátek příkazu). Dále je nutno předat FLASH adresu (platí totéž co pro zápis). Poté je nutno dvakrát po sobě vyslat znak "G". První z nich je opětován nižším bytem paměťové buňky a druhý vyšším bytem. Tím je čtení dokončeno.

Zápis do EEPROM: na znak "E" zavaděč odpoví hodnotou 0. Ostatní je stejné jako u zápisu do FLASH, s tím rozdílem, že se přenáší pouze jeden byte adresy a jeden byte dat.

Čtení z EEPROM: na znak "e" zavaděč odpoví hodnotou 0. Dále je to stejné jako u čtení z FLASH, s tím rozdílem, že se přenáší pouze jeden byte adresy a jeden byte dat.

Následující obrázek znázorňuje průběhy příkazů:

zápis do FLASH

příchozí znaky od uživatelské aplikace	"W"		ADRL		ADRH		DATAL		DATAH		"G"	
odpověď zavaděče		0		ADRL		ADRH		DATAL		DATAH		0

čtení z FLASH

příchozí znaky od uživatelské aplikace	"R"		ADRL		ADRH		"G"		"G"			
odpověď zavaděče		0		ADRL		ADRH		DATAL		DATAH		

zápis do EEPROM

příchozí znaky od uživatelské aplikace	"E"		ADR		DATA		"G"					
odpověď zavaděče		0		ADR		DATA		0				

čtení z EEPROM

příchozí znaky od uživatelské aplikace	"e"		ADR		"G"							
odpověď zavaděče		0		ADR		DATA						

ukončení zavaděče

příchozí znaky od uživatelské aplikace	"Q"											
odpověď zavaděče		0										

potvrzení navázání spojení

příchozí znaky od uživatelské aplikace	128											
odpověď zavaděče		229										

Nyní něco k samotné implementaci rozhraní. Možná se ptáte (po prohlédnutí programového kódu), proč jsou zde takové podivné konstrukce (volání zápisu - část 1,2,3,4,5 a podobně). Rozhodl jsem se totiž pro zvláštní filozofii programováni. Místo toho, abych příkaz zápisu shrnul do jednoho dlouhého podprogramu, rozdělil jsem jej na elementy, které se postupně volají z hlavního programu. Každý element má tu vlastnost, že přijme nový znak, přečtený z portu, provede nějakou činnost a vrátí odpověď. K tomu existuje proměnná STAV, která říká, co se vlastně děje (například 0 znamená, že se čeká na nový příkaz, a 2 znamená, že čekáme na druhý byte FLASH adresy pro zápis). Takže hlavní program dělá jen to, že přijímá znaky ze sériového portu a předává je "interpretu příkazů" (každý znak zvlášť). Tento interpret na oplátku pokaždé odpoví zpětným znakem.

Stále jsem ještě nezdůvodnil, proč jsem takto postupoval. Ve hře je totiž hlídací časovač (16bitový TMR1). Ten se resetuje buď po přetečení nebo po přijetí znaku na sériový port (vše v rámci hlavního programu). Pokud se vyresetuje díky přetečení, vynuluje se i proměnná STAV. V tom je "jádro pudla". Musíme si uvědomit, že porucha v komunikaci (třeba krátké rozpojení) může mít snadno za následek "zamrznutí" (jeden čeká na druhého), nebo alespoň ztrátu synchronizace (co je příkaz a co data?). Pokud tedy osobní počítač delší dobu neodpovídá (v podstatě jen několik milisekund), potom je vynulována proměnná STAV a tím je i vyresetován interpret příkazů. Totéž se stane, pokud dostane interpret

neočekávaný znak (např. nesprávný povel k zápisu). Naopak, pokud chybu rozpozná aplikace na PC, stačí, když krátkou chvíli počká, než se interpret „zresetuje“, a poté může celý příkaz od začátku zopakovat. Pokud je navíc tato aplikace vhodně napsána, můžete během přenosu klidně odpojit a znovu připojit sériový kabel, aniž by došlo k selhání systému. Ten prostě počká na obnovení spojení a následně práci dokončí. Přitom se nemusíte obávat chybného zápisu do FLASH či EEPROM. Tuto vlastnost podporuje i aplikace Loader16 (použijte prosím opravenou verzi, která se nachází v aktuální příloze).

Možná se ptáte, kde je uvedeno potvrzení komunikace. Jednoduše není! Je to jen "vedlejší produkt" komunikačního rozhraní. Toto rozhraní odpoví na každý přijatý byte vlastním bytem. Pokud je navíc proměnná STAV nulová, znamená to, že se čeká na kód příkazu. Příchozí byte je potom rozpoznáván jako kód příkazu pomocí rutiny Z_VYBER. Ta používá opakovaně instrukce "xor". Pokud je výsledek v některém jejím kroku nulový, potom se jedná o určitý příkaz, v opačném případě se pokračuje dále. Pokud se nejedná o řádný příkaz, potom hodnota projde "sítem xorů" až na "dno". Změní se na transformovanou nenulovou hodnotu, která se potom vyšle zpět na sériový port. No a právě hodnota 128 se změní na 229. Toho jsem využil k potvrzení spojení. Nic víc v tom není.

Provádění zápisu a čtení

Jak se zapisuje a čte v rámci FLASH a EEPROM pamětí určitě nemusím vysvětlovat. Měl bych však upozornit na jednu zajímavost zavaděče. Aby zavaděč nemohl přepsat sám sebe, je kód zápisu do FLASH obohacen o kontrolu rozsahu. Pokud je uživatelská adresa FLASH mimo bezpečný rozsah, zápis se jednoduše neuskuteční. Obdobně je to zařízeno u čtení z FLASH. Pokud se pokusíme číst "nepovolenou" paměť vrátí nám zavaděč hodnotu 0. V případě EEPROM není důvod k ochraně.

Ukončení zavaděče

Pokud má být zavaděč ukončen, skočí na blok programu, který navrátí použité registry do stavů, ve kterých byly bezprostředně po spuštění mikrokontroléru (zahlazování stop), a provede skok na adresu 0003h. Přesto doporučuji nespoléhat se v inicializaci uživatelského programu na "defaultní" hodnoty registrů.

Důležité poznámky:

1) *K tomu, aby zavaděč fungoval, musí být splněny podmínky povolující programový zápis do FLASH (viz. konfigurační bity).*

2) *Pokud má být zavaděč použit v mikrokontrolérech PIC16F873 a PIC16F874, je nutno jeho adresu posunout z 1E00h na 0E00h. Zároveň je třeba poupravit povolený adresový rozsah pro zápis do FLASH. Vše ostatní může být zachováno.*

3) *Jak jste si mohli všimnout, zavaděč nevyužívá poslední blok FLASH paměti až do konce. Tato paměť je totiž rezervována pro budoucí rozšíření.*

4) *Pokud používáte obslužnou aplikaci Loader16, vezte, že ačkoliv se jeví zavádění programu relativně svižné (oproti levným ICSP programátorům), je to ještě daleko od možností zavaděče. Problém je v tom, že se neustále komunikuje způsobem: „vyšlu jeden byte a čekám na odpověď“. Tento režim dává prostor vysoké latenci operačního systému. Zkoušel jsem posílat povely do zavaděče pro změnu druhým mikrokontrolérem a rychlost zavádění dat do FLASH byla citelně vyšší, než je tomu s aplikací Loader16 ve Windows. Jenom teoreticky: zapsání slova do FLASH trvá typicky 4 ms. Slov je v paměti celkem 8192 (včetně zavaděče). Na kompletní zápis do celé FLASH paměti je zapotřebí asi 32 sekund. Pro každý zápis slova, musí proběhnout 2x6 bytů (tam a zpět), tedy 12 postupně za sebou. To znamená 98304 bytů. Pokud nastavíme komunikační rychlost na 115200 baudů, potom potřebujeme asi 9 sekund na přenos dat. Pokud to sečteme, teoreticky je zapotřebí asi 41 sekund na přepsání celé FLASH paměti zavaděčem. Ve skutečnosti ale nemůžeme přepsat celou paměť právě kvůli zavaděči. Ale i tak je to zajímavý odhad. Moje zkušenost je však taková, že pokud zapisuji celou přístupnou paměť (7677 slov FLASH a 256 bytů EEPROM), trvá to téměř celou minutu. To by tedy znamenalo, že komunikace trvá v případě „windowsovské“ aplikace asi půl minuty, tedy podstatně déle, než je třeba. Dalo by se tedy říci, že mezi jednotlivými byty vznikají prodlevy, které jsou delší, než přenášené byty samotné.*

5) *Na adrese 1E00h je příkaz ke skoku na adresu 0003h. To proto, aby chybný uživatelský program nespustil zavaděč tím, že se postupně „dosápe“ až k adrese 1E00h.*