PROGRAMOVÁNÍ 32BITOVÝCH MIKROKONTROLÉRŮ V C

POZNÁVÁME PIC32

KDO BY MĚL ČÍST TUTO KNIHU?

PIC32 se ukazuje jako pozoruhodně snadno ovladatelné zařízení, ale přesto je to opravdu výkonný stroj, založený na dobře zavedeném 32bitovém jádře (MIPS) a podporujícím velké množství nástrojů, knihoven a dokumentace. Tato kniha vám může nabídnout pouze malý zběžný náhled do tak rozsáhlého světa a ve skutečnosti říkám, že to je první „průzkum“. Je to moje pevné přesvědčení, že učení by mělo být zábavné a já doufám, že prožijete příjemné chvilky s některými z „hravých“ cvičení a projektů, které uvádím v každé kapitole knihy. Nicméně budete potřebovat zcela určitě přípravu a tvrdě pracovat, abyste mohli strávit materiál, který vám předkládám takovým tempem, že budete rychle akcelerovat přes prvních pár kapitol.

Tato kniha je určena pro programátory se základní až středně pokročilou úrovní zkušeností, ale ne pro „absolutní“ začátečníky; tak neočekávejte, že začneme se základy binárních čísel, hexadecimální notací nebo základy programování. I když budeme stručně zkoumat základy programování v C, tak se to týká vztahu k nejnovější generaci univerzálních 32bitových mikrokontrolérů před přechodem na náročnější projekty. Předpokládám, že vy čtenáři, patříte k jedné z následujících čtyř kategorií:

- Programátor vestavěných řídících aplikací se zkušenostmi v programování v assembleru mikrokontrolérů, ale pouze se základními znalostmi jazyka C.

- Expert přes mikrokontroléry PIC® se základními znalostmi jazyka C.

- Student nebo profesionál s nějakou znalostí programování v C (nebo C++) pro PC.

- Ostatní SLF (superior life forms - nadřazené životní formy): Vím, že programátoři nemají rádi klasifikaci, takže jsem klidně vytvořil tuto speciální kategorii právě pro vás.

V závislosti na vaší úrovni a typu zkušeností, byste měli být schopni najít něco zajímavého v každé kapitole. Pracoval jsem tvrdě, abych se ujistil, že každá z nich obsahuje jak programovací techniky v C, tak nové hardwarové detaily o periferiích. Pokud jste již obeznámeni s oběma, neváhejte přeskočit na odborné sekce na koncích kapitol, nebo zvažte další cvičení, odkazy na knihy a odkazy na další výzkum/čtení.

Zvláštní poznámka je vyhrazena pro ty z vás, kteří jste již četli mou předchozí knihu o programování 16bitových mikrokontrolérech v C. Nejprve mi dovolte, abych vám poděkoval, pak mi dovolte vysvětlit, proč dostanete určitý pocit deja vu (to už tu bylo). Ne, nesnažil jsem se podvádět cestou přes staré 16bitové materiály k vytvoření nové knihy, ale reprodukoval

jsem většinu projektů, abych prakticky demonstroval hlavní tvrzení o architektuře a sadě nástrojů PIC32: je to bezproblémový přechod od 8 a 16bitových aplikací PIC, výrazné zvýšení výkonu a přesto velmi snadné použití. Na konci každé kapitoly jsem pro vás zařadil speciální sekci, kde jsem se zaměřil na vzniklé rozdíly, na vylepšení a další informace, které vám pomohou procházet vaše aplikace rychleji a s větší jistotou.

Toto jsou některé z věcí, které se naučíte:

- Struktura řídícího programu C pro vestavěnou aplikaci: smyčky, smyčky a další smyčky
- Základní časování a operace se vstupy/výstupy
- Základní řídící multitasking v C pro vestavěnou aplikaci s použitím přerušení PIC32
- Nové periferie PIC32 (konkrétní pořadí nespecifikujeme):
    - Zachytávání vstupu
    - Výstup komparace
    - Změna notifikace
    - Paralelní řídící port
    - Asynchronní sériová komunikace
    - Synchronní sériová komunikace
    - Analogově – digitální převod
- Jak ovládat LCD displeje
- Jak vytvořit video signály
- Jak vytvořit audio signály
- Jak přistupovat do velkokapacitních paměťových médií
- Jak sdílet soubory velkokapacitních paměťových zařízení s PC

STRUKTURA KNIHY

Každá kapitola knihy nabízí pro každý den zkoumání 32bitového vestavěného světa programování. K dispozici jsou tři části. První část obsahuje šest malých kapitol s narůstající úrovní složitosti. V každé kapitole se budeme věnovat přezkoumání jedné základní hardwarové periferie rodiny mikrokontrolérů PIC32MX a jednoho aspektu jazyka C s pomocí

kompilátoru MPLAB C32 (studentská verze na disku CD-ROM). V každé kapitole rozvineme alespoň jeden demonstrační projekt. Zpočátku budou tyto projekty vyžadovat výhradní použití softwarového simulátoru MPLAB SIM (součástí sady MPLAB na disku CD-ROM) a nebude nutný žádný skutečný hardware, i když bude možné používat demonstrační desku Explorer 16 nebo Starter kit PIC32.

Ve druhé části knihy, nazvané „Experimentování“ a obsahující pět dalších kapitol, bude vhodné a potřebné více využívat demonstrační desku Explorer 16 (nebo jiné ekvivalenty), protože některé z použitých periferií budou vyžadovat k otestování skutečný hardware.

Ve třetí části knihy s názvem „Rozšíření“ existuje pět větších kapitol. Každá z nich staví na zkušenostech, získaných v několika předchozích kapitolách, zatímco přidáváme nové periférie pro rozvoj projektů s větší náročností. Projekty ve třetí části knihy vyžadují použití demonstrační desky Explorer 16 a také základních dovedností (ano, možná budete muset použít páječku). Pokud nechcete nebo nemáte přístup k základním nástrojům pro výrobu prototypů, pak pro tento případ obsahuje rozšiřující deska (AV32) všechny obvody a komponenty, potřebné k dokončení všech demonstračních projektů a bude k dispozici na stránkách doprovodné webové sítě: http://www.exploringpic32.com.

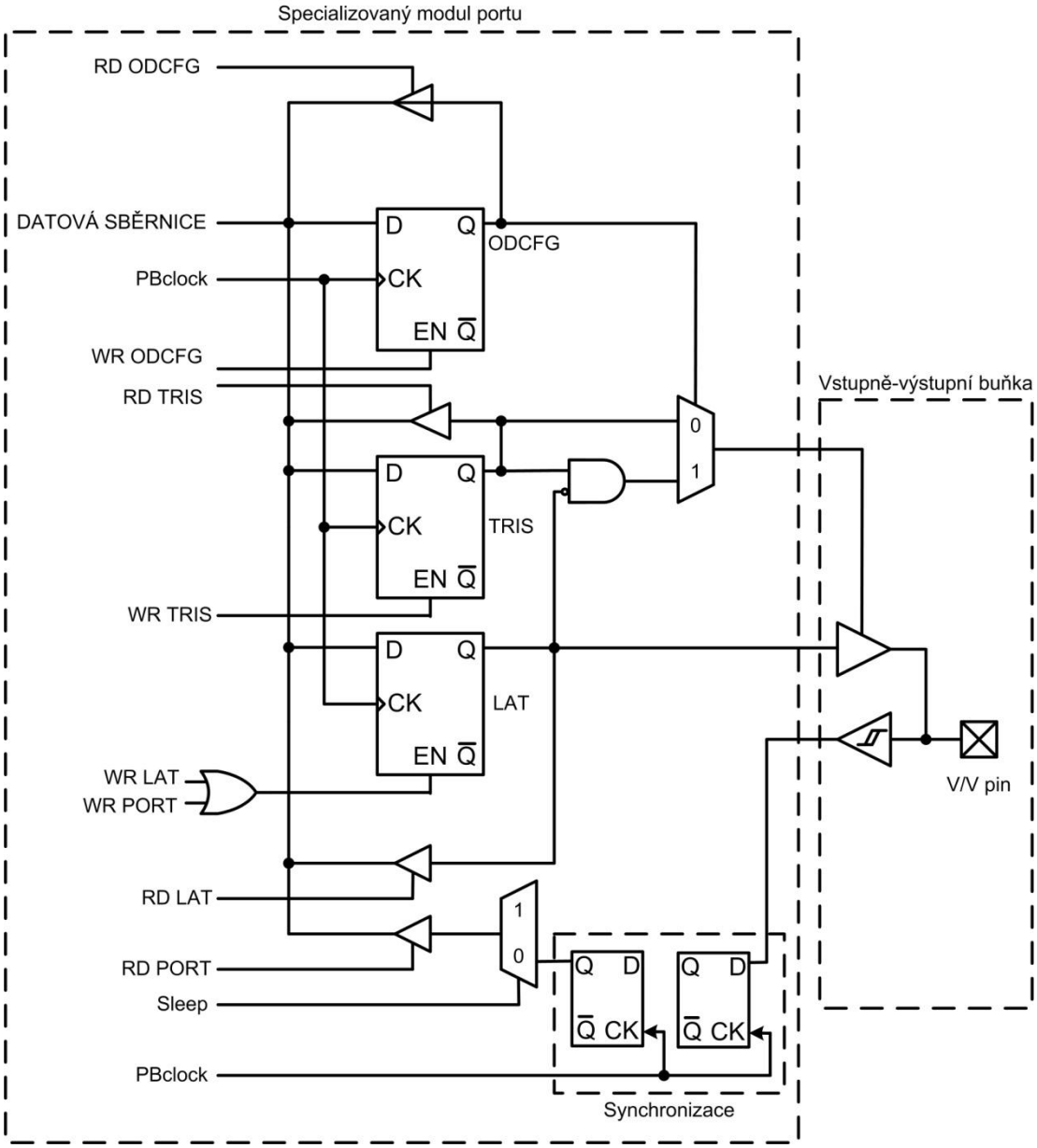
Všechny zdrojové kódy z každé kapitoly jsou také k dispozici k okamžitému použití na doprovodném CD-ROM.

ČÍM TATO KNIHA NENÍ

Tato kniha není náhradou za datový list PIC32, referenční manuál a příručku programátora, kterou vydala společnost Microchip Technology. Také není náhradou za uživatelskou příručku kompilátoru MPLAB C32 a odpovídající knihovny a související softwarové nástroje, nabízené společností Microchip. Kopie jsou k dispozici na doprovodném CD-ROM, ale doporučujeme ke stažení nejnovější verze všech těchto dokumentů a nástrojů z webu Microchip (http://www.microchip.com). Seznamte se s nimi a držte je stále v ruce. Často se na ně v celé knize odkazujeme a prezentujeme malé blokové diagramy a další ukázky sem a tam podle potřeby. Ale toto vyprávění nemůže nahradit informace uvedené v oficiálních příručkách. Pokud si všimnete rozporu mezi výkladem a v oficiální dokumentaci, VŽDY se řiďte tou druhou. Nicméně prosím, pokud vznikl konflikt, pošlete mi email, uvítám vaši pomoc a budeme publikovat jakékoliv korekce a užitečné nápovědy na webové stránce společného webu: http://www.exploringpic32.com.

Tato kniha není slabikářem jazyka C. Přesto, že uděláme v průběhu prvních několika kapitol určitou recenzi „céčka“, čtenář zde najde několik odkazujících návrhů na kompletní úvodní kurzy a knihy na téma jazyka C.

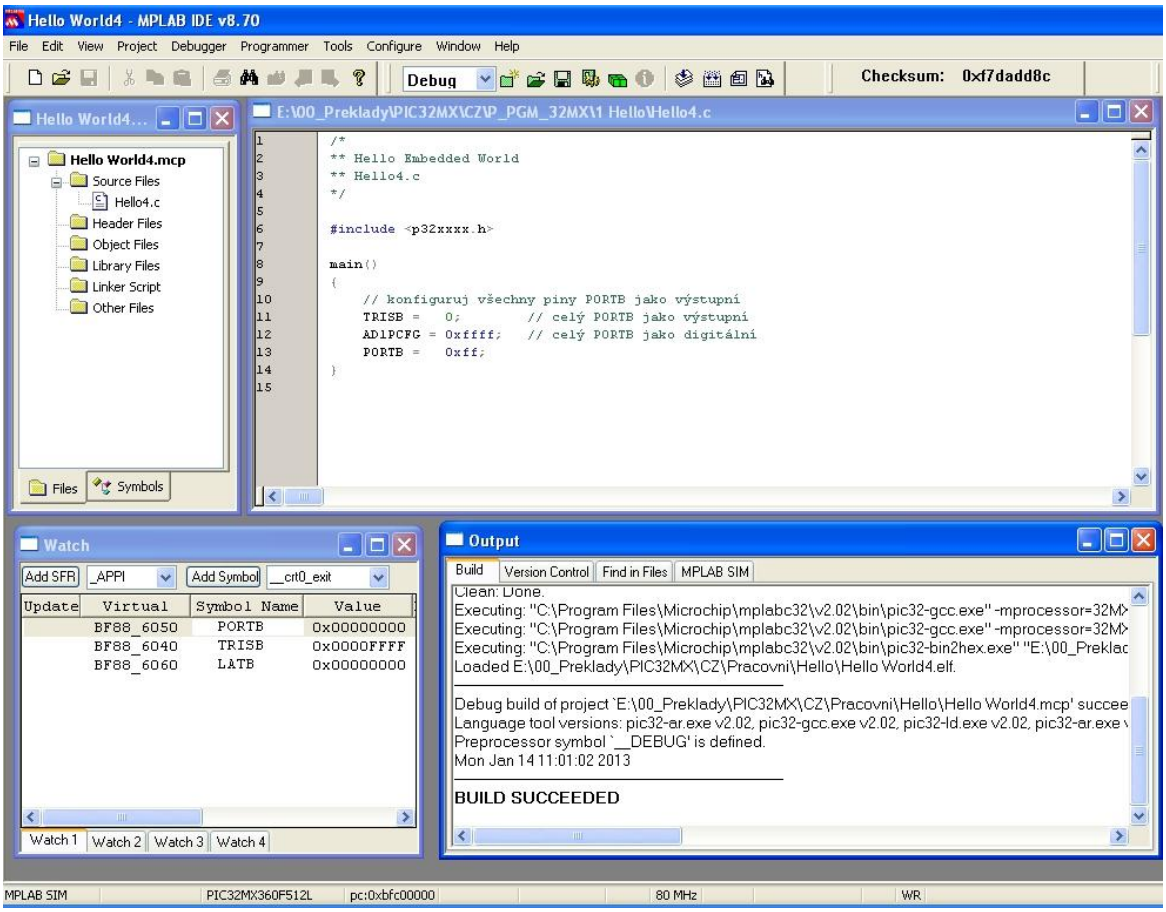
A NĚKOLIK NÁHODNÝCH STRÁNEK NA UKÁZKU…



Obr. 1.5: Blokové schéma typické struktury V/V portu PIC32

Přestože kompletní pochopení schématu na Obrázku 1.5 je nad rámec našich dnešních výzkumů, můžeme začít trochou jednoduchého pozorování. Existují pouze tři signály, které nakonec ovlivňují V/V buňky. Jsou to signály výstupu dat, vstupu dat a třístavový řídicí signál. Posledně jmenovaný je nezbytný pro rozhodnutí, zda má být pin použit jako vstup nebo výstup a je často označován jako *směr* pinu.

A opět z datového listu můžeme určit výchozí směr pro každý pin, který je ve skutečnosti konfigurován jako vstup po každém resetu nebo přivedení napájení do obvodu. Toto je bezpečnostní funkce a standardní pro všechny mikrokontroléry PIC. PIC32 nedělá žádnou výjimku.



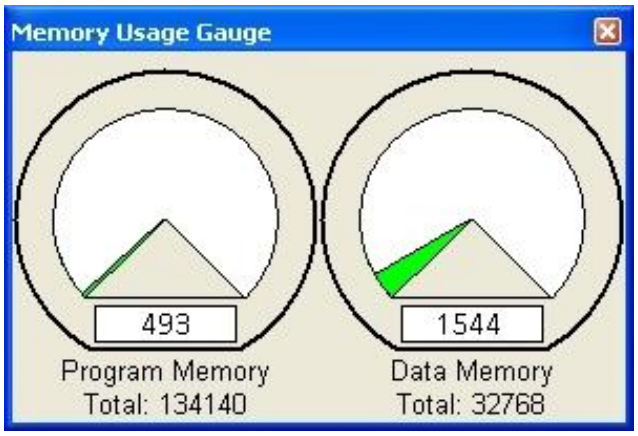
Obr. 1.8: Ahoj vestavěný světe s použitím Portu B

Bohužel, neexistuje nic, čím by kompilátor MPLAB C32 uměl číst naši mysl. Stejně jako v překladu, jsme zodpovědní za stanovení správného směru V/V pinů. Je stále zapotřebí studovat datové listy a učit se o malých rozdílech mezi 8bitovými a 16bitovými mikrokontroléry PIC tak, abychom s nimi byli obeznámeni a samozřejmě také s novým 32bitovým typem.

Ať je programovací jazyk C na jakkoliv vysoké úrovni, myslím, že psát kód pro vestavěné řídící zařízení stále vyžaduje, abychom se důvěrně seznamovali s nejjemnějšími detaily hardwaru, který používáme.

POZNÁMKY PRO EXPERTY V ASSEMBLERU

Pokud máte potíže slepě přijímat platnost kódu, generovaného překladačem MPLAB C32, můžete najít útěchu ve vědomí, že v potřebném okamžiku se můžete rozhodnout k přechodu na zobrazení disassemblovaného výpisu (viz Obrázek 1.9). Můžete rychle zkontrolovat kód



Obr. 1.10: Okno měřidel využité paměti MPLAB IDE

POZNÁMKY PRO EXPERTY MCU PIC

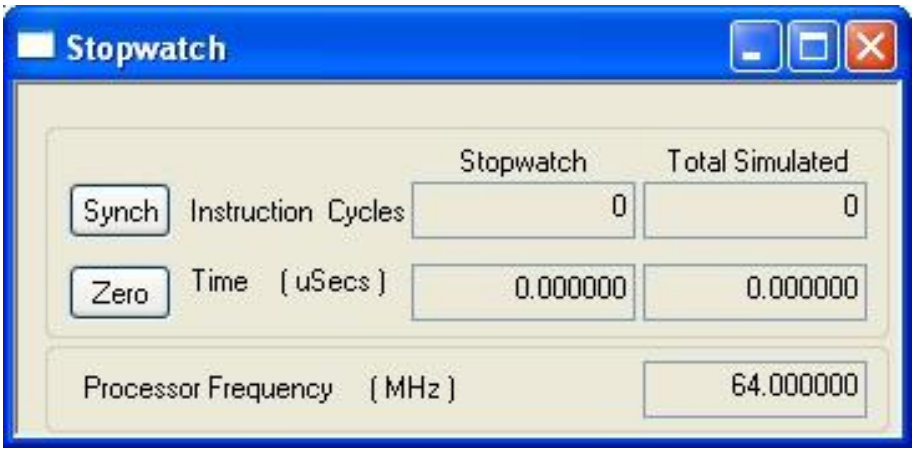
Ti z vás, kteří už jsou obeznámeni s PIC16, PIC18 a dokonce i s architekturou PIC24 bude zajímavé to, že všechny registry SFR PIC32 jsou nyní 32 bitů široké. Ale zejména, pokud jste obeznámeni s architekturou PIC24 a dsPIC, by vám mohlo přijít jako překvapující, že porty nebyly v tomto měřítku zvětšeny! I když PORTA a TRISA jsou nyní 32 bitů široké registry, modul Portu A má stále skupiny menší než 16 pinů, stejně jako v PIC24. Budete si muset v následujících kapitolách uvědomit, že to má několik pozitivních důsledků pro snadnou migraci kódu z 16bitové architektury, zatímco je poskytován optimální výkon 32bitového jádra.

Ať už přecházíte ze světa 8bitových nebo 16bitových PIC/dsPIC, se sadou periférií PIC32 se budete cítit jako doma raz dva!

POZNÁMKY PRO EXPERTY C

Jistě bychom mohli použít funkci *printf()* ze standardních knihoven C. Ve skutečnosti jsou v kompilátoru MPLAB C32 snadno dostupné. Ale my se zaměřujeme na vestavěné řídicí aplikace a nepíšeme kód pro multigigabajtové pracovní stanice. Využívejte k manipulaci nízko-úrovňové hardwarové periférie uvnitř mikrokontrolérů PIC32. Jedno volání knihovny funkcí, jako printf(), by do vašeho spustitelného souboru přidalo několik kilobajtů kódu. Nepředpokládáme, že bude vždy k dispozici sériový port a terminál nebo textový displej. Místo toho rozvíjejte cit pro „váhu“ každé funkce a knihovny, které používáte ve světle omezených zdrojů, dostupných ve světě vestavěného designu.

Po překladu a spojení projektu otevřete okno Stopky (**Debugger|StopWatch**) a umístěte okno podle vašich preferencí (viz Obrázek 4.1). (Osobně se mi líbí, když ho zakotvím do dolní části obrazovky tak, aby se nepřekrývalo s oknem editoru a bylo vždy viditelné a přístupné).



Obr. 4.1: Okno stopek simulátoru MPLAB

Vynulujte časovač stopek a proveďte příkaz Step-Over (**Debug|StepOver** nebo stiskněte klávesu **F8**). Jakmile simulátor dokončí aktualizaci okna stopek, můžete ručně zaznamenat prováděcí čas, potřebný k provedení celočíselné operace. Čas je simulátorem zobrazován ve formě počtu cyklů a v mikrosekundách, odvozených z počtu cyklů, vynásobených simulovanou taktovací frekvencí, parametr je specifikován v nastavení debuggeru (karta **Debugger|Settings|Osc/Trace**).

Pokračujte nastavením kurzoru na další násobení a proveďte příkaz **Run To Cursor** nebo prostě pokračujte **StepOver**, dokud se tam nedostanete. Znovu vynulujte stopky (**Zero**), proveďte **Step-Over** a zaznamenejte druhý čas. Pokračujte, dokud neotestujete všech pět typů (viz Tabulka 4.3).

Test násobení	Šířka (bity)	Počet cyklů	Relativní výkon pro:	
			Int	Float
Celá čísla char (char)	8	6	1	-
Celá čísla short (short)	16	6	1	-
Celá čísla (int, long)	32	6	1	-
Celá čísla long (long long)	64	21	3,5	-
Plovoucí čárka jednoduchá přesnost (float)	32	71	11,8	1
Plovoucí čárka dvojitá přesnost (long double)	64	159	26,5	2,23

Tab. 4.3: Výsledky testu relativního výkonu za pomoci MPLAB C32 rev. 0.20 (všechny optimalizace zakázány)

úprav na příkladu kódu. Uvidíte, jak se rutina přerušení volá znovu a znovu a nezbývá mnoho času, který se stráví uvnitř hlavní smyčky. V našem jednoduchém příkladu to není žádná velká ztráta, souhlasím, ale v praktické aplikaci by to byla katastrofa. Když je příliš mnoho přerušení, příliš často, nebo je jednoduše špatně řízeno, může se hlavní program úplně zastavit. Je naší povinností se ujistit, že rutina handleru přerušení, včetně prologu a epilogu, opravdu nepoužívá všechny dostupné cykly procesoru.

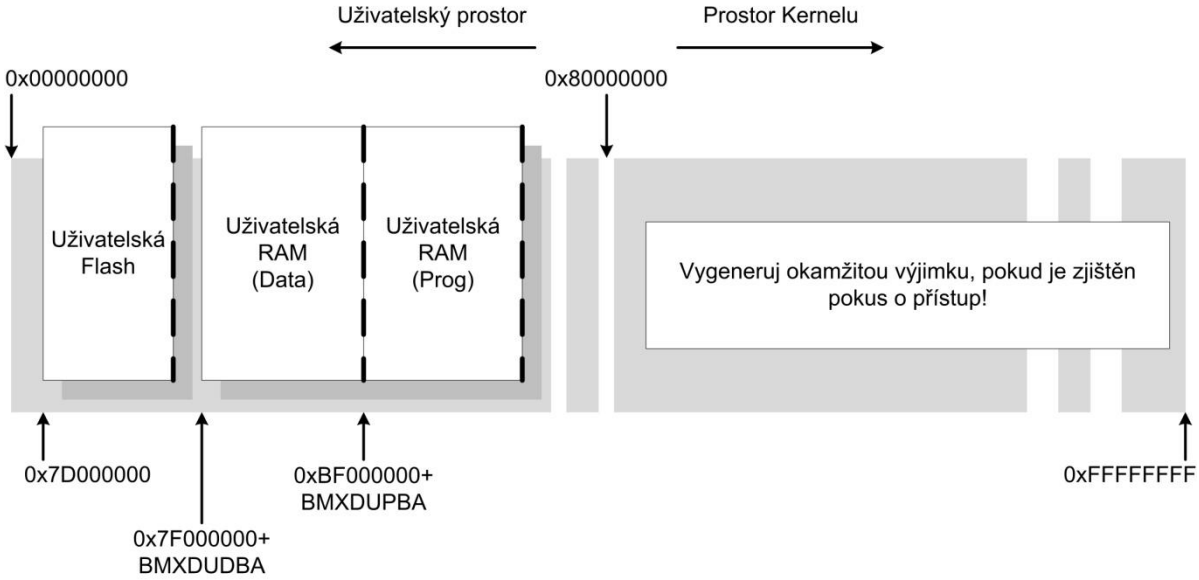
SPRÁVA VÍCENÁSOBNÝCH PŘERUŠENÍ

Pokud pro aplikaci používáme více zdrojů přerušení, pak přiřazením různých úrovní priority pro každý zdroj řešíme pouze jednu část problému. Prioritou se rozhodne, kdo obdrží obsluhu jako první, pokud se současně vyvolaly dvě nebo více událostí přerušení. Ale když je jedno z (mnoha) přerušení obsluhováno, ostatní budou muset počkat, až na ně přijde řada, aby se mohla obsloužit. Nicméně, v některých případech aplikace vyžaduje nejen vícenásobné přerušení, ale také schopnost vnoření se (nest) do volání přerušení. Pokud je přijato přerušení s nižší prioritou a je zpracováváno ISR, může přijít přerušení s vyšší prioritou a vyžadovat okamžitou pozornost, v závislosti podle pořadí přerušovacího handleru.

Pokud chcete povolit možnost vnoření volání přerušení, budete muset „ručně“ znovu povolit přerušení ihned po vstupu do handleru přerušení (pomocí instrukce assembleru MIPS), namísto čekání na kód epilogu, který to udělá automaticky při výstupu z obsluhy přerušení.

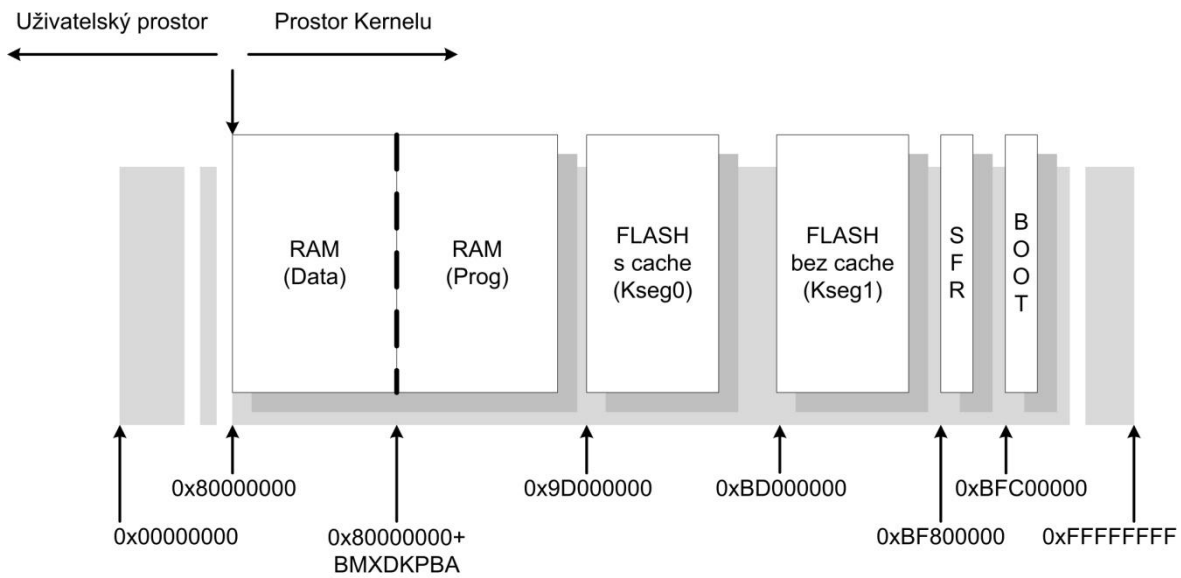
Zde je jednoduchý příklad, který rozšiřuje náš první projekt v imaginární aplikaci, kde je Timer3 použit ke tvorbě druhého periodického přerušení s vysokou/vyšší prioritou (úroveň 3):

```
/*
** Vnoření jednoduchého vektoru přerušení
*/
#include  <p32xxxx.h>
#include  <plib.h>

int count;

void __ISR (0, ipl1) InterruptHandler (void)
{
    // 1. Bezprostřední opětné povolení přerušení (vnoření)
    asm (" ei ");

    // 2. Kontrola a přednostní obsluha nejvyšší priority
    if ( mT3GetIntFlag())
        {
count++;
```



Obr. 6.9: Virtuální paměťová mapa v uživatelském módu

To je skvělá zpráva, protože to znamená, že od nynějška můžeme zcela ignorovat spodní polovinu Tabulky 6.1 a soustředit veškerou svojí pozornost na jedinou virtuální mapu módu kernel (viz Obrázek 6.10)!

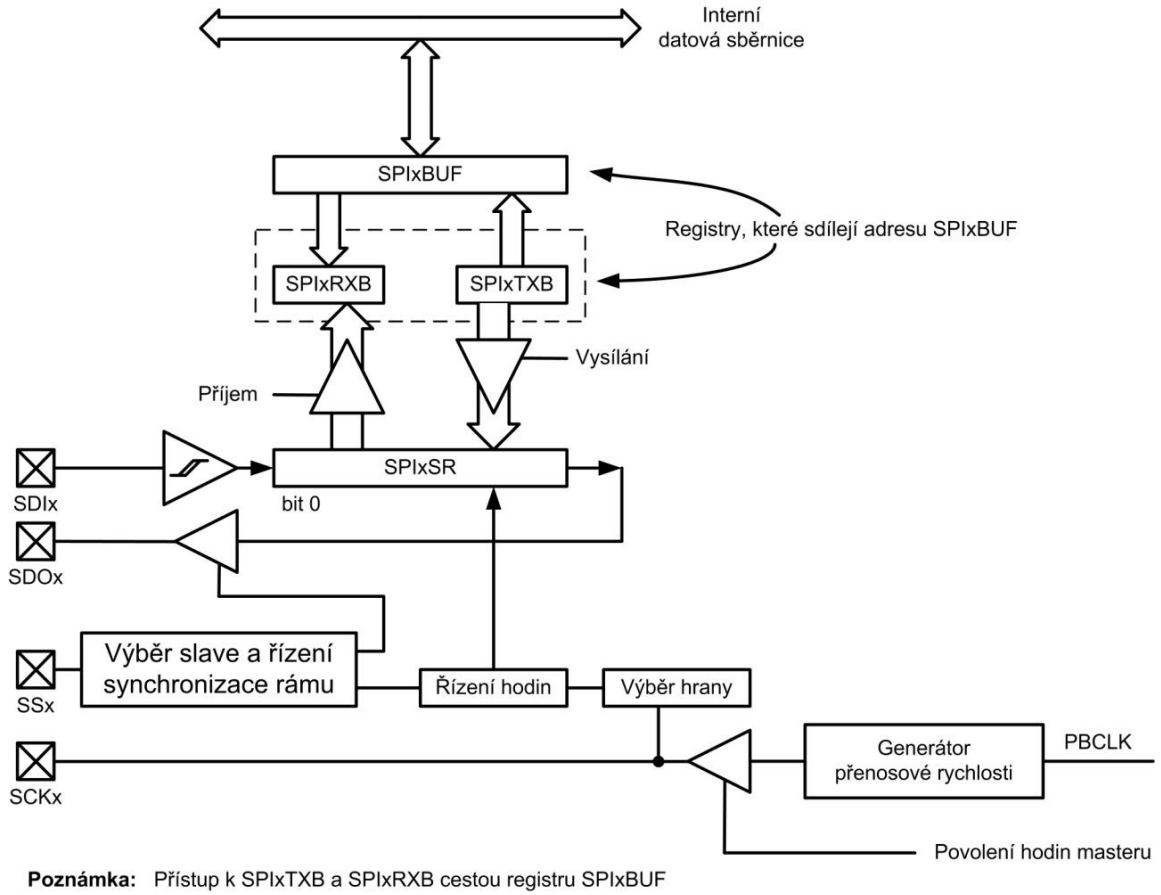


Obr. 6.10 Mapa virtuální paměti PIC32 pro vestavěné řízení (mód kernel)

Poslední poznámka vyžaduje objasnění důvodu dvou virtuálních adresových prostorů vzhledem k zaměření kernelového programu do paměti Flash. V literatuře MIPS jsou tradičně označovány jako *kseg0* a *kseg1*. Pokud se podíváte na sloupce fyzické adresy v Tabulce 6.1,

Rozhraní SPI je snad nejjednodušší ze všech dostupných rozhraní, i když implementace PIC32 je obzvláště bohatá na možnosti a zajímavé funkce.

Rozhraní SPI (viz Obrázek 8.5) se v podstatě skládá z posuvného registru. Bity jsou současně z linky SDI posouvány jako první do nejvýznamnějšího bitu (MSB) a posouvány ven z linky SDO a synchronizovány s hodinami na pinu SCK. Velikost posuvného registru může být proměnná v hodnotách 8, 16, nebo 32 bitů.

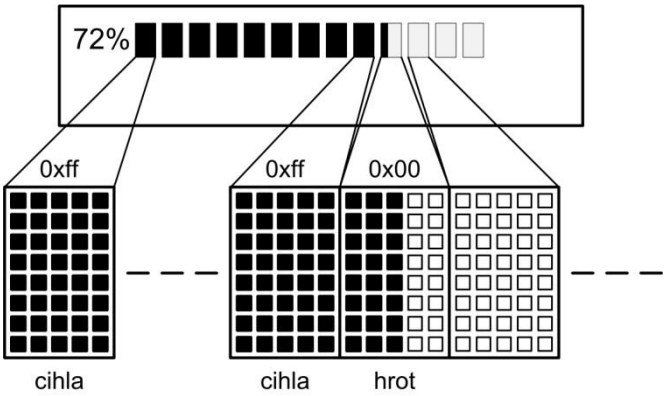


Obr. 8.5: Blokové schéma modulu SPI

Pokud je zařízení nakonfigurováno jako master sběrnice, jsou hodiny generovány interně, odvozené z periferní sběrnice hodin (Fpb), pomocí generátoru přenosové rychlosti a výstup je přiveden na pin SCK. Jinak je zařízení připojeno jako podřízené vůči sběrnici a přijímá taktovací kmitočet z pinu SCK.

```
#pragma        config   POSCMOD = XT, FNOSC = PRIPLL
#pragma        config   FPLLIDIV = DIV_2, FPLLMUL = MUL_18, FPLLODIV = DIV_1
#pragma        config   FPBDIV = DIV_2, FWDTEN = OFF, CP = OFF, BWP = OFF

#include    <p32xxxx.h>
#include    <explore.h>
#include    <LCD.h>
```

Mohli bychom nakreslit pohyblivý pásek, sestavený z bloků, pouze pomocí řetězce (až) 16 znaků „cihliček“, které lze získat z tabulky písma LCD výběrem kódu 0xff, což je pevný vzor 5 x 8 černých pixelů. Ale abychom získali jemnější rozlišení a plynulejší pohyb, můžeme raději plně využít možnost uživatelského definování znaku tak, jak jsme se právě naučili. Trik spočívá ve vytvoření většiny pohyblivého pásku z (5 x 8) kostiček a pak definovat jeden nový znak požadované tloušťky na hrotu (viz Obrázek 10.8).

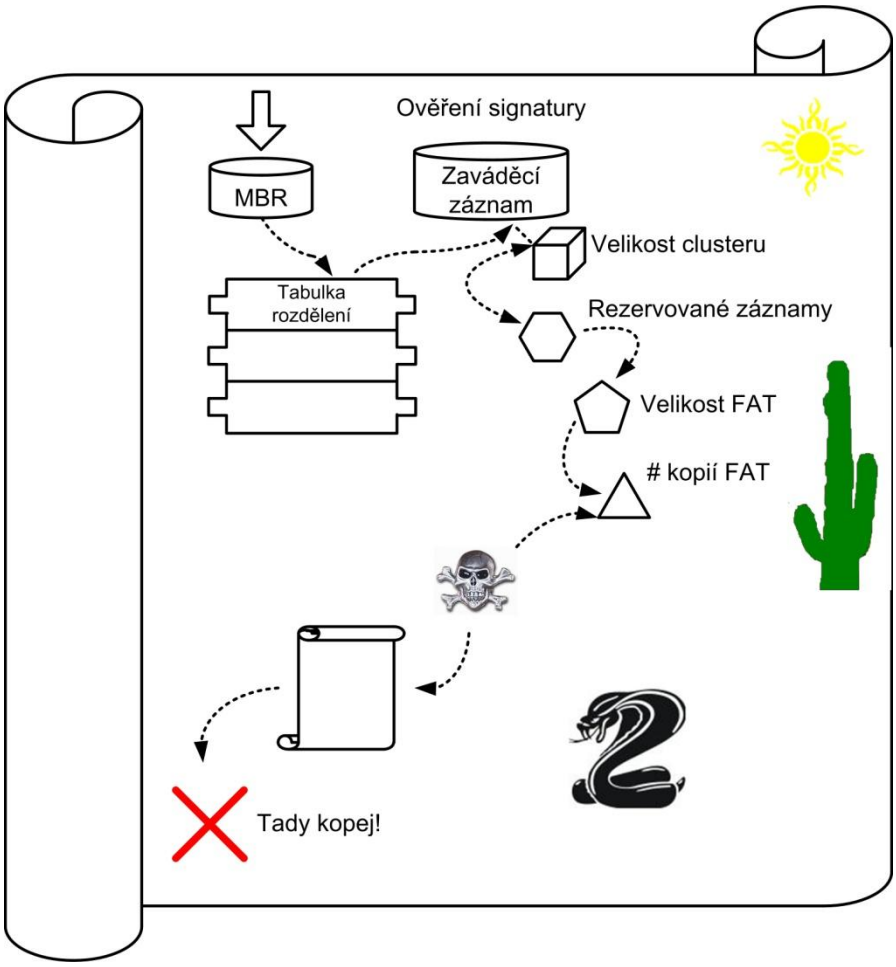


Obr. 10.8: Výkres pohyblivého pásku

Zde je kód, požadovaný k definování hrotu pohyblivého pásku dané tloušťky:

```
void newBarTip(int i, int width)
    {
        char bar;
        int pos;

        // ulož pozici kurzoru
        while(busyLCD());
        pos = addrLCD();

        // tvorba nového znaku na pozici i
        // nastaví datový ukazatel do bufferu CGRAM LCD
        setLCDG( i*8);

        // jako horizontální pásek (0-4)x tloušťka posunu zleva doprava
// 7 pixelů výška
```

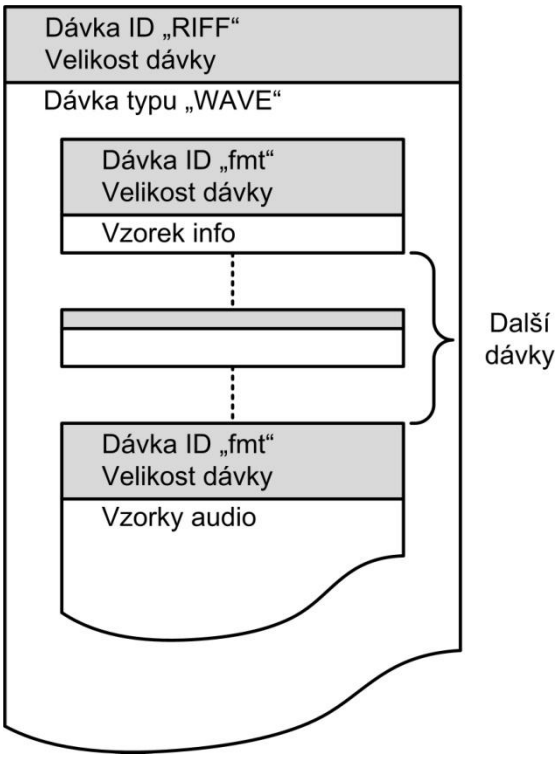
Získat přístup k MBR a najít tabulku oddílů je trochu jako dostat mapu s novou sadou symbolů a vodítek, které je potřeba interpretovat (viz Obrázek 15.7).



Obr. 15.7: Mapa pokladu

Výběrem 32bitového slova najdeme ofset FO_FIRST_SECT (0 x 1C6) jako součást prvního (a jediného, dle našich předpokladů) záznamu v tabulce oddílu, získáme adresu (LBA) úplně prvního sektoru oddílu.

```
// 9. Získá první sektor prvního oddílu - > Boot Record – zaváděcí záznam
firsts = ReadL(buffer, FO_FIRST_SECT);
// 10. Získá sektor (boot record)
if (!readSECTOR( firsts, buffer))
    {
        free(D);        free(buffer);
        return NULL;
    }
```



Obr. 16.13: Rozložení základního souboru WAVE

Dávka *fmt* obsahuje definované posloupnosti parametrů, které plně popisují proud vzorků, které následují v *datové* dávce, jak je reprezentováno Tabulkou 16.3.

Ofset	Velikost	Popis	Hodnota
0x00	4	Dávka ID	fmt
0x04	4	Velikost dávky Data	16 + extra formátové byty
0x08	2	Kompresní kód	Celé číslo bez znaménka
0x0a	2	Počet kanálů	Celé číslo bez znaménka
0x10	4	Vzorkovací rychlost	Celé číslo bez znaménka long
0x12	4	Průměr bajty za sekundu	Celé číslo bez znaménka long
0x14	2	Zarovnání bloku	Celé číslo bez znaménka
0x16	2	Významné bity na vzorek	Celé číslo bez znaménka (>1)
0x18	2	Extra formátové bajty	Celé číslo bez znaménka

Tabulka 16.3: Obsah dávky fmt

Mezi dávkami *fmt* a *data* by mohly existovat další dávky, které obsahují přídavné informace o souboru, takže bychom také mohli prohledávat ID dávek a přeskakovat v položkách seznamu, dokud nenajdeme (*datovou*) dávku, kterou hledáme.