...alebo 80 MIPS v jednom MIPS

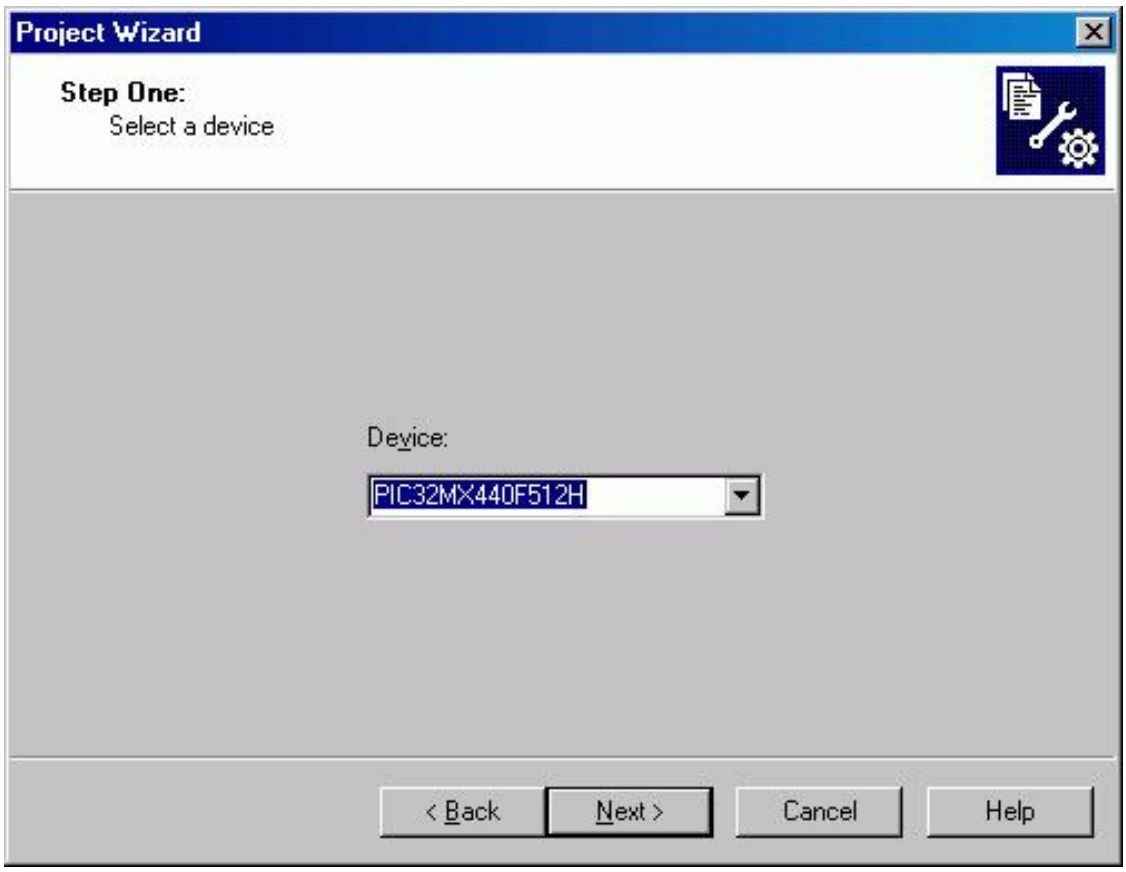
1, Prečo

Vždy sa dobre bavím pri pohľade na ARM megahnutie v posledných rokoch, na všetkých horlivých zastupiteľov jedinej správnej platformy, na firmy, ktoré sa snažia odtrhnút si z koláča čo najväčší kus a nakoniec mi je trochu ľúto tých, ktorí sa z tej motanice a záplavy totálne nepoužiteľných devkitov (niekto zabudol, že to „dev“ znamená development) snažia vymotať. Motanica spočíva predovšetkým v tom, že výrobcovia sa spoliehajú na to, že ARM je natoľko bežná a jasná platforma, že netreba robiť nijaké špeciálne vývojové nástroje, pretože „všetko je dostupné“. Existujú aj rôzne okliešťené verzie komerčných a nie veľmi lacných IDE, ponúkajúcich takmer všetko – a na prvý pohľad sa zdá, že to „na nejakú dobu musí stačiť“. Nakoniec sa ukáže, že to nie je až také veselé a ľudia sa snažia donútiť molochov ako Eclipse spolupracovať s nejakým derivátom GCC, k tomu treba dolepiť debugger alebo simulátor, ak nejaký existuje. Samozrejme, verzií je veľa, treba zvoliť tú správnu, už len zistiť na diskusnom fóre akú verziu... a tak sa môže vývojár namiesto práce s jednočipom, na dlhú dobu utopiť vo vyvíjaní vývoja na vývoj. Ku všetkému na pozadí hrá rozprávka o tom, že ARMy sú „multi-sourced“, čo znamená, že sú dostupné z viacerých zdrojov a keď jeden výrobca prestane dodávať (preto lebo zbankrotuje, alebo mu vyhorí fab, alebo všetci zamestnanci dostanú strašnú úplavicu z majonézového šalátu, ktorý bol na obed), tak sa dá „nejaký ARM“ kúpiť inde. No, tak toto platilo kedysi, keď napríklad 6502 vyrábala materská firma MOS Technology a ten istý kremík vyrábala napríklad aj firma Rockwell alebo Synertek (zdroj) a teda stačilo vytiahnuť z pätice jeden procesor, strčiť tam iný a všetko fungovalo ako predtým. Sazmorejme, s ARM MCU je situácia iná. Každý ma iné periférie, iné usporiadanie v pamäťovom priestore, iné rozloženie vývodov, takže ilúzia o zmene ARM->ARM nadobúda viac na svojej éterickosti. Z toho vyplýva, že akýkoľvek problém s jedným výrobcom ARMov znamená obrovský problém pre jeho zákazníkov a migrácia na iný ARM bude tak isto bolestivá ako migrácia na akýkoľvek iný typ MCU.

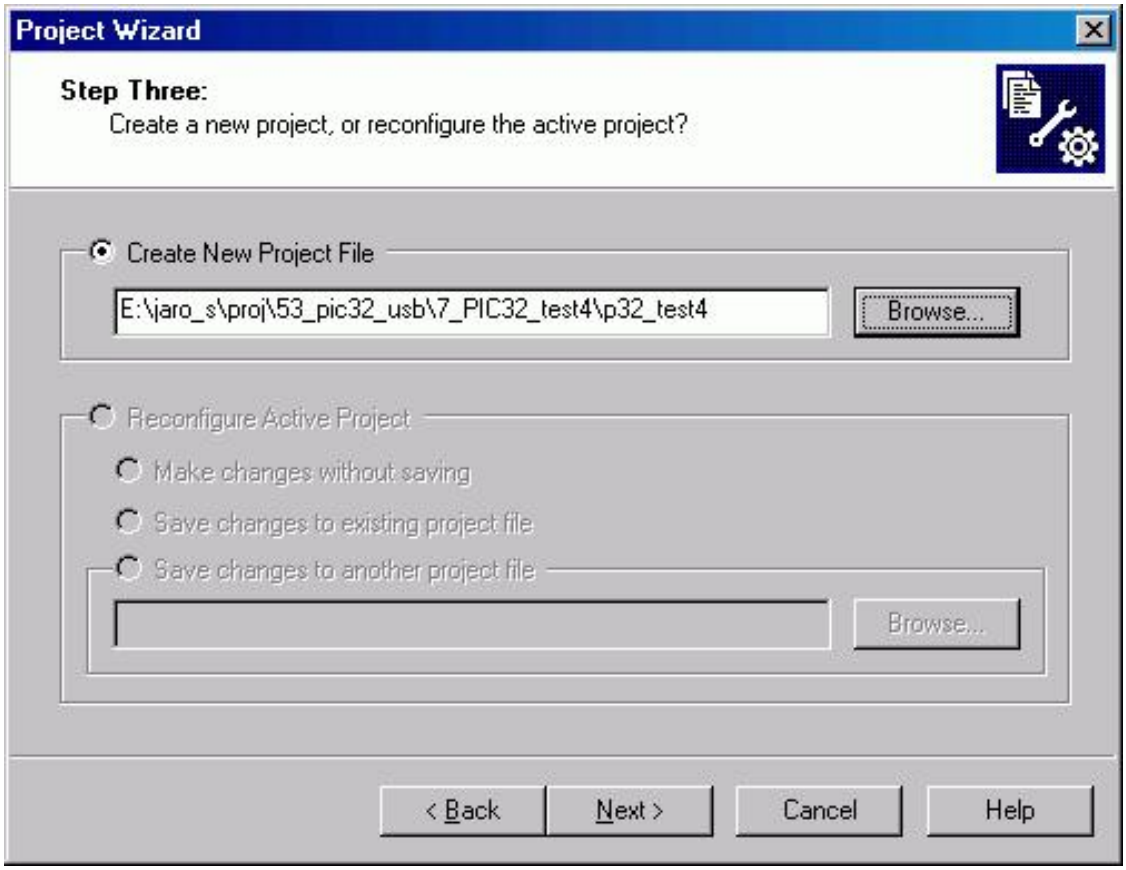
Našťastie nie všetci výrobcovia mikrokontrolérov podľahli davovej psychóze na tri písmená – jedným z nich je Microchip (vlastne, koľko je ďalších?). Microchip pred zhruba tromi rokmi prišiel s rodinou PIC32, čo je MIPS jadro obalené perifériami a pamäťou tak, aby z toho bol jednočip. V ponuke sú kúsky s 512kB FLASH a 128kB RAM a množstvom periférií (napríklad), kusovka stojí pri naj high-endovejších typoch cca 8EUR. Porovnávať výkon s bežnými ARM MCU je problematické a nikdy som ho nemal veľmi rád, ale dá sa o tom niečo dočítať napríklad tu , ale zdá sa, že nové ARM MCU pomaličky začínajú doťahovať na PIC32, ktoré sú 3 roky staré. Dá sa predpokladať, že počas tejto doby v Microchipe nespali, tak uvidíme čo navaria.

2, Ako

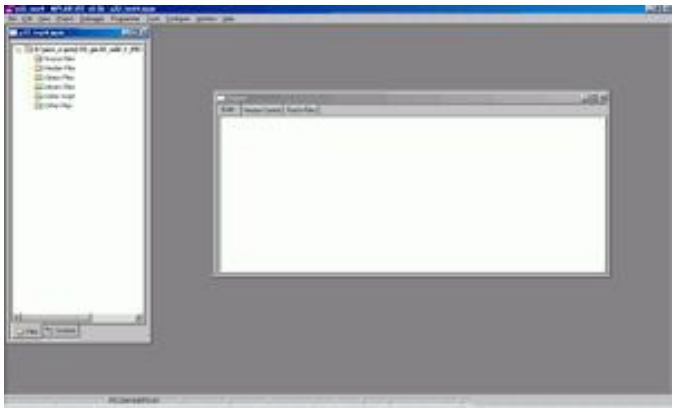
Pre mňa bol na začiatku hrania sa (a neskoršej komerčnej práce) s PIC32 v roku 2008 bol záujem o platformu, ktorá poskytuje viac možností, než tie, na ktoré som bol zvyknutý. A bolo to pomerne triviálne. Vzal som existujúci plošný spoj, na ktorom som mal PIC24 MCU, ten som odfúkol preč a na jeho miesto som prispájkoval PIC32. Tým bolo hranie sa s HW skončené. HW vývojové nástroje som použil presne tie isté ako pre PIC18 alebo PIC24 – teda PicKit3 a ICD3. SW vývojové nástroje sú opäť tie isté – MPLAB, ako doteraz a samozrejme bolo treba stiahnuť a nainštalovať si prekladač MCC32 od Microchipu. Týmto úkonom, ktorý trval 15 minút vrátane konzumácie polovice feferónovej pizze, som ukončil všetky potrebné úkony, už zostalo len duševne sa naladiť na 32-bitovú platformu. Bolo to pomerne jednoduché. To sa vezme MPLAB s nainštalovaným MCC32 kompilátorom a založí sa nový projekt (Project->Project Wizard).



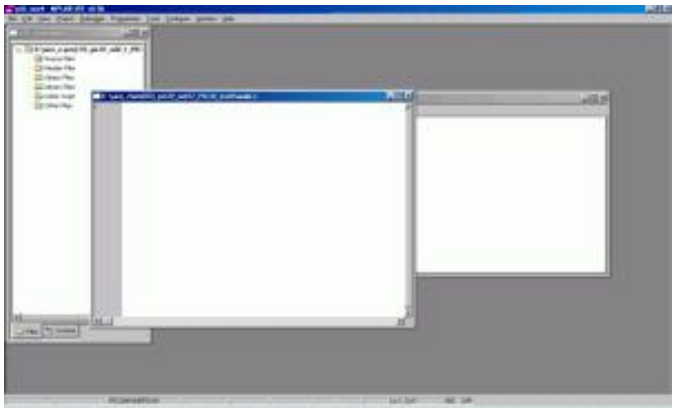
Zvolíme trebárs s PIC32MX440F512H alebo čo už máme na doske.



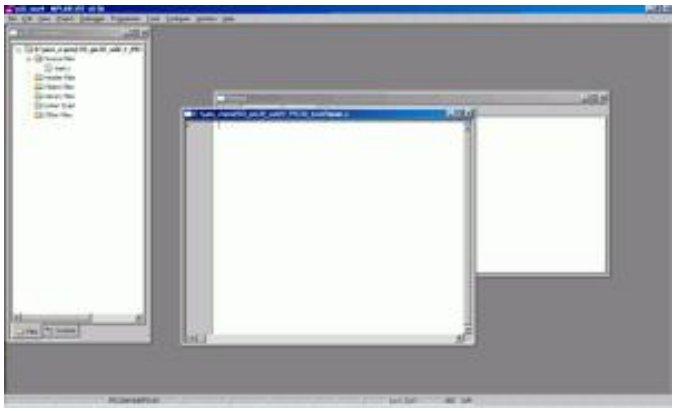
Zvolíme Microchip PIC32 Compiler.



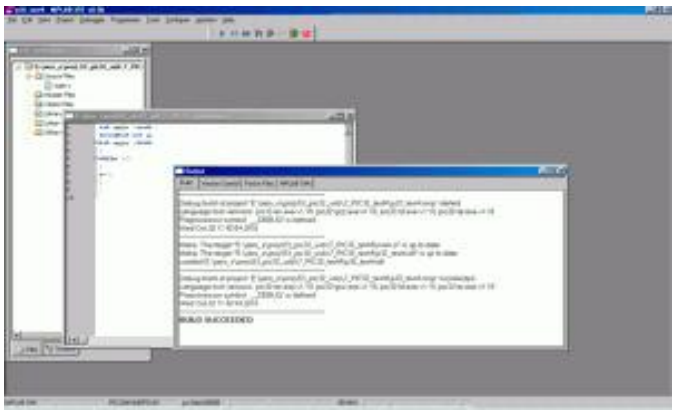
Zvolíme si cestu kde bude projekt



A po pár kliknutiach máme prázdny projekt.



Vytvoríme si nový súbor a uložíme ho ako trebárs main.c do aktuálneho projektu.

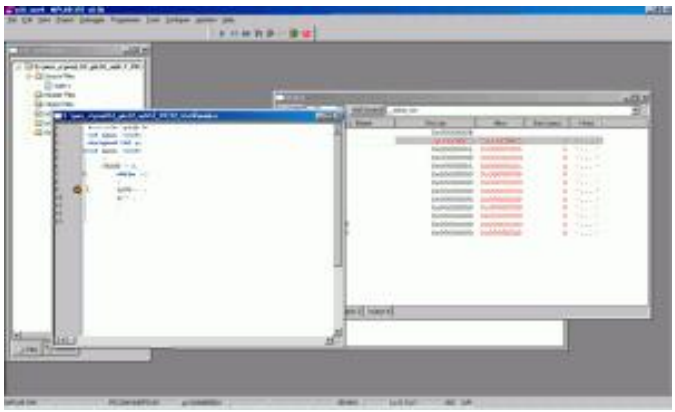


A vložíme ho medzi zdrojové súbory.

Tak a teraz je to pripravené na programovanie.
Zadáme program napríklad takýto:

```
1. int main (void);
2. unsigned int a;
3. int main (void)
4. {
5. while (1)
6. {
7. a++;
8. }
9. }
```

v položke „Debugger->Select tool“ zvolíme MPLAB Simulator a potom F10 – Build. Malo by to vyzerať asi takto:

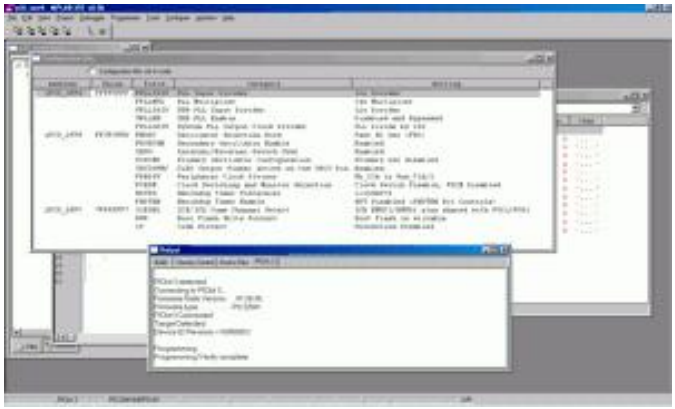


No, a sme doma.

Na to, aby človek mohol hýbať perifériami MCU, tak potrebuje includovať súbor „plib.h“ (nebudem rozoberať jeho štruktúru, ale podľa toho, aký MCU je zvolený, vloží ďalšie hlavičkové súbory s názvami registrov pre konkrétny MCU, okrem iného) a potom sa dá manipulovať s perifériami, asi takto:

```
1.  #include "plib.h"
2.  int main (void);
3.  unsigned int a;
4.  int main (void)
5.      {
6.      TRISD = 0;
7.          while (1)
8.          {
9.          LATD++ ;
10.         a++ ;
11.         }
12.     }
13.
```

A vo watch window (View -> Watch) je možné sledovať ako sa ten port hýbe (na PORTD++ je breakpoint).



Takýto program skutočne hýbe pinmi portu D, ale pomerne rýchlo – na blikanie LEDkou to treba spomaliť. Na to sa hodí napríklad časovač, tak ho použime:

```
1. #include "plib.h"
2. int main (void);
3. unsigned int a;
4. int main (void)
5. {
6. TRISD = 0;
7. T1CONbits.ON = 1;
8. T1CONbits.TCKPS = 0b10;
9. PR1 = 50000;
10. while (1)
```

```
11.
        {
12.
        if (IFS0bits.T1IF==1)
13.
            {
14.
            IFS0bits.T1IF=0;
15.
            LATD++;
16.
            a++;
17.
            }
18.
        }
19.
    }
20.
```

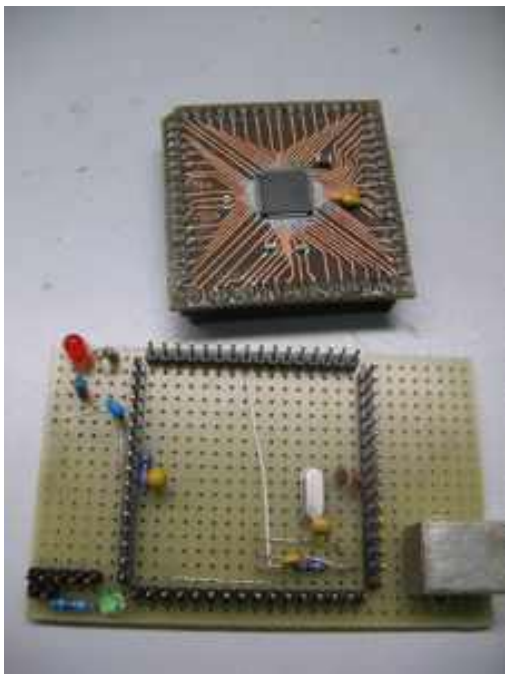
Časovač sa zapne, zvolí sa mu vhodná preddelička a čaká sa v nekonečnej slučke na jeho pretečenie (na interrupt flag, ale pretože prerušenie pre tento časovač nie je povolené, tak nenastane). Ak nastane toto pretečenie (interrupt flag), vynuluje sa príznak a inkrementuje port D. Časovač je nastavený na pretečenie po 50000 inkrementáciach, tieto sú odvodene od frekvencie hodín vydelené číslom 64. Pokiaľ je frekvencia hodín 8MHz, tak inkrementácia portu D nastane 2,5x za sekundu. Dá sa to overiť v simulátore (Debugger->Stopwatch, po nastavení taktu 8MHz v Debugger->Settings).

Teraz je program pripravený na napálenie do MCU. Treba si vhodne zvoliť konfiguračné bity (Configure->Configuration Bits), zvoliť PicKit3 (alebo čo už máme) ako programátor (Programmer->Select Programmer) a napáliť (tlačidlo Program). LEDka pripojená na pin PORTD0 (pin 36) by mala blikať.

Takže blikanie LEDkou, čo je elementárna vec pri novom MCU, máme zvládnuté, spolu so základnou orientáciou v MPLABe. Zdroják má cca 20 riadkov aj s chlpami, nepoužívajú sa žiadne „HW knižnice“ (grrr), je prehľadný a jasný.

3, Čo k tomu treba

Sám som to zbastlil na doske, ktorú som používal pre prácu s PIC24 (16-bitová platforma). Pozostáva z univerzálneho plošného spoja s rastrom 2,54mm a breakoutu, na ktorom je samotné PIC32 (resp. iné PIC s totožným pinoutom) a blokovacie kondenzátory. Vyzeralo to asi takto:



Je to jednoducho vyrobiteľné doma, s bežným postupom fotocestou, prípadne u nejakého lacného výrobcu DPS:
Samozrejme, dá sa použiť aj hotové DPS s PIC32, ale to už stráca kúzlo samodomo uchopenia problému.
K tomu som použil PicKit3 ako programátor/debugger, ktorý sa dá kúpiť napríklad v TME za cca 40EUR. PicKit3 poslúži pre rovnaký účel aj pre takmer všetky PIC MCU, od osembitových až po PIC32.
Ako vývojové prostredie som použil MPLAB v najnovšej verzii k tomu kompilátor MPLAB C32 Lite Mode , ktorý sa od plnej, platenej verzie líši tým, že neumožňuje zvoliť optimalizácie vyššie ako O2, inak veľkosť generovanej binárky nie je obmedzená. Toto obmedzenie pre hobby projekty nehrá nijakú úlohu a sám som s týmto nastavením už čo-to porobil. Pre komerčné použitie sa kompilátor zaplatí sám - výhodou je, že sa netreba učiť nič nové. A každý, kto sa elektronikou živí, si vie spočítať koľko času a peňazí ho môže stáť zápasenie s kompilátorom, ktorý bol inak "zadarmo".

4, Záver a tak

Samozrejme, možnosti PIC32 nekončia pri blikaní LEDkou. K dispozícii je toho viac, či už vnútorých periférií (6 USART-ov napríklad) alebo aj bohatej dokumentácie a zdrojových kódov (USB host/device/OTG stack, TCP/IP stack, ZigBee stack a iné), ale týmto príkladom s jednou LED som chcel poukázať na triviálny začiatok. K dispozícii sú kompletné vývojové nástroje, dostupný a lacný kremík ako aj množstvo podpory od výrobcu. Preniknúť do hĺbky PIC32, tak ako aj inej 32-bitovej platformy je ale zložitejšie, ak má byť využitá optimálne.