Fixed point

předpokládané znalosti: DSP engine

O fixed point si přečteme na straně 149 datasheetu k překladači MPLAB\_XC16\_C\_Compiler\_Users\_Guide.pdf adresář je něco jako C:\Program\_Files(x86)\Microchip\xc16\v1.36\docs

Podívejte se do toho datasheetu, nejde to kopírovat

V tabulce 8.3 máme všechny možné typy. Vidíme, že je to nachystáno na práci s DSP engine.

Dále máme od strany 168 – kapitola 9 – popsán Fixed-point aritmetic support. Račte si přečíst.

dále si pustíme program fixed01.c , slouží pro základní orientaci v problematice. Poznámka v datasheetu říká: -menable-fixed must be specified.

Vyřešil jsem to tak, že jsem to prostě dopsal

Cvaknout pravým tlačítkem na název projektu – dole Properties – XC16 – XC16 Global Option – dopsat do řádku aditional options

Při Build si pak zkontrolujte, že se vám to objeví ve volání gcc :

 "C:\Program Files (x86)\Microchip\xc16\v1.36\bin\xc16-gcc.exe" ../Vrh2018/diskP/mikroproc/33EV32/programovani/progr\_a\_vysv/fixed\_point/fixed01.c -o build/default/production/\_ext/1713490250/fixed01.o -c -mcpu=33EV32GM002 -MMD -MF "build/default/production/\_ext/1713490250/fixed01.o.d" -mno-eds-warn -g -omf=elf -DXPRJ\_default=default -legacy-libc  **-menable-fixed** -O0 -msmart-io=1 -Wall -msfr-warn=off

Vzorový file je fixed01.c

Tento program slouží k základní orientaci pro práci s typem fixed\_point , více od něj nečekejte.

Na první řádek kódu si dáme breakpoint, mačkáme F7, díváme se do variables. Čísla si zobrazíme také jako Binary, abychom viděli, co se doopravdy děje.

Neumí to míchat integer a Fract, proto je při násobení (-1) přetypování ( číslo -1 je samozřejmě integer ):

b = ( signed \_Fract )(-1) \* b ;

neumí to úplně dobře přetypovávat, hroutí se to u záporných čísel

Dále si přeložíme oba programy v adresáři ./engine01

Zde máme spolupráci asemblerovského programuc Cčkovým. Do source file addeneme oba zdrojové file a přeložíme. Pokusíme se udělat úvod k tomu, jak by se asi mohlo pracovat s DSP engine. potřebujeme k tomu znát látku v <http://ozeas.sdb.cz/panska/mikroproc/33EV32/programovani/asm_progr_a_vysv/asm_a_C/>

Soubor podprog.s slouží pouze pro nadefinování podprogramů, které se budou volat v programu hlavni.c . Pokud chceme definovat nějaké buňky, musíme k tomu použít sekci .bss . Překladač pak skombinuje tyto buňky s tím, co jsme nadefinovali v cčkovém file. Pokud použijete konkrétní adresu ( konkrétní číslo ) pro vaše proměnné, hrozí havarie, neboť překladač v části Cčkového file netuší, co jste udělali.

Zároveň se pokusíme zajistit, aby nám Cčkoá část programu „nehrabala“ do W registrů. To uděláme tak, že přepneme na druhou sadu – viz

<http://ozeas.sdb.cz/panska/mikroproc/33EV32/programovani/asm_progr_a_vysv/presuny/alternateWREG.s>

 Ale je také možné, že při volání podprogramu si to prostě registry W někam uloží a obejde se to bez přepínání sad registrů. Ostatně celý podprogram je celek, takže při jeho opuštění je jedno, jakou hodnotu mají registry a co jste s nimi dělali. **Určitě ale při provádění assemblerovského programu zakažte přerušení,** jinak hrozí, že zavolané přerušení změní obsah registrů ve vašem programu.

Podprogramu uvod a prvni slouží k tomu, abychom si mohli ujasnit, jak celá věc funguje. Mačkáme F7, vlezeme do podprogramů a díváme se, co se děje s registry. V uvod jsme samozřejmě do registrů W0 a W1 nacpali nějakou úvodní hodnotu.

V podprogramu uvod jsme zadali dvě hodnoty. Ty se uloží do registrů W0 a W1. Na začátku programu uvod přepínáme na druhou sadu registrů, to znamená, že předané čísla mizí v nenávratnu. Pomůžeme si tím, že hodnoty z registrů nejdříve uložíme do zásobníku (push) , a po přepnutí (CTXTSWP) je ze zásobníku uložíme zpět do registrů ( pop )

Je samozřejmě také možné využít nějakou buňku paměti, jak to vidíme ve funkci paty() , možná je to jednodušší než přepínat sady registrů.

Dále voláme podprogram první() . Vidíme, že po přepnutí jsou v registrech W0, W1 ta čísla, která jsme tam zanechali podprogramem uvod(). Uděláme s nimi nějakou operaci (inc) , přepneme zpět na . sadu registrů a konec. Při dalším volání první() je v registrech tatáž hodnota, kterou jsme v nich zanechali.

Dále zapisujeme hodnoty do pole polefract.

Pomocí programu spolem pak s tímto polem pracujeme. Do registru W uložíme adresu počátku polefract. Nezapomeneme na podtržítko, kterým zpřístupníme proměnnou z Cčkové části assemblerovské části ( 3.pád ) . Tedy ještě jednou předchozí větu jinak: tím zajistíme, že assemblerovská část bude vidět proměnnou z cčkové části .

V podprogramu je opět jednoduchá operace, zvýšení některých prvků pole o 1. Samozřejmě zde může být cokoli.

uvod02() je již skoro použitelná podprogram pro inicializaci DSP engine. Na začátku nastavíme formát signed fractional, dále zapneme aritmetiku se saturací. Nastavíme použití Program Space Visibility (PSV) a začátek tabulky konstant si uložíme do W8. Musí to být některý z registrů pro X-space, PSV je považováno za X-space. Vynulujeme akumulátor A a pomocí MAC prefetches nastavíme do W5 první hodnotu z tabulky, posunujeme ukazovátko W8.

Podprogram ctvrty() provádí operaci mac, opět jenom pro ukázku, zase si sem dejte cokoli. Dále vidíme, jak je možno vytáhnout vypočtené hodnoty z assembleru do Cčka. Nezapomeňte na deklaraci funkce na začátku, jinak nebude Cčko vědět, za jaké číslo má považovat vrácenou hodnotu a jak s ním pracovat. Funkci ctvrty() voláme několikrát, abyste si mohli vychutnat uvedený mechanismus.

Pomocí programů podobným programům uvod02 a ctvrty se již dá udělat výpočet sinusovky pomocí DSP algoritmu. Program uvod02 nacpe do registrů konstanty pro výpočet sinusovky , něco jako ctvrty bude při každém zavolání blejt výstupní hodnoty. V algoritmu je násobení dvěma, to uděláme prostě tak, že mac uděláme dvakrát. Násobení (-1) je hračkou, uložíme si do nějakého registru 0x8000 . Nebo místo mac můžete použít msc, a tak dále a tak dále. Výstupní hodnotu můžete rovnou poslat na SPI po malé úpravě – z fixed uděláme signed int jednoduše tak, že bity několikrát posuneme pomocí ASR ( aritmetical shift right ). Pak k tomu přičteme polovinu rozsahu, zamaskujeme a máme to.

Dále je funkce paty() , která dále ještě umožňuje nacpat do assemblerovské části nějaké číslo z Cčkové části. Číslo, které předáváme do podprogramu, se objeví v registru W0. Zde máme jiný způsob, jak přelstít přepínání registrů. Použijeme k tomu buňku paměti, do které uložíme obsah W0, a po přepnutí na první sadu registrů toto číslo uložíme z buňky do registru W4. Tady nemáme příliš na výběr, protože chceme násobit a sčítat ( mac ). Výsledek opět zpřístupníme Cčkovému programu, připomínám, že hodnota při návratu musí být ve W0 a nikde jinde. Do VARIABLES si samozřejmě dáme proměnnou a a budeme se dívat na její hodnotu.

Funkce modulo 01 předvádí další famózní možnosti DSP engie. Jedná se o „modulo addressing“ , datasheet kapitola 4.5 , strana 76. Při inkrementování registrů u mac prefetches často potřebujeme, aby se hodnota registru po překročení nějaké hranice vrátila na začátek. A to je právě ono ! Začátek nacpeme do regisru XMODSRT ( pokud pracujeme s X-space ), konec do registru XMODEND ( opět u X-space ). V registru MODCON nastavíme registr W, který má být takto adresován ( MODCON je v datasheetu na straně 42, tab4-1. Pořádný výklad k tomu chybí. Bity 15 a 14 zapínají modulo addressing pro X-space nebo Y-space, bity 7-4 definují registr W pro Y space, pokud zadáte 0xf , je to disabled. Obdobně bity 4-0 definují W registr pro X space. ) Dejte si do VARIABLES W8 a dívejte se, jakých hodnot nabývá při opakované instrukci mac. Funguje to samozřejmě i dolů – instrukce pod tím. Na vaší laskavé úvaze ponechávám, zda hodnota v registru W oproti MODSRT a MODEND je větší , menší, větší nebo rovno, ……..

Poslední lahůdka nás čeká u funkce uvodmY. Je to nastavení modulo addressing pro Y space. Nastavujeme tam W10 na hodnotu nasepole, což je místo v Yspace paměti ( podívejte se, co se uložilo do W10 , a porovnejte to s rozsahem pro Y space ). Na začátku programu se podívejte, jak donutit překladač, aby umístil nasepole do Y space.

Podprogram pokrY pak ukládá čísla do našeho bufferu, a to pořád dokolečka. Dívejte se na hodnotu W10 , vidíme, jak se cyklí. Předpokládám, že je jasné, že tohle se dá využít pro konstrukci posouvátka pro filtr FIR. Vstupní vzorek ukládáme dál a dál do pole, a modulo addressing zajistí, aby na konci pole ukazovátko W10 samo skočilo na začátek. Hračkou by pak mělo být udělat vynásobení a posčítání konstant z pole tabulka a vzorků, uložených v nasepole. Rychlý FIR je tedy hračkou.