AD Blackfin

Hardverrel kapcsolatos kérdések

Mi az egyik legjelentősebb előnye a Blackfin DSP processzor modellnek? Milyen típusú működést/használati módokat támogat?

(És ehhez kapcsolódó kisebb kérdések pl. vezérlési feladatok támogatása, bájt címezhetőség, MMU jelentősége, stb. Ugyanakkor a DSP használat támogatása MAC egység(ek), adatstruktúra felépítése, operandusok, számábrázolás.)

Az AD Blackfin processzorcsalád legnagyobb előnye, hogy kombinálja a nagy teljesítményű DSP processzorok képességeit a hagyományos mikrovezérlők széles periféria, szenzor és I/O kínálatával. A kombináció előnye, hogy egyetlen csip alkalmazása helytakarékos a nyomtatott áramkör tervezése során, valamint a vezérlési és jelfeldolgozási funkciók egységesen, egyetlen fejlesztőkörnyezetben, egyetlen toolchain alkalmazásával fejleszthetők.

A mikrokontrollerek "világából hozott" funkciók:

- L1 memóriaterület a stack és a heap számára, melyek kezeléséhez dedikált hardver pointerek állnak rendelkezésre valamint Memory Management Unit a biztonságos memóriakezeléshez,
- bájtos címzés,
- bit-szintű adatmanipulációk (shift-elés, logikai műveletek) támogatása.

A jelfeldolgozó processzoroktól "örökölt" funkciók:

- gyors és rugalmas, az FFT és konvolúció elvégzésére szabott aritmetikai feldolgozóegységek,
- kiterjesztett pontosság és dinamika ("szélesebb" számábrázolás, műveletvégzők és akkumulátor),
- hatékony címaritmetika, sequencer ("programfolyam-vezérlő") és I/O feldolgozás,
- szabad adatáramlás a feldolgozóegységek között (memória közbeiktatása nélkül).

Mit jelent a Harvard architektúra?

A Harvard-architektúra az program- és adatmemória fizikai elválasztását jelenti. Előnye, hogy az utasítás- és operandusbeolvasás egymástól független lehet, ezzel támogatva a DSP processzorokban jellemző pipeline utasításfeldolgozást. További előnye, hogy adatcímzés útján nem kerülhetünk programkód-területre, így a kódot véletlenül vagy szándékosan elvileg nem írhatjuk felül. Hátránya a nagyobb komplexitás a Neumann-architektúrához képest.

A Blackfin Core felépítése, fontosabb egységei, műveletvégzők és a fontosabb párhuzamos használati esetek

A Blackfin Core főbb egységei:

- Aritmetikai egység: több különböző műveletvégző együttese, mely támogatja a SIMD utasításokat. Tartalmaz két 16 bites hardver szorzót (valójában teljes MAC-egységek), két 40 bites és négy darab nyolc bites aritmetikai-logikai egységet, utóbbiak kifejezetten videó-jelfeldolgozáshoz előnyösek. Ezeken kívül tartalmaz még egy barrel-shiftert, ami a shift-eléseket és egyéb bitműveletekre hivatott. A felsorolt műveletvégzőket 2 db 40 bites akkumulátor és 8x32 bites regisztertömb egészíti, mely Load/Store utasításokkal tölthető és menthető.
- Címképző egység (adatcímzés):
  - két darab Data Address Generator egység, melyek függetlenül képesek 32 bites címeket generálni,
  - 6 darab általános célú regiszter, melyek 8-, 16- vagy 32 bites memóriacímek beolvasására és tárolására alkalmasak,
  - 4 regisztertömb a speciális, DSP-kre jellemző kettős, 16- vagy 32 bites memóriaolvasás támogatására,
  - dedikált stack- és frame pointer-ek a stack és a heap biztonságos és gyors kezeléséhez.
- Sequencer (programcímzés): a programkód zavartalan futtatását elősegítő eszköz. Generálja a következőnek beolvasandó utasítás memóriacímét, a különböző szélességű utasításokat igazítja (alignment), kezeli a kivételeket és megszakításokat, a feltételes ugróutasításokat hatékonyabbá teszi a műveletvégzők belső állapotának (státuszbitjeinek) figyelésével és hardveres huroktámogatással.

Mit jelent az "Algebrai assembler" szintaxis, miben különbözik ez pl. a RISC-V (vagy más) CPU "hagyományos assembler" szintaxisától? Hogyan támogatja a különböző speciális adatformátumok pl. videó feldolgozását?

Az algebrai assembler szintaxis a hagyományos assemblerrel szemben nem utasításneveket és operandusfelsorolást tartalmaz (pl. ADD R1, R2, R3), hanem "egyenletszerű" (pl. R3 = R1 + R2) a könnyebb olvashatóságot és utasításon belüli párhuzamosítást támogatandó (pl. R3 = R1 + | - R2, R6 = R4 + | + R5). Speciális formátumok, pl. videó-jelfeldolgozás támogatását ez oly módon támogatja, hogy a párhuzamosan, több pixelre végzett aritmetikai műveletek (pl. négy darab 8 bites művelet) egyetlen utasításba foglalható.

Mi a szerepe a címaritmetikai egységnek? Milyen jellemző támogatást ad?

A címaritmetikai egység a nagy teljesítményű műveletvégző folyamatos és hatékony "munkában tartására" hivatott, ennek érdekében pedig az alábbi funkciókat támogatja:

- Kettős címzés: a jelfeldolgozó műveletekhez kettő vagy több operandus szükségeltetik, ezért egyszerre két, egymástól független címen lévő adat beolvasását is támogatja az egység.
- Hagyományos és utólagos címzés: utóbbi esetben a memóriaolvasást követően is módosítható a célcím és újraolvasható az adat, ez például ugró utasítások esetén hasznos a téves feltételezés gyors javítására.
- Cirkuláris buffer overhead nélküli címzése.
- Minden címzés regiszter indirekt: dedikált stack-, frame- és egyéb mutató regiszterekben tárolt címek használata.
- Bájtos szervezésű címzés: támogatja a különböző hosszúságú (8-, 16- vagy 32 bites) adatok olvasását (32 bites) szóillesztéssel.

Mi az a sequencer? Hányszoros egymásba ágyazott ciklusok végrehajtását támogatja ciklusveszteség nélkül?

A programkód zavartalan futtatását elősegítő eszköz. Generálja a következőnek beolvasandó utasítás memóriacímét, a különböző szélességű utasításokat igazítja (word alignment), kezeli a kivételeket és megszakításokat, a feltételes ugróutasításokat hatékonyabbá teszi a műveletvégzők belső állapotának (státuszbitjeinek) figyelésével és hardveres huroktámogatással. Hardveresen támogatott módon, overhead-mentesen két ciklus ágyazható egymásba, ekkor az utasítások olyan sebességgel futnak, mint lineáris programszervezés esetén.

Nagyjából milyen mélységű az adatfeldolgozó pipe-line?

A Blackfin adatfeldolgozó pipeline-ja 10 lépés mély:

- három lépésből álló utasítás-beolvasás: utasításcímzés és utasítás-cache vizsgálat, utasításra várakozás, utasításolvasás és igazítás (alignment),
- egylépéses utasítás dekódolás,
- egylépéses adatcím-számítás,
- kétlépéses adatbeolvasás: adatcímzés és adat-cache vizsgálat, regiszterolvasás,
- kétlépéses utasítás-végrehajtás: adatbeolvasás és hosszú műveletek (szorzás, videó feldolgozás) megkezdése, egyszerűbb műveletek végrehajtása vagy a megkezdettek befejezése,
- eredmények visszaírása egyetlen lépésben.

Mi a különbség a megszakítások és a kivételek között? Mit jelent ez a kiszolgálásban?

Megszakítások és kezelésük: a megszakításokat hardver egységek (ritkábban szoftver), jellemzően DMA, perifériák vagy I/O egységek generálnak normál működésük során a processzormag aszinkron értesítése céljából, kezelésüket a System Interrupt Controller végzi. A SIC lehetővé teszi a perifériák megszakításkéréseinek csoportokba és prioritásszintekbe történő szervezését.

Kivételek és kezelésük: hibás szoftveres működés vagy hibafeltétel teljesülése esetén generálódnak, feldolgozásukat a Core Event Controller végzi. A CEC 16 prioritásszintet támogat, melyből az alsó 9 használható általános célú periféria megszakítások kezelésére.

Mi jellemző az utasításformátumra, milyen elvárásokat támaszt ez a felhasználóval / programozóval szemben? Miért van értelme a változó hosszúságú utasításkódolásnak?

A Blackfin utasításformátumára a változó hosszúság jellemző. Három különböző utasításhosszt támogat: 16-, 32- és összetett 64 biteseket. Utóbbi azért érdemli ki az összetett jelzőt, mert valójában több utasítást tartalmaz: bizonyos 32 vagy 16 bites utasítások párhuzamosan végrehajthatók más, 32 vagy 16 bites utasításokkal, ezeket egyetlen 64 bites utasításként egy lépésben be lehet olvasni (lsd. következő kérdés).

Az efféle utasítás szervezés elsődleges célja a magas kódsűrűség elérése és a jelfeldolgozó egységek hatékony és gyors információval való ellátása. A programozó ahhoz, hogy ezt a kifinomult és teljesítménycentrikus utasításformátum adta lehetőségeket kiaknázza célszerű natív assemblyben kódolnia, továbbá előre alaposan megfontolnia, hogy az implementálandó algoritmus hogyan írható le a leghatékonyabban a speciális utasítások felhasználásával.

Értelmezzen egy adott utasítást, írja le,hogy mi a funkciója az egyes részkifejezéseknek!

Példák:

R6 = R2 + R3

A 32 bites R2 és R3 regiszterek összeadása, majd az eredmény elhelyezése a szintén 32 bites R6 regiszterben.

R3 = R1 - R2, R4 = R1 + R2

Két darab, párhuzamosan végrehajtott, 32 bites művelet, azonos operandusokkal, de különböző célregiszterekkel.

R6.H = R2.H + R3.L (s)

A 32 bites R2 regiszter felső (H) 16 bitjének és a szintén 32 bites R3 regiszter alsó (L) 16 bitjének összeadása, majd az eredmény mentése a z R6 regiszter felső 16 bitjébe. Az "(s)" az utasítás végén "single" 16-bites utasításra utal.

R6 = R2 + | - R3

Dupla 16 bites utasítás, mely egy összeadást és egy kivonást eszközöl az R2 és R3 regiszterek felső és alsó 16 bitjén (ebben a sorrendben). A felső 16 bitek összeadásának eredménye R6 felső 16 bitjébe kerül, az alsó 16 bitek kivonásának eredménye R6 alsó 16 bitjébe kerül. Az utasítás hatása egyenértékű két darab "single" 16 bites utasítással: R6.H = R2.H + R3.H illetve R6.L = R2.L - R3.L utasításokkal.

R3 = R0 + | - R1, R2 = R0 + | + R1

Négyszeres (quad) 16 bites utasítás, azonos operandusokkal, de különböző célregiszterekkel, ahol az egyes operációk a 32 bites regiszterek felső és alsó 16 bitjén értendőek.

A1 -= R2.H * R3.H, A0 += R2.L * R3.L

Két darab, párhuzamosan elvégezhető MAC művelet (a szorzások csillaggal, az összeadás/kivonás += és -= operátorokkal jelzettek). A műveletek ebben az esetben is ugyanazon 32 bites regisztereket, de külön felső és alsó 16 bitjüket használják, az eredmények a 40 bites regiszterekbe kerülnek.

A1+=R0.L*R1.H, A0+=R0.L*R1.L || R0 = [I0++] || R1 = [I1++]

Összetett 64 bites utasítás. A 64 bitbe egyként belefoglalt három utasítást (egy 32 bites és két 16 bites) a "||" karakterek választják el egymástól. Az első utasítás a korábban megismert, kettős MAC utasítás, a második az I0 indexregiszter inkrementálása után kapott memóriacímről való adatbeolvasást jelenti az R0 regiszterbe, hasonló módon a harmadik utasításhoz.

Mi a különbség a REG, L1, L2 és L3 memóriák között a használhatóság szempontjából?

- A regiszterek (REG) a műveletvégzők közvetlen közelségében lévő, emiatt gyors elérésű memóriák. Hátrányuk, hogy kevés van belőlük.
- Az L1 (utasítás- és adat-) memóriák a processzormaghoz legközelebb elhelyezkedő cache memóriák, elérésük valamivel lassabb (egy ciklus idő), mint a regisztereké, cserébe méretük nagyobb és az adatcímző egység még mindig közvetlenül kereshet bennük a gyorsítás céljából korábban behozott adatok gyorsított beolvasása érdekében.
- Az L2 opcionális "nagy cache" memória, mely az L1-nél lassabb (több órajel-ciklus szükséges az olvasásához), de az L3-nál gyorsabb, illetve az L1-nél nagyobb, de az L3-nál kisebb kapacitású.
- Az L3 memória már a csipen kívül lévő (pl. külső SDRAM) memória, melynek így a legnagyobb a késleltetése, cserébe a kapacitása is meghaladja a korábban említetteket.

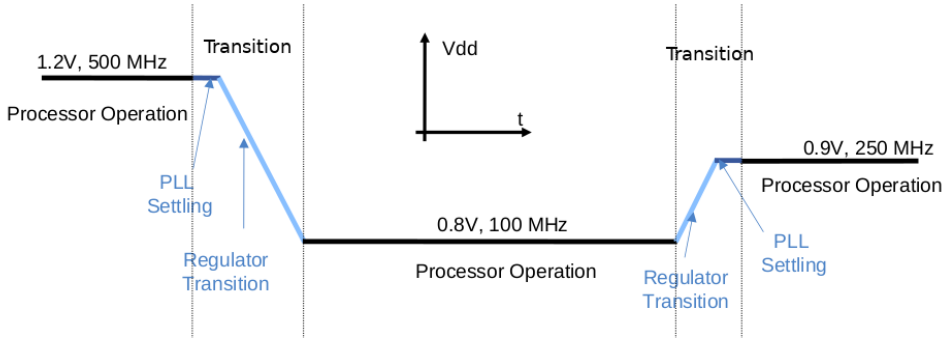
Milyen teljesítmény optimalizálási módokat támogat a Blackfin CPU?

Alacsony aktív fogyasztási mód: a processzor aktivitása közben képes a tápfeszültségének és órajel-frekvenciájának dinamikus megváltoztatására, valamint a nem használt perifériák automatikus kikapcsolására.

Alacsony passzív vagy stand-by mód: a processzor inaktivitása alatt öt különböző állapotban működhet: 1) továbbra is teljes fogyasztással, 2) készenléti állapotban várakozva, 3) alvó állapotba vagy 4) mély alvó állapotba kapcsolva, 5) teljes hibernálásban. Ezeket a passzív állapotokat támogatandó az RTC (Real-Time Clock) modulnak és a felébresztő logikának saját tápellátása van. A passzív módok mindegyike hardveres vagy szoftveres reset-től függetlenül megmarad.

Mit jelent a dinamikus teljesítmény menedzsment?

A dinamikus teljesítmény menedzsment a Blackfin processzor azon képessége, hogy aktív állapotában, program futtatás közben képes tápfeszültségének és órajel-frekvenciájának megváltoztatására a fogyasztás csökkentése érdekében. Ehhez hardveres támogatást jelent, hogy a belső PLL az órajel-bemeneten kapott jel frekvenciáját képes akár 64-szeresére is csökkenteni, miközben a magfeszültség a beállított frekvenciához hangolódik (magasabb frekvenciához nagyobb feszültség szükséges). A két paraméter együttes változtatása hatékonyabb fogyasztáscsökkentés eredményez, mintha csak az órajel-frekvenciát csökkentenénk.



1. ábra. Példa dinamikus fogyasztás menedzsmentre

Szoftverrel kapcsolatos kérdések

Miért szoktunk közvetlen alacsony szintű assembler programozást használni a DSP processzoroknál? Mi a cél?

A cél a DSP processzorokban rendelkezésre álló speciális hardver egységek, teljesítménycentrikus címzési módok és utasítás architektúra minél jobb kihasználásra, ezért a magas szintű absztrakciót feladva közvetlenül a "hardverhez szólunk". A C elhagyásának további oka, hogy a jelfeldolgozó processzorok architektúrái gyorsabban fejlődnek, mint az a szabványos C nyelv követni tudná.

Miért szeretnénk itt is C szintű programozást? Mi a probléma ezzel, miért nem magától értetődő ez?

Bár a magas szintű C kódok gyorsan fejleszthetők és hordozhatók, a DSP processzorok speciális képességei általában rosszul képződnek le a fordítás során. Ahhoz, hogy C kódból igazán hatékony gépi kód forduljon, a fordítónak figyelembe kell vennie az adott architektúrában rejlő optimalizációs lehetőségeket, mint amilyen a Blackfin esetében az utasítások párhuzamosítása.

Mit jelent a hardver tulajdonságok figyelembe vétele?

Azt jelenti, hogy a C fordító "tudatában van" a cél architektúra különlegességeivel és speciális tulajdonságaival, ki tudja használni a többszörözött műveletvégzőkből, összetett utasításformátumokból adódó optimalizációs lehetőségeket, továbbá tisztában van az utasításfeldolgozás pipeline struktúrájával és azzal, hogy ennek támogatására mekkora hardveres segítség áll már rendelkezésre, milyen lépéseket kell neki tennie a pipeline "telítetten tartása" érdekében.

Milyen lehetőségei vannak a programozónak és milyen lehetőségei vannak a fordítóprogramnak?

C nyelvű programozás esetén a programozónak mindössze arra van lehetősége, hogy a programkódot minél inkább a rendelkezésre álló műveletvégzőket figyelembe véve implementálja az algoritmust: ne alkalmazzon olyan műveleteket, melyeket natív módon nem támogat a processzor, ugyanakkor alkalmazzon minden rendelkezésre álló lehetőséget, pl. hardveres szorzókat, MAC-egységeket.

A fordítónak lehet ennél több és kevesebb lehetősége is attól függően, mennyire "van tudatában a hardvernek". Az architektúrát figyelembe vevő fordító észlelhet pl. adat hazárdot a fordítás eredményéül adódó gépi kódban és elkerülése érdekében felcserélhet utasításokat. A C kód értelmezésekor optimalizációs lehetőségeket aknázhat ki, amik az utasításformátumból vagy a rendelkezésre álló hardvertámogatásból adódik.

Assembly nyelvű programozás esetén a programozó, megfelelő mennyiségű szaktudást, előkészületet és odafigyelést feltételezve, teljes ura lehet a program-végrehajtásnak és komplex megfontolásokat tehet az utasításoknak illetve sorrendjüknek megválasztása során.

Milyen eszköz a profiler és mire tudjuk használni? Mit mutat meg forráskód szinten és mit mutat meg utasítás-végrehajtás szinten?

A profiler a programkód futását elemző eszköz, segítségével feltárhatók a program kritikus szakaszai és gyengeségei. Forráskód szinten megmutatja, mely sorok vagy kódrészletek futnak a legtovább vagy a leggyakrabban, utasítás-végrehajtási szinten pedig a, redundáns gépi utasításokra, a gépi kódból adódó hazárdok vagy egyéb hatékonyságcsökkentő jelenségekre hívhatja fel a figyelmet. A profiler által szolgáltatott információk segítségével kézi optimalizálás alá vehetjük a program kritikus szakaszait, miközben a nem kritikus vagy elégségesen hatékony részeket magasabb szintű leírásban meghagyhatjuk, ezzel hatékonyabbá téve a kódot a hordozhatóság és olvashatóság minél nagyobb mértékű megtartása mellett.

Mire érdemes odafigyelni, ha hatékony végrehajtást szeretnénk? Általános értelemben mikor keletkezik pipe-line fennakadás?

A hatékony utasítás-végrehajtás záloga a pipeline "telítve tartása". A pipeline feldolgozás fennakadhat különféle típusú hazárdok esetén: strukturális hazárd esetén foglalt erőforrásokra való várakozás miatt, adat hazárd esetén még ki nem értékelt vagy memóriába vissza nem írt, így rendelkezésre nem álló bemeneti adatokra való várakozás esetén vagy rosszul előrejelzett, feltételes ugróutasítás kiértékelését követő "korrigálás" alkalmával.

Mi a tartalma a fast-fp és az ieee-fp táblázatban közölt értékeknek?

Mivel a DSP processzorok jellemzően fixpontos számábrázolásra vannak "kihegyezve", ezért a lebegőpontos műveleteket csak emulálni képesek. A fast-fp és ieee-fp táblázatok egyaránt arra vonatkozó adatokat tartalmaznak, hogy az egyes lebegőpontos (floating point) műveletek emulációja az adott módszerrel ("fast" vagy "ieee") hány gépi utasítást vesz igénybe. A fast-fp jellemzően kétszer gyorsabb / fele annyi gépi utasítással működik.

Milyen értelmes engedmények/egyszerűsítések teszik lehetővé ezt a nyereséget?

A fast-fp jellemzően kétszer gyorsabb / fele annyi gépi utasítással működik, mert az IEEE szabványos emulációval szemben "lazábban kezeli" a NaN (Not A Number) értékeket, melyek például nullával való osztás esetén adódnak.