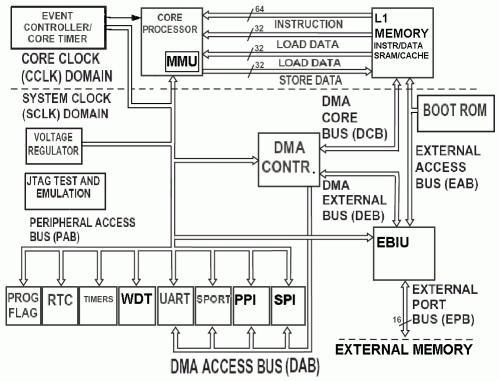
Készítette: Nagy István buenos@freemail.hu

ADSP-BF533/532 - Blackfin processzor - SEGÉDLET

1. Hardver

Általános :

❖ **Blokkdiagram:**



❖ **Címkezelés:** Egy közös 4GB-os címterületen van minden: MMR-ek, L1 SRAM (cash nem külön terület), külső memóriák, perifériák. Nincs külön I/O-terület. Ezeket bázisrelatívan címezzük, a P0…P5 regiszterekkel. Látsd: ALU>cím aritmetika.

❖ **Üzemmódok:** User / Supervisor / Debug. Az MMR-ekhez, és (éppen) védett memóriaterületekhez csak az utóbbi kettőben lehet hozzáférni, és bizonyos utasítások is csak itt érhetők el. Ha mégis user módban próbáljuk, akkor kivételt generál az Event Controller. Supervisor mód: reset után, és a megszakítási/kivétel rutinokon belül. Reset után célszerű beállítani a user-módot (az első IT után úgyis beállna): [P1.L=Exit_addr; P1.H=Exit_addr; RETI=P1; RTI;]

A CPU:

❖ **Belső regiszterek:** R0…R7, Acc0, Acc1. Látsd:ALU.

❖ **Címzés:** Látsd: ALU>cím aritmetika, és Assembler.

❖ **Program sequencer:** Ez felelős az utasítás betöltésért, pipeline kezelésért, ugrások/IT/kivétel…kezeléséért (ALU). SEQSTAT regiszter információkat nyújt a legutóbbi reset és kivétel –eseményekről. Feltételes ugrásokat a cc (feltételkód flag) alapján lehet csak.

Hardver-programhurkok kezelésére külön 32bites regiszterek vannak: LC0, LC1 (ciklusszámlálók), LT0, LT1 (loop top address), LB0, LB1 (loop bottom address). Ezeket a megfelelő ciklus-utasítások kezelik. Működése: feltöltjük a regiszterket (LC,LB, LT) majd ha a programvégrehajtás elér az LT1-ben megadott pmem címig, akkor indul az LC1 hurokszámláló delkrementálása minden ciklusban. Amikor elér az LB1-címig, akkor visszaugrik LT1 címre, LC1-et dekrementálja. Ha LC1=0, nem ugrik már vissza, hanem folyamatosan halad tovább. Ha a ciklus kevesebb mint 4 utas, akkor a pipeline-t nem üríti, begyorsul.

Visszatérés szubrutin/IT/kivét…-ből: Ezek mindegyikéhez külön visszatérési cím-regiszter van, amit automatikusan kezel. Szubrutin: RETS, IT: RETI, kivétel: RETX, NMI: RETN. Tehát többszintű szubrutinhívásokkor manuálisan kell kezelni a RETS tartalmát és stack-be menteni !!! (hívás előtt: [--SP]=RETS; Hívás+visszatérés után RETS=[SP++];)

❖ **A Stack Pointer:** 2db van, egy az user módnak, egy a supervisor módnak. Mindkettőt SP néven érhetjük el, csak egyik módban az egyik, másikban a másik az amit SP-nek hívunk.

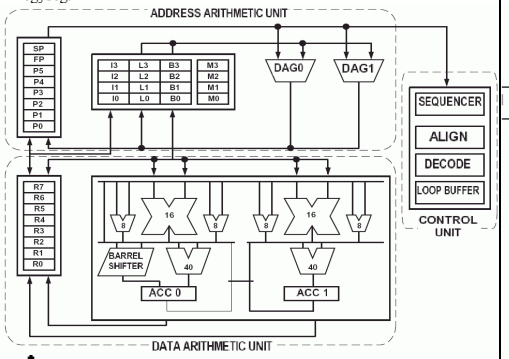
❖ **A pipeline:** 10 fázisú. Nincs pipeline-ütközés, mert a proci azt magától kezeli, beiktat várakozásokat, ha egy utasítás egy még nem létező eredményre vár. (kikapcsolható).

❖ **Adat formátumok:** A számításokat többnyire 2-es komplemens kóddal végzi (Q15-ös tört számok.), de lehetnek integer (előjeles/nem előjeles) vagy bináris string (logikaihoz) operandusú műveletek is. Adat lehet 32(word)/16(half word)/8(byte) bites. Számok kezelésekor lehet előjelkiterjesztést vagy nulla-kiterjesztést kérni, ezeket a programban (X) (Z) –utánírással kell kérni. Little-Endian.

❖ **Reset:** 4 féle reset lehetséges: hardver (RESET láb által), system szoftver (ha #B111-et írunk az SWRST regiszterbe. Csak a perifériákat érunti, kivéve RTC), csak core szoftver reset (RAISE1; utasítással), és watchdog reset. Hardver reset után a proci boot módba megy át.

Az ALU:

A CPU része. 32/16/8 bites adatok feldolgozására, akár párhuzamosan több adattal is. 2 fő rész: Adat aritmetikai egység, és Cím aritmetikai egység.



❖ **Adat aritmetika:** 40bites akkumulálás, de 16 bites adatok. Ehhez tartozik a Data-register-file (R7…R0, ezek 32 bitesek de 16 bites műveletekkor 16x16bit. Ekkor pl. R0 helyett R0.L és R0.H van.), és a 2db akkumulátor (ACC0, ACC1, 40bit. Darabjai: A0.L A0.H és A0.X ami a felső guard) van, mert 2db ALU van. 16 bites vagy 8 bites adatokon dolgozik, egyszerre 2 (2x16, 2x32) / 4(4x8, 4x16) adattal. A nagyobbakhoz párhuzamos utasítást kell kérni az asm-ben. 2db MAC egység van, ezek kiválasztása automatikus. Műveleteket nem lehet közvetlenül a memóriával végezni, csak az ALU belső regisztereivel. Őket pedíg a Load/Store utasítások töltik/ürítik.

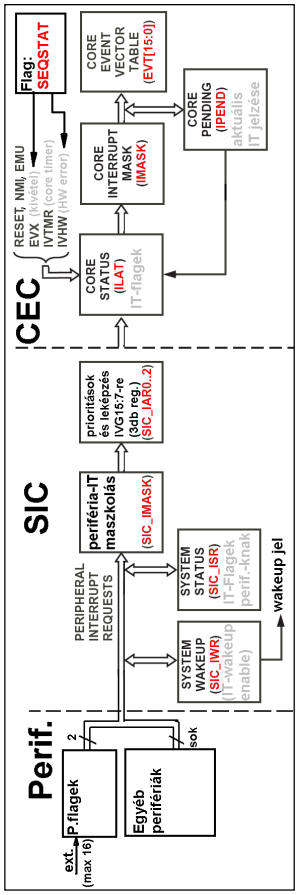
❖ **Cím aritmetika:** A memóriák, és nem-core regiszterek elérése mindig ezen keresztül történik. Jellemző a bázisrelatív címzés (adatmem), amikor a memóriacím=P[i]+offszet. P[i] az a 8db pointer regiszter (32bit) egyike (általános: P5…P0 6db). A program utasítások is így működnek, csak PC-re relatívan. Az assembler a címkéket offszetre váltja. A **DAG** címgenerátorok, és a hozzájuk tartozó Ix, Lx, Bx, és Mx regiszterek (mindből 4db: 3…0) a cirkuláris buffereket szolgálják ki, egyszerre 2db-ot, a 2db címgenerátornak köszönhetően (DAG). Címet lehet post/pre módosítani autom. Ennek a része a stsck pointer is.

Cirkuláris bufferek: 4db mehet egyszerre, az Ix (kimenő cím), Lx (bufferhossz), Bx (buffer bázisím), Mx (lépésköz) regiszterek segítségével. Ezek sorszámuk alapján összetartoznak !!! Buffer készítés: Feltöltjük pl B1, L1-et, (M1 nem is kell, számmal helyettesíthető,) Majd címzés: R3=[I1++]; vagy ha M-et is használjuk: R2=[I1++M1]; Vagy konstanssal: R4=[I1++16]; I1 magától generálódik, de nullázhatjuk is, nem nullával, hanem B1-et átmásoljuk I1=B1; .

❖ **Státusz:** Aritmetikai műveletek (kivéve P regiszterekkel) státusza az ASTAT regiszterben kérdezhetők le. Ugrási feltételként csak egyet lehet felhasználni.

A Megszakítások (események):

Esemény fajták: Reset, NMI, megszakítások, kivételek (hibás működéskor, pl védett memória címzése), emuláció. Ezeket az Event controller kezeli, aminek két része van: **Core Event Controller** (CEC, 16 általános interrupt: IVG0 … IVG15), és a **System Interrupt Controller** (SIC a periféria IT-ket csoportosítja 9db kimenetté a CEC számára). Hogy melyik perif. IT-t hova map-peli, az a HW reference 4-24 -es oldalán van.



Vektortábla: 16db IT -vektor van (IVG0 … IVG15-hez), ezek a core event controller-ben az EVT15…0 regiszterekben, tehát nem a sima memóriában. Nem bázis +fix offszet, hanem külön címek.

Kivételek: illegális memória-elérések, illegális utasításhívások. Hogy melyik forrás okozta: SEQSTAT regiszterben.

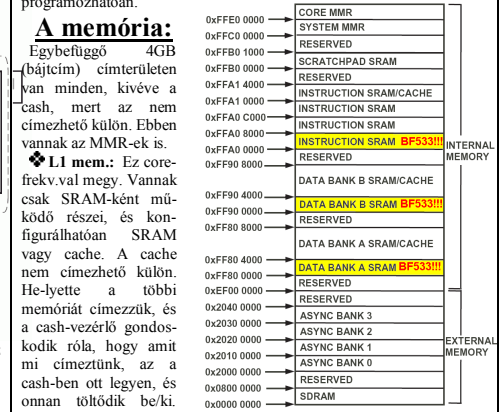
HW error: pl busz ellenőrzi a paritást, azaz ha adatot tesz ki, azt vissza olvassa, és ha nem jó (rövidzár vagy ütközés van), hiba. Más: timeout. Forrás szintén a SEQSTAT-ban.

IT rutin: csak ezekben (IT, kivétel, NMI…) van Supervisor mód. Közben az IT-k tiltva vannak. A végén RETI; utasítás.

Kívülről jövő megszakítások: NMI láb (aktív magas!), és a programozható Flag interfész bármely bemenetéről (max 16db), programozhatóan.

A memória:

Egybefüggő 4GB (bájtcím) címterületen van minden, kivéve a cash, mert az nem címezhető külön. Ebben vannak az MMR-ek is.



❖ **L1 mem.:** Ez core-frekv.val megy. Vannak csak SRAM-ként működő részei, és konfigurálhatóan SRAM vagy cache. A cache nem címezhető külön. He-lyette a többi memóriát címezzük, és a cash-vezérlő gondoskodik róla, hogy amit mi címeztünk, az a cash-ben ott legyen, és onnan töltődik be/ki. Külön van az utasításoknak és az adatmemóriának. Konfigurálni kell! Ezt az IMEM_CONTROL és a DMEM_CONTROL regiszterekkel lehet megtenni. Reset után cash disabled. Bankokra és sub-bankokra van osztva: 16k/4k.

Utasítás cache: Ha cache: akkor sorokra (32Bájt) van osztva. Minden sor tartalmaz egy TAG-et, ami megmondja honnan származik, érvényes adat van-e benne, és hogy mikor volt használva (régen/nem). Cache Hit: amikor egy olvasáskor az adatot már bent találjuk a cache-ben. Cache Miss: amikor nem. Ekkor egy teljes sort tölt be. Először a kért szót, utána a többit. A procinak nem kell várnia az első bájt cache-be másolására, ő egyből megkapja. Ezt a TAG alapján asszociatív címzéssel teszi cache vezérlő. Lock-olás: Az utasítás cache 4-way-re van osztva. Ezekről egyenként megmondható, hogy üríthető-e, vagy rezidens legyen. Ürítés: IFLUSH utasítással. (IFLUSH[P5]; ekkor P5 által mutatott címeknek megfelelő sorokat vegye ki.)

Adat cache: 2db címgenerátora van a DSPnek, mindegyiknek 2db portja. Ezeket hozzá kell rendelni a cache-bankokhoz. Az adat cache-nél fordul csak elő, hogy a betöltött adat megváltozott (piszkos lett). Ekkor kimásolódik automatikusan. Ehhez használ egy FIFO-t is. Ehhez 2 mód van: Write through: azonnal kiír. Write back: csak akkor ír ki, ha az adott sort le kell cserélni. Kezelő utasítások: PREFETCH; FLUSH;

❖ **MMU:** memóriavédelem. Ha olyan memóriacímre hivat-kozunk, ami nem létező, akkor kivétel történik.

CPLB: cache vezérlő logika+védelem. Van egy táblázat (MMR-ek) 16db bejegyzés, 2db 32bites regiszter az adatmemória eléréshez (DCPLB_ADDRn, és DCPLB_DATAn) és külön a programmemória eléréshez (ICPLB_ADDRn, és ICPLB_DATAn). Ezek részletesen: **Az adatregiszter:** beállítások egy db memória lapra, és a lapméret megadása (1k/4k/1M/4MB), ezenkívül hozzáférési jogok, cache flagek, írható/csak olvasható, cache-elhető/nem (pl periféria-reg. -nem!!!)... **Címregiszter:** Az adott lap hol kezdődik. **A védelem működése:** ha adatot/utasítást címzünk, akkor egy asszociatív kereső HW megkeresi hogy az melyik megadott lapon van, és ott mi érvényes. Ha nem talál ilyen CPLB bejegyzést, ami ilyen címet tartalmazó tartományt ad meg, akkor kivétel keletkezik. Ha kevés ez a 2x16 bejegyzés, akkor a memóriában tárolni kell egy PageDescriptorTable-t, aminek (pl az egyes taszkokhoz tartozó) a mostanában használni kívánt részeit beásoljuk a CPLB regiszterekbe. Vagy a kivételkezelő rutin oldja meg a cserét. Hibázó lap kinyomozása: DCPLB_STATUS és a DCPLB_STATUS. Ezek csak a kivételkez.rutinban elérhetők.

Engedélyezni/tiltani lehet. Ha engedélyezzük a cache működést, akkor engedélyeznünk kell a CPLB-t is, az adott területen. Ezt mind a cache vezérlő regiszterekben kell. (IMEM_CONTROL , DMEM_CONTROL)

A DMA:

Célszerű a nagyobb adatmozgatásokat DMA-val csinálni. Minden on-chip perifériának , és az L1 memóriának (Mem DMA contr.) van 2-2db DMA vezérlője külön. Ezek 3db DMA buszon osztoznak, azt arbitrálják. Átvitel kérése regiszter írással, vagy descriptorból auto-init.

2 féle DMA vezérlő van: periféria-DMA és memória-DMA-vezérlő. Ezek feladata és regiszterei eltérőek. Periféria-DMA: 12db fix csatorna van, ezek egyenként ki/bekapcsolhatóak. MemDMA: L1 és külső memória között nem 2db csatorna hanem 4db stream van (D0,S0,D1,S1) Regisztternevek: DMAx_valami, ahol x a csatorna sorszáma, a fix 12ből. Mdma-nál MDMA_xx_valami. MDMA kétszer gyorsabb.

Átvitel (csatorna) inicializálás: Minden csatornát / stream-et(memDMA) külön kell inicializálni. A DMA regisztereknek megfelelő buffert hozunk létre a memóriában, abba regiszter-értékeket írjuk. Ez a descriptor-tábla. Minden csatornához lehet több ilyen táblát rendelni, sőt azoknak láncolt listában kell lenniük. (ha csak 1 van, saját magára mutat). Tehát a descriptor utolsó regisztere egy pointer. A legfőbb beállításokat a DMAx_CONFIG, és a MDMA_xx_CONFIG (mem.dma ve. esetén) regiszterekbe kell. Itt adjuk meg pl a descriptortábla méretét, ami lehet kicsi/nagy.

Lehet descriptor mód nélkül simán MMR írással is inicializálni. Ezt a config regisztterrekben adjuk meg.

Startup: A CONFIG és/vagy a NEXT_DESC_PTR reg.eket először MMR írással kell/lehet kitölteni. (MDMA-nál előbb a cél stream-ét) Ekkor lehet tudatni velük a descr.-ok címeit.

Interrupt: Ha egy periféria-DMA csatorna engedélyezve van, akkor a periféria-IT nem vált ki interruptot, csak DMA átvitel kezdést. A DMA vezérlők kiválthatnak IT-t.

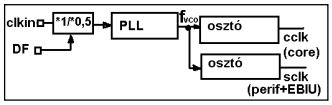
Módok: újra és újra (flow mode, autobuffer mode. a descr.láncolt lista elemeit töltögeti be és hajtja végre), vagy csak IT kérésre vagy MMR írásra (config reg.).

2D-DMA: video feldolgozáshoz lehetséges, pl egyszerre egy makroblokk betöltése L1 SRAM-ba, vagy váltottsoros TV átvitel.

Globális beállítások: A 3 buszon osztozás prioritásokat igényel. Ezeket a DMA_TC_PER és a DMA_TC_CNT regiszterekkel lehet.

Szinkronizálni user.prog.ot a DMA-val: IRQ státuszbitek pollingjával.

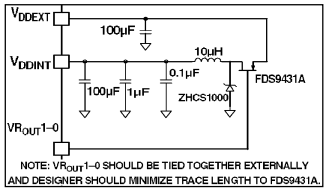
Órajel:



Változtatható működés közben az órajel (a PLL szorzó), csak a PLL behúzásra kell várni. Órajel bemenet: Kvarc (XTAL és CLKIN lábak közé.), vagy oszcillátor. Kimenet a CLKOUT lábon, sysclk sebességén. Szabályok: SCLK <= CCLK, és SCLK <= 133 MHz. Ha a DF láb=1 akkor az ½-es leosztás kapcsolódik az elejére. A PLL szorzót (1-64x) a PLL_DIV regiszterben, a többi paramétert a PLL_CTL regiszterben lehet állítani: core osztó (1-1/8), sys osztó (1-1/15). 10MHz kvarcnál reset után: 50MHz/10MHz. PLL változtatás: IT tiltás, PLL_COCKCNT (behúzási idő) állítása, PLL_DIV állítása , IT eng.

A táp:

2 feszültség: core táp-fesz (VDDINT lábak) 0,8V…1,4V, és I/O feszültség: VDDEXT lábakon, 2,25…3,6V



❖ **Beépített core táp:** Változtatható mű-

ködés közben a core tápfesz, csak a PLL behúzás-ra kell várni. BF532/531: 0,8V…1,2V, BF533 felme-het 1,3V-ig. Az kell is neki a nagyobb frekvenciához. Beállítás: VR_CTL regiszterrel. Feszültség változtatás: IT tiltás, VR_CTL írás, a PLL magától újra behúz(és a procit váratja, majd wakeup), majd IT eng. Közös föld: GND láb.

❖ **Low power módok:** A különböző low power módok állíthatják / lekapcsolhatják ezeket az órajeleket vagy aPLL-t. Módok: FullOn (normál), Active (PLL lekapcsol, 1x szorzás), Sleep (core lekapcsol), DeepSleep (sclk is off), Hibernált (clk off, VDDINT off). Ezek közti váltás: a PLL_CTL regiszter bitjeit a megfelelő mód szerint beállítjuk (még nem vált módot), majd IDLE; utasítás. Ez előtt IT tiltás, IDLE után IT engedélyezés. PLL behúzás: az IDLE után leáll a proci, a PLL ébreszti majd wakeup jellel, de mivel IT tiltva, nem lesz interrupt hívás, csak folytatódik a program.

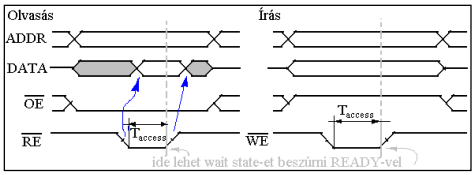
Külső busz interfész (EBIU):

Amikor a processzor címez, akkor az EBIU tudja, hogy melyik interfésznek kell reagálnia, mivel a címtartományok fixen ki vannak osztva.

❖ **Közös jelek:** Adatbusz/címbusz: ADDR[19:1], és DATA[15:0]. A0 nincs, helyette 2db dekódolt byte select: ABE/SDQM[1:0]. Van busz arbiter, külső egység kérheti a buszt a BR jellel, megkapja a BG jellel, a proci jelez a BGH jellel, hogy most már nagyon kell neki.

❖ **Az aszinkron interfész:** Aszinkron memóriáknak, flash-eknek, és perifériáknak. Lassú perifériák elérésekor automatikusan belassul a busz, az előre beprogramozott értékre (5,3…44,3 Mszó/sec). 4db bankra van osztva, mindegyik 1Mbájt, és külön select jeleik vannak, hogy ne kelljen külső címdekódert illeszteni. Bankonként külön programozhatók az elérési idők. **Jelei:** AMS[3:0] select jelek, AWE, ARE, AOE, strobe jelek, ARDY ready. Regiszterek: EBIU_AMGCTL (bank, clk enable), EBIU_AMBCTL0, EBIU_AMBCTL1 (programozható elérési idők, ready enable, ready polaritás).

❖ **Az SDRAM interfész (PC133):** 16-128MB. CL=2 vagy 3. BL=1 mindig. Jelei: SRAS, SCAS, SWE, SMS (SDRAM/chip select), SA10



(SDRAM cím10), SCKE, CLKOUT. ADDR[19:18]: SDRAM bank select jeleire! Regiszterek: EBIU_SDGCTL (fő paraméterek), EBIU_SDBCTL (enable, mem.méret), EBIU_SDRRC (auto refresh időzítés), EBIU_SDSTAT. SDRAM vezérlő parancsokat a vezérlő (SDC) automatikusan generálja, refresh/precharge közben wait state-el. SDRAM inicializálás: paraméterek beállítása a DSP-SDC regisztereiben, majd az SDRAM-módregiszter írása az EBIU_SDGCTL.PSSE bit 1-be állításával.

Perifériák:

❖ **Programozható flagek (GPIO):** A Flag, a PPI, és az SPI egyes jelei közös lábakon vannak. 16db kétirányú GPIO láb: PF[15:0]. Ezek külön kezelhetőek. Kimenet állítás / bemenet olvasás / IT törlés: FIO_FLAG_D (sima adatreg), FIO_FLAG_S (Write 1 to Set), FIO_FLAG_C (Write 1 to Clear), FIO_FLAG_T (Write 1 to Toggle) regiszterekkel. Beállítások: Irány: FIO_DIR. Bemenet engedélyezés: FIO_INEN. Bemenet szint / élérzékenység: FIO_POLAR, FIO_EDGE, FIO_BOTH - regiszterekkel állítható. Az élérzékeny bemenetek 1-et jeleznek, amíg nem töröljük az adatreg.–gel.

Hogy egy adott láb GPIO vagy más periféria-jel, azt az adott periféria konfig.regiszterében kell állítani, és persze FIO_DIR-ben legyen bemenet.

Interruptok: Minden lábhoz lehet interruptkérést rendelni (16db), de ezeket két (A, B) IRQ jellé csoportosítja a processzor számára (SIC). Az IRQ beállítható élre/szintre a . Maszk: FIO_MASKx_D, FIO_MASKx_S, FIO_MASKx_C, FIO_MASKx_T – vel állítható (x: A vagy B). Lekérdezés: az adatregiszterekből. Célszerű élérzékenyre tenni, így lekérdezhetőbb. ISR-ben törölni kell a jelzéseket az adatreg-ben.

PF	Peripheral That Shares the PF Pin		
	PPI	SPI	Timers
0		Slave Select Input (SPISS)	
1		Slave Select Enable 1 (SPISE1)	Input clock
2		Slave Select Enable 2 (SPISE2)	
3	FS 3	Slave Select Enable 3 (SPISE3)	
4	I/O #15	Slave Select Enable 4 (SPISE4)	
5	I/O #14	Slave Select Enable 5 (SPISE5)	
6	I/O #13	Slave Select Enable 6 (SPISE6)	
7	I/O #12	Slave Select Enable 7 (SPISE7)	
8	I/O #11		
9	I/O #10		
10	I/O #9		
11	I/O #8		
12	I/O #7		
13	I/O #6		
14	I/O #5		
15	I/O #4		
PPI[3:0], FS1, FS2, PPI_CLK külön van			

❖ **SPI:** Master vagy slave módban is működik. Slave módban a DSP brmenő select jele a PF0, amit itt SPISS jelnek hívnak. Master módban 8db slave select kimenet van: PF1…PF7. Egyéb jelek: MOSI, MISO, SCK. MISO és a kimenő select jelekre 10kOhm pullup kell. Regiszterek: SPI_BAUD, SPI_CTL, SPI_FLG, SPI_STAT, SPI_RDBR, SPI_TDBR .

❖ **PPI:** 4-16bites adatok küldése/fogadása max SYSCLK/2 sebességgel. Csak DMA-val megy. Mód van az adatok 8/16bites auto-csomagolására. (Lsb:korábbi bájt) **Regiszterek**: PPI_CONTROL, PPI_STATUS, PPI_DELAY, PPI_COUNT, PPI_FRAME . Van egy 16x16bites FIFO-ja. **Automatikus SAV/EAV** kódszekvencia detektálás: csak input módnál, ha keretszinkron jeleket nem használunk, detektálja a stream-ben, és beállíthatóan pl kioltás alatt nem másol. Kimenetkor az adatfolyamba manuálisan bele kell tenni a kódszekvenciákat, és ez alapján generál szinkron jeleket. **Jelek:** adatjelek: PPI0…PPI15, 3db szinkron PPI_FS1…3 (konfigurálható. Az FS3 GPIO-val közös lábon), és egy külön órajel bemenet: PPI_CLK . Frame sync: FS1(hs)/Timer1,FS2(vs)/Timer2,FS3(field)/PF3 közös láb.

❖ **SPORT:** 2db van: SPORT0, SPORT1. Max SCLK/2 sebességig működik. 1db SPORT-hoz 2db adatbe (DRxPRI, DRxSEC: primary / secondary), 2db adatki (DTxPRI, DTxSEC: primary / secondary), 2db órajel (TSCLKx, RSCLKx), és 2db keretszinkronjel (RFSx, TFSX) tartozik. 4x32 vagy 8x16bites FIFO van. Előre kell inicializálni, működés közben nem lehet.

❖ **UART:** PC-hez hasonló szabvány kezelés. Képes IrDA jeleket fogadni és adni. Jelei: RX, TX. Regiszterei 8bitesek (16bites, csak az alsó 8 használt). Több regiszter ugyanazon a címen van, egyszerre egy csoport hozzáférhető. Hogy mikor melyik, azt az UART_LCR regiszterben adjuk meg (ez az egy nem osztozik, fő kontr.reg). Ezenkívül a bemenő/kimenő adatregiszterek is közös címet kaptak. UART_DLH(baud rate osztó felső)-UART_DLL(baud rate osztó alsó) közösen, valamint a UART_THR(kimenő adat FIFO eleje)-UART_RBR(vett adat FIFO vége)-UART_IER(interrupt en) van közösen. Különálló regiszterek: UART_LCR(fő kontrol), UART_GCTL(másik fő kontr.) UART_MCR, UART_LSR(státusz), UART_IIR (interr. státusz).

❖ **Számlálók:** 3db 32bites általános timer, és egy core timer + WDT.

Az általános timerek: Mehetnek PWM kimenet, időmérés/PWM-mérés, és külső jelet számláló módban. Mindegyiknek van egy I/O lába: TMRx. Regiszterek: TIMERx_CONFIG (csak akkor írható ha a számláló nem megy: TIMER_DISABLE reg.), TIMERx_COUNTER, TIMERx_PERIOD, TIMERx_WIDTH. Órajel: vagy a külső lábról (TMRx, PF1, PPI_CLK, RX) vagy a sysclk-ról mennek. Közs regiszterek: TIMER_ENABLE, TIMER_DISABLE, TIMER_STATUS. Autobaud detektálás UART-nak: RX-ről clock-oljuk, capture módban a startbit ('a' kar.), vagy a teljes bájt ('0' kar.) szélességét megmérjük.

Core timer: periódikus interrupt generálásra. Órajel: CCLK belső jel. Regiszterei: TCNTL, TCOUNT, TPERIOD, TSCALE (előosztás). Lefelé számlál.

Watchdog: SCLK-val megy. Defaultból off. Okozhat resetet, vagy interruptot. Regiszterei: WDOG_CTL, WDOG_CNT, WDOG_STAT.

❖ **RTC:** Számlálókból áll: 60sec, 60perc, 24óra, 32768nap (max. 89 év). IC-lábak: RTXI, RTXO, ezekre 32,768kHz-es kvarc kell. Külön táp: VDDRTC. Van beépített alarm, és stopper. Regiszterek: RTC_STAT (aktuális idő), RTC_ICTL, RTC_ISTAT, RTC_SWCNT (stopper), RTC_ALARM (mikor ébreszt. IRQ), RTC_PREN (előosztás). Másodpercenként csak egy regiszter írható.

❖ **Beépített debug:** J-tag (TCK, TDO, TDI, TMS, TRST, EMU), Watchpoint egység (regisztrálja hogy adott címet hányszor értünk el.), trace egység (utóbbi ugrásokat tárolja), Performance-monitor egység.

Boot:

A BMODE[1:0] lábakon kell megadni, hogy milyen módon akarunk bootolni. A ROM-ban lévő bootloader ez alapján bootol.

Párhuzamos flash: BMODE=01. Boot a rendszerbuszról. A 0. aszinkron mem.bankból. RDY aktív magasra érzékeny, hold time=3ciklus, access time=15ciklus.

SPI master boot: BMODE=11. 8/16/24bitesen címezhető SPI flash memóriákból. A 0-s címről az L1 utasítás-SRAM elejére. Sebesség: 500kHz A per.flagek közül PF2 lesz a mem.CS jele. MISO, PF2: pullup

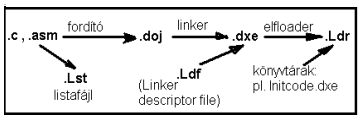
SPI slave boot: BMODE=10. uC bootolja a DSP-t.

No boot: BMODE=00. Utasításvégrehajtás indul a 0x2000'0000 Címről. Ez aszinkron külső terület, bank 0.

2. Szoftver

RISC-hez hasonló utasításkészlete van, lehetőség van a többszörös adattal (1/2/4) való számításokra. Hosszú/rövid opkódok (VLIW).

Fájlok a VDSP++ -ban:



Az assembler:

Algebrai assebly nyelve van (pl uatsítás formátum ilyen: R3=R7+18;)

Pontosvessző kell minden utasítás-sor és direktíva után. Preprocesszoros dolgok után nem kell. Komment: c-stílusú, azaz //komment vagy /* komment */

Címke: mint máshol, kettősponttal utasítás előtt.

❖ **C-struktúrák:** pl. c-ben megadtunk egy ilyet: struct név {int egyik; int másik;} Ezt asm-ben így lehet elérni: A=_név->egyik;

❖ **Preprocesszor-utasítások:** #INCLUDE „fájlnév.h" sima include. Konstans megadása fordítónak (nem tárolódik!): #DEFINE név 125 Makró megadása: #DEFINE név(a,b) ((a)=(b)+1)

❖ **Programszerkezet:**

```
#INCLUDE "név.h"
#define név 123
   …
   .SECTION data1;
   .VAR   nev1;
   .VAR   nev2[];
   …
   .SECTION prog;
címke:  utasítas;
        utasítás;
```

❖ **Műveletek: (operátorok)** Ezeket egy-egy opkódba képezi le, vagy fordító számítja ki, ha ki lehet akkor. Az asm programok egyszerű kifejezésekből állnak (pl A=B+C;). Ezek végére pontosvessző kell. Elé lehet címkét tenni. Műveletek: = (legyen egyenlő), == (feltételben), ( ), ~ (1es komplemens), +, -, *, /, % (modulo), >>, | (bitenkénti vagy), &&, BITPOS(15)... Pontos alkalmazásuk az utasításkészlet leírásában!

❖ **Számok:** Integerek vagy tört. Integerek: **0Xszám** –ez a hexa, **B#szám** –ez a bináris, **szám** –decimális, **szám.szám e +szám** –ez lebegőpontos. (pl.: 2.77 e+5)

Törtek: **szám r** ez a –1…+1 közötti tört megadás 16bitre (pl.: –0.55 r)

❖ **Direktívák:** pontosvesszővel a végükön kell ezeket is.

Globális szimbólumok: [.GLOBAL név;] ahol definiáltuk a változót, és [.Extern név;] –ahol hivatkozunk rá.

.ENDSEG;	Szekció vége.
.EXTERN név;	Hivatkozáshoz kell. (c-változóra: .extern _név;) fájl elejére.
.EXTERN struct név;	Struktúrára hivatkozáshoz
.GLOBAL név;	Szimbólum láthatóvá tétele máshol
.INC/BYNARY „fájl";	Fájl tartalmának bemásolása adott helyre
.SECTION név;	szekciókezdet
.VAR x1; **.VAR x2[10];**	Változó/konstans deklarálás (aktuális szekcióba) 32bites. Minden RAMban lesz, nincs különbség változó/konstans között. 10elemű tömb.
.BYTE x3; **.BYTE sz[20]='abc',0;**	Változó/konstans deklarálás (aktuális szekcióba). 32/16/8 bitesen: BYTE4 / BYTE2 / BYTE. Ezzel is lehet tömböt. String, lezárással.

Konstans mintatáblázat megadása: .BYTE2 név[]= 1, 2, 3, 4, 5;

❖ **Szekciók:** A program két féle szekciót tartalmaz: kód és adat szekció. Ezek neve tetszőleges. Változók/kód írása előtt legyen a szekciómegadás, pl.: .SECTION program; Ezeket a linker descriptor fájlban soroljuk be memóriacímekhez. Vannak default szekciók

❖ **Címzés: 32bites adatot:** R4=[P5]; vagy R3=R2; vagy R1=12345678; vagy A1=R2; **16bites adatot:** R5.L=W[P5]; vagy R5.L=R1.H; vagy R1.H=1234; vagy A.L=R4.H; **8bites adatot:** R6=B[P4]; vagy R6=R5.B; R4=A0.X (A felső 8 guard bitjét). N bájtos adat csak N-nel osztható címet kezdődhet !!! Video uasításoknál nem.

Valami.L a belső regiszter alsó 16 bitje, valami.H a felsőé. Belső regiszter alsó bájtja: valami.B . Memóriában lévő 16bites adat: W[P5] 32bites: [P4] 8 bites: B[P1] Bizonyos adat/szám mozgatáskor kell választani előjelkiterjesztést (X), vagy 0-kiterjesztést (Z), ha nagyobb regiszterbe töltünk kisebből. Pl. R2=[P1+12] (X);

Adatmemóriát csak indirekten lehet címezni. Pl R5=[P4]; azaz a P4 által mutatott címről tölt be. Ebbe beletartozik a bázisrelatív címzés is, ahol R3=[P5+1234]; ahol a P5-ben tárolt bázis + (offszet=1234) a cím, ahonnan betöltünk. Lehet pre/post inkrementálni, dekrementálni. Pl R4=[P4++]; Ez volt a post-inkrement. **Az offszet** (előjeletlen) nem lehet akármekkora: 32bites adatnál: max 128kB tartomány, 16bites adatkor: max 64kB tartomány, 8bites adat: max 32kB tartomány!!!! **Konkrétan a memória / MMR címzés:** Elejére: #define modreg1 0x543 (offszetcím) később: P5=LAN_bazis; R4.L=W[P5+modreg1];

Program memória címzése: JUMP, CALL utasításoknál. Lehet direkt címezni (pl. cimkével), de ez is inkrementális! 25bites előjeles számmal,azaz max +/-16MB távolságra. IF cc JUMP esetén max +/-1kB

Utasításkészlet:

❖ **Programvezérlés: Ugrások:** alapvetően inkrementális, de az assembler kiszámolja a cimkékből az offszetet. Ne ugorjunk nagyobbat, mint amekkorát lehet! Delta max: 25bites előjeles bájtcím, azaz +/-16MB. JUMP címke; JUMP delta; JUMP (P1); -indirekt ugrás. **Feltételes ugrás:** IF cc JUMP címke; cc értéke alapján. **Szubrutinhívás:** CALL címke; vagy CALL (P1); indirekten. **Visszatérés:** attól függ honnan. Szubrutinból: RTS; interruptból: RTI; kivételből (exception): RTX; NMI-ből: RTN; emulációból: RTE; . **IT kényszerítés:** RAISE n; ahol n=0…15.

Hardver programhurok: Egyszerre 2 hurok mehet (külső/belső) a 2db ciklusszámláló miatt (LC0, LC1). 2 módon adhatók meg: **1.)** LSETUP (kezdőcímke, végcímke) LCO=P4; Ekkor LC0-t használjuk, a hurokszámot P4-ből adjuk neki. **2.)** Elnevezzük a hurkot: LOOP név LC1 = P5; LOOP_BEGIN = kezdocimke; LOOP_END = végcímke; .

❖ **Feltételbit állítása:** A feltételt összehasonlító műveletekkel lehet állítani, vagy direktbe: cc=AZ; vagy cc=AN; vagy bármely flag az ASTAT regiszterből, néven nevezve.

❖ **Load / store:** Hova lehet: A0, A1, R, P, I, L, M, B. **Számot belső regiszterbe:** R7.L=0x1234; (16bit) vagy R7=0x12345678; (32bitesen). **P / R feltöltés memóriából:** Csak indirekten címezhető!!!!!!!: R5=[P5]; Közben előjelkiterjesztés vagy 0kiterjesztés lehet, zárójeles (X) vagy (Z) opcióval. Pl R7.H=[P2] (X); Címhez offszetet lehet/szoktak adni (mivel máshogy nem címezhető a memória): R1=[P2+158]; **Vissza:** [P4]=R5; Látsd a címzéseket kicsit visszább!!

❖ **Move:** P / R / I / L / M / B regiszterek között: R6=R7; P1=R6; A0=A1; R7=A1; (csak páratlan R-et lehet páratlan A-hoz.) **Feltételes másolás:** IF cc R6=R4;

❖ **Stack:** push: [--SP]=R7; pop: R4=[SP++];

❖ **Összehasonlítás:** Először feltételflag beállítás: cc = R6 = = R5; Azaz cc=1 ha R6=R5. lehet <, >, !=, <=, >=. Lehet hasonlítani számmal.

❖ **Logika / aritmetika:** Logikai: R5= R3&R1; Aritmetika: összeadás/kivonás: R5=R4+R1; Számot csak +/- 64 közöttit lehet hozzáadni/kivonni: R2+=12; Abszolút érték: R1= ABS R2; Van min/max keresés… szorzás: 16x16bites: R3.L = R2.L * R1.H; 32bites: R3*=R2; MAC: 16x16 bites 40biten akkumulál, majd 16 bitet tartunk meg a végén. R1.L=(A0+=R5.L*R4.L);

Ilyen műveletekkor lehet szaturálást kérni: R4=R2+R1 (S);

❖ **Üzemmódok:** Low power módba: IDLE; disable IT: CLI R5; ekkor az IT maszk regisztert elmenti R5be. enable IT: STI R6; Szoftver IT: RAISE n; software kivétel: EXCEPT n;

❖ **Cache kezelés:** PREFETCH [P1]; P1 által mutatott című adatok környékét az adat cache-be tölti. FLUSH [P4]; azokat üríti, azaz ha nem lett felülírva törli, ha igen (piszkos lett) kiírja a külső memóriába.

❖ **Video műveletek:** 8bites adatok ki/be csomagolása, interleaving 1-4 adattal egyszerre…

❖ **Vektor műveletek:** Vektorhossz, összeg, skálázás, összehasonlítás, vekt.szorzás, 180°-os elforgatás, vektor képzés számokból.

❖ **Párhuzamosítás:** pl. R4.L=R3.L+R2.L || R4.H=R3.H+R2.H;

A linker:

A linker descriptor fájl (.LDF) írja le a memóriába való elhelyezést. Több processzorra is lehet linkelni a dolgokat. Input szekciók (.asm, .c, .doj fájlban): amiket az asm fájlban megadtunk. Output szekciók (.dxe): DATA és Program. Ezeket helyezzük el a memória-területeken, pl SDRAM, L1 progmem… Meg kell adni az adott memóriák bitszélességét is.

Vannak előre elkészített linker fájlok minden procihoz, és még változatok is.

Az expert linker-rel lehet vizuálisan linker fájlt előállítani.

```
MEMORY          memória darabok
{                  (pl. L1 sram)
 szegmens1 {típ,cím,méret,bit.}
 …
}
PROCESSOR p0
{ …}
SECTIONS
{
 Program  ←output szekció
  {
    saját szekc1 ←input szek.
    saját szekc2   (asm-ből)
  }>szegmens1
 DATA  ←output szekció
  {
    saját szekc3
  }>szegmens2
}
```

Startup:

1.) Reset vektorra ugrás: EF00'0000 cím. Ott van az on-chip ROM, benne egy bootloader. Ez bootloáskor ha talál INIT_CODE-ot a betöltendő kódban, akkor azt meghívja. Az visszaadja a vezérlést (RTS;) a bootloader-nek.

2.) A bootloader a FFA0'0000 címre adja át a vezérlést BF533 esetén. BF532/531 esetén FFA0'8000 címre. Innen indul a saját programunk (ha nem használunk U-boot -ot) vagy az U-boot.

3.) Indul az U-boot, ha használunk ilyet. Ez indítja a saját programunkat (ha nem használunk Linuxot), vagy a Linuxot. Vezérlésátadás: ahova a programunkban a programszekció elejét tettük.

❖ **Az INIT CODE:** Ez beállítja az SDRAM-ot, MMR-eket. Enélkül pl nem lehetne SDRAM-ba bootolni. Külön project a VDSP++ -ban. Ilyenek vannak készen is, .Dxe fájl formájában. Ha mi magunk csinálunk ilyet, tudatni kell a VDSP-vel, hogy a project INIT kód project lesz.Ez nem az, amit startup-kódnak hívnak! Az linuxos projectben egyéb HW inicializálás.

```
.SECTION program
…utasítások…
RTS;
```