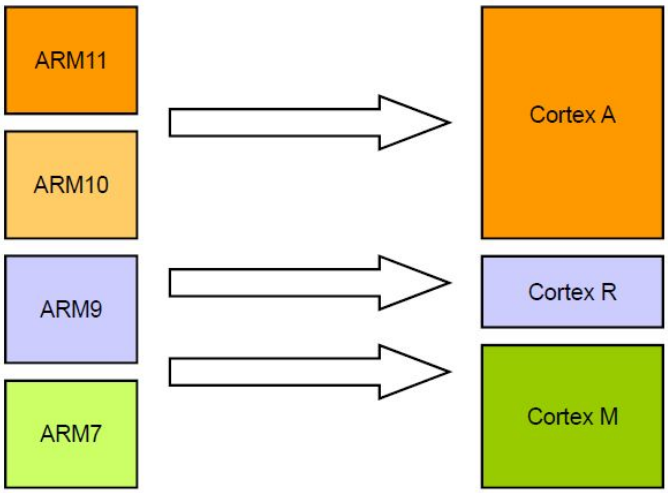
1. Mutassa be, a mikrovezérlő piac alakulását az elmúlt 20 évben! Mutassa be az ARM cég filozófiáját és a hagyományos ARM magok: ARM7, ARM9, ARM11 jellegzetességeit. Mutassa be az ARM Cortex mag sorozatának (M, R, A) jellegzetességeit! Melyik Cortex-es sorozat melyik hagyományos mag sorozat kiváltására készült!

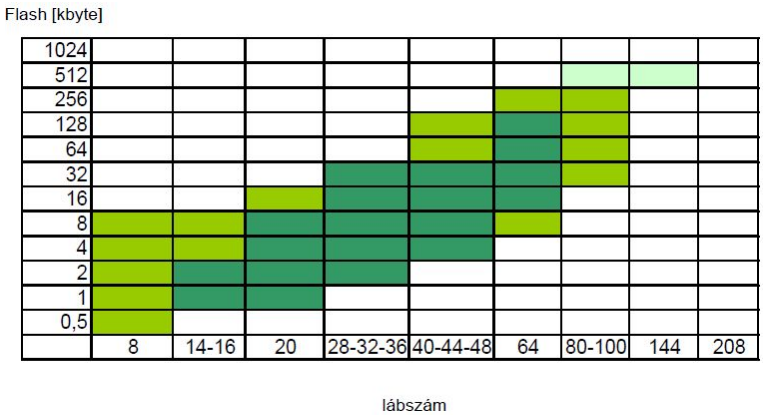
1980: 8051: 10 MHz
1993: PIC16C84 - EEPROM
1997: Atmel ISP - 8051-es magú, Flash memóriával rendelkezik, nem kell hozzá nagyobb feszültség
2003: Első ARM7-es alapú mikrovezérlő
2005: ARM9
2006: Cortex M3
2010: M4
2015: M7



Cortex A- "Application" - operációs rendszerrel (pl. Linux, Android) rendelkező magok, mint a pl. Raspberry Pi
Cortex R - "Real Time" - Valósidejű alkalmazások
Cortex M - "Microcontroller"

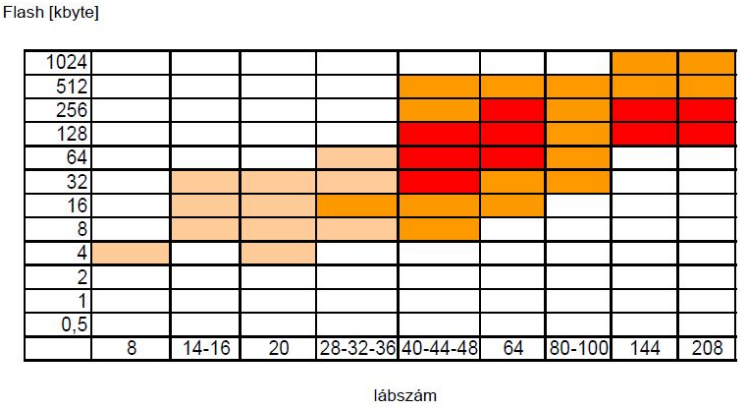
2. Hasonlítsa össze a 8bites és 32bites mikrovezérlők lábszámának és Flash memóriájának, változását az elmúlt 10 évben, ábrázolja ezt a lábszám - flah memória koordinátarendszerben!

8 bit:



A fejlesztések az alsó régiót célozták meg (kisebb memória és lábszám), ahol a fő szempont az alacsony ár és kis fogyasztás.

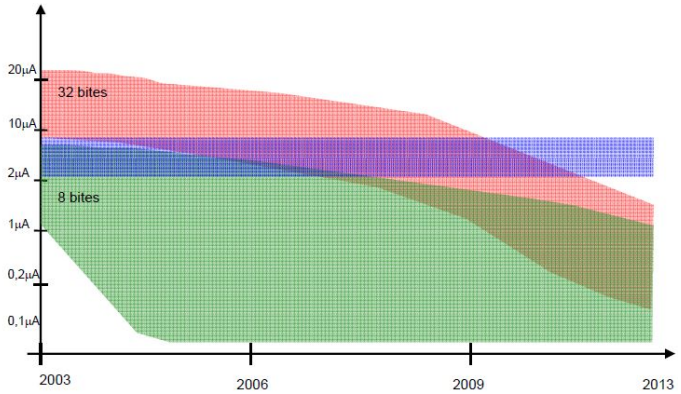
32 bit:



A 32 bites mikrovezérlők egyaránt felfelé és lefelé bővítették a határt. Az ARM architektúrájú mikrovezérlőknél a Cortex M0 sorozat az alsóbb régió felé bővült (alacsony ára és fogyasztás), hogy kiszorítsa a 8 bites mikrovezérlőket, a Cortex M3/M4 sorozatokkal pedig a felsőbb régiók felé (nagyobb flash méret, teljesítmény)

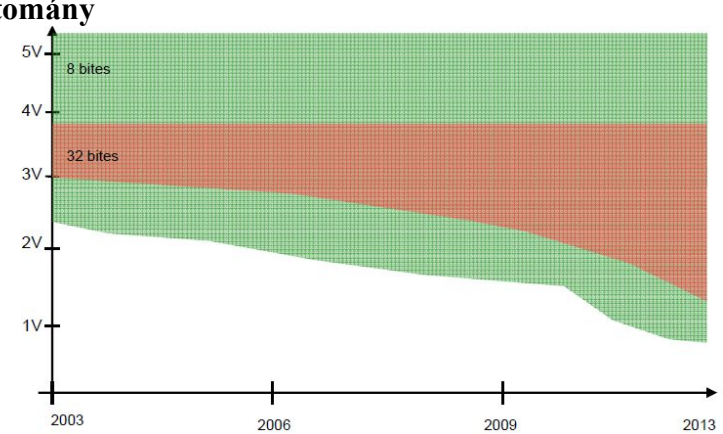
3. Hasonlítsa össze a 8bites és 32bites mikrovezérlők sleep fogyasztásának és működési feszültség tartományának, árának, valamint perifériakészletének alakulását az elmúlt 10 évben! Ismertesse a 8bites és 32bites mikrovezérlők választása mellett és ellen szóló legfőbb érveket!

Sleep fogyasztás:



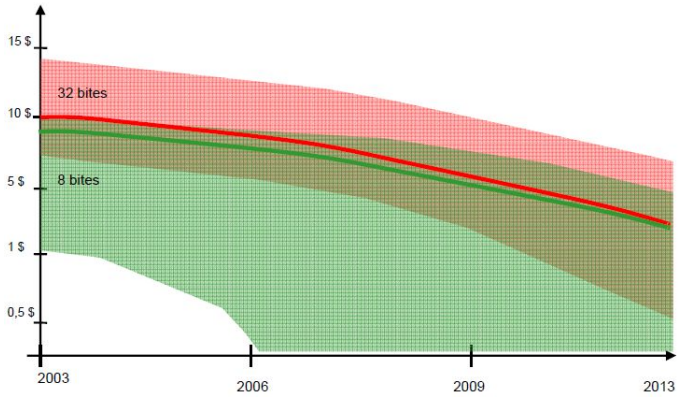
Az elmúlt 10 év alatt jelentősen csökkent a sleep fogyasztása a 32 bites mikrovezérlőknek. Sleep fogyasztás szempontjából több hátrányuk is van a 8 bites társaikhoz képest mint a nagyobb SRAM, több láb, több periféria. Ezt úgy kompenzálhatják, hogy ha nincs szükség a SRAM-ra akkor azt is kikapcsolják és a nagyobb órajelt igénylő perifériákat.

Működési feszültség tartomány



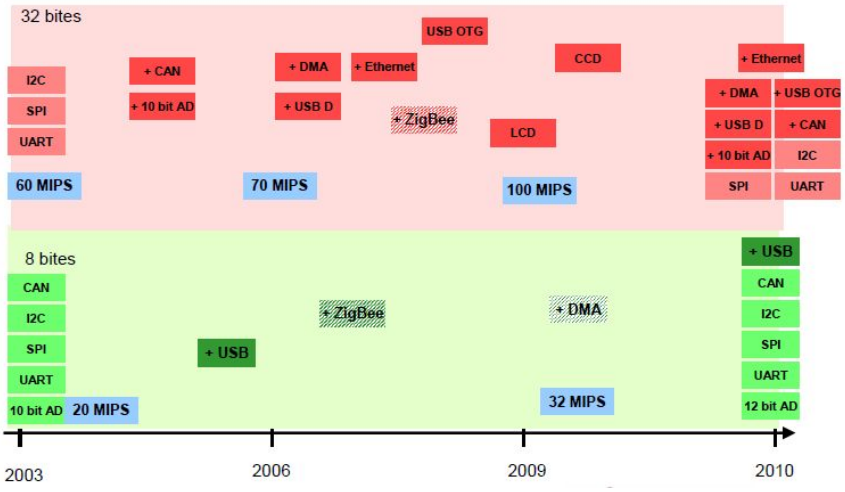
Az elmúlt 10 évben lefelé tágították a működési feszültség határát. A nagyobb feszültségű logikai szint (5 V) előnye a kisebb zajérzékenység, de hátránya a nagyobb fogyasztás, ami kb a kétszerese 3.3 V-hoz képest.
Az alsó határ kibővítésének egy célja hogy egy ceruza elemről is működhessen a mikrovezérlő.

Árak alakulása

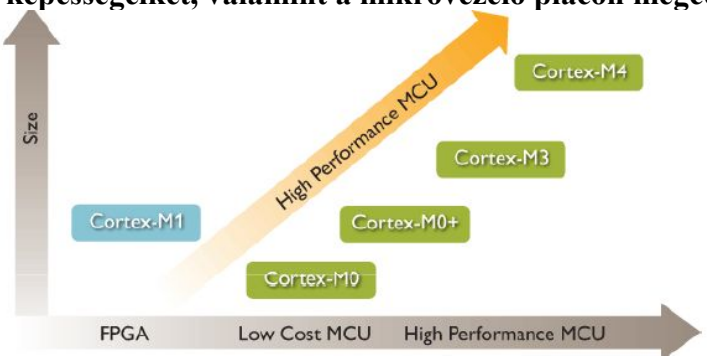


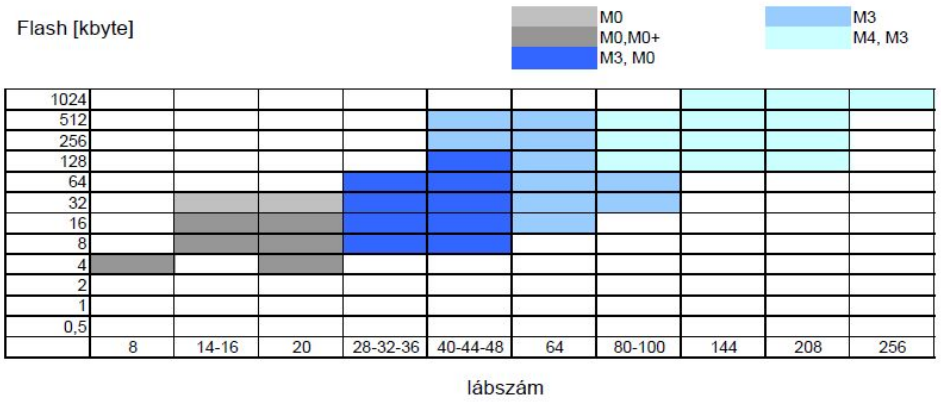
A 32 bites mikrovezérlők áruk is jelentősen lecsökkent, de még nem tudják felvenni a versenyt a 8 bites mikrovezérlőkkel. (ATtiny)
Főbb érvek a 8 bites mikrovezérlők mellett:
- ár (elérhető áron 32 bites mikrovezérlők is)
- fogyasztás
- fejlesztőkörnyezet
- kevésbé tápfeszültség érzékeny
- egyszerűség
- népszerűség

Főbb érvek a 32 bites mikrovezérlők mellett:
- nagyobb számítás kapacitás
- magasabb órajel
- több periféria
- nagyobb SRAM
- nagy kódsűrűség



4. Mutassa be az ARM Cortex M sorozat tagjainak (M0, M3, M4, M7) képességeit és hasonlítsa őket össze. Röviden adja meg képességeiket, valamint a mikrovezélő piacon megcélzott szerepüket!





Cortex M0:
- egyszerű felépítés
- kis fogyasztás
- Neumann architektúra
- 3 lépéses pipeline (M0+ 2 lépéses)
- alacsonyabb lábszám
- WIC (Wake up Interrupt Control) blokk
- 8 bites mikrovezérlők kiváltására szolgál

Cortex M3:
- nagyobb teljesítményű utasítások (ARM7-hez képest)
- 2 külön stack
- specifikált memória térkép (ARM7-el ellentétben)
- Harvard architektúra
- egyszerűbben programozható (ARM7-hez képest)
- Thumb2 utasítás
- 3 lépéses pipeline

Cortex M4:
- "kibővített M3-as"
- Harvard architektúra
- Thumb2 utasítás
- DSP, SIM és MAC utasítások
- kód kompatibilis az M3-al
- órajel > 100 Mhz
- 3 lépéses pipeline
- 3 AHB-lite busz
- FPU
- Debug, trace
- Memória védelmi egység

Cortex M7:
- megközelítőleg kétszeres teljesítmény az M4-hez képest
- 6 állapotú pipeline
- superscalar
- branch prediction
- DSP
- TCMP
- cache

5. Ismertesse az ARM7 magok jellemzőit: utasításkészlet, architektúra működési módok, interrupt kezelés! Mutassa be a Cortex M3 processzor mag fő jellemzőit: architektúra, főbb blokkok, újdonságok az ARM7-hez képest!

ARM7
- kevés utasítás, kis komplexitás
- 3 elemű pipeline
- alacsonyszintű megszakítás kezelés
- 2 utasítás készlet: 32bit: ARM, 16bit THUMB
- gyártó függő memória elrendezés
- bonyolult kezelés
- Align memória kezelés
- Load and Store Architektúra
- Alacsonyszintű interrupt kezelés, 6 fajta működési mód (User, Fast Interrupt (FIQ), Interrupt (IRQ), Supervisor, Abort, System) (szókép: sasfiu)

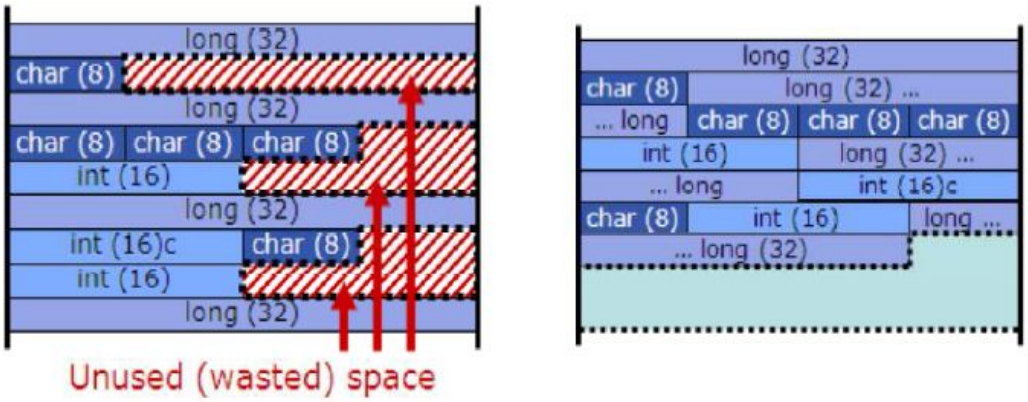
Cortex M3:
- THUMB2 - egyszerre tartalmaz 16/32 bites utasításokat
- 3 lépéses pipeline
- nagyobb teljesítmény
- Harvard architektúra
- egyszerűbb programozó modell
- újdonságok: NVIC és IT rendszer, System Timer, fejlesztett debug rendszer, memória térkép, bitbanding
- non-aligned memória elrendezés
- magasabb szintű interrupt kezelés (NVIC),
- 2 működési mód (Thread, Handler mód)

6. Mutassa be röviden a Cortex M3 alapú vezérlők memória szervezésének jellemzőit. Milyen főbb cím tartományok vannak, mi az a bit banding, mi a non-aligned memória hozzáférés. Mik a főbb különbségek az ARM7 alapú vezérlőkkel szemben!

- pontosan specifikálva van az egységes memória térkép (az ARM7 és ARM9-nél nem volt)
- 4 GB címtartomány:
  - kód és SRAM terület
  - On chip periféria
  - külső memória és külső egységek
  - Cortex regiszterek

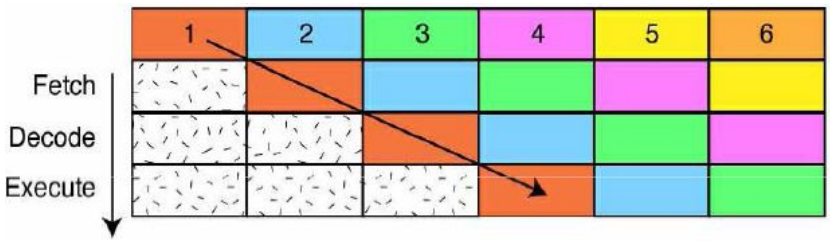
Bit banding
- bitenkénti címzés gyorsítása
- direkt bit vezérlést ad, és ez miatt nincs szükség az AND/OR maszkolásra

Aligned - csak a megadott címnél kezdődhetnek (ARM7)
Non-aligned - nem csak a megadott címről kezdődhetnek (Cortex M3)



7. Mutassa be a Cortex M3 utasítás végrehajtását (pipe-line), programozói modelljét (load and store), regiszter készletét! Mutassa be a Cortex M3 működési módjait és hasonlítsa ezeket össze az ARM7 működési módjaival!

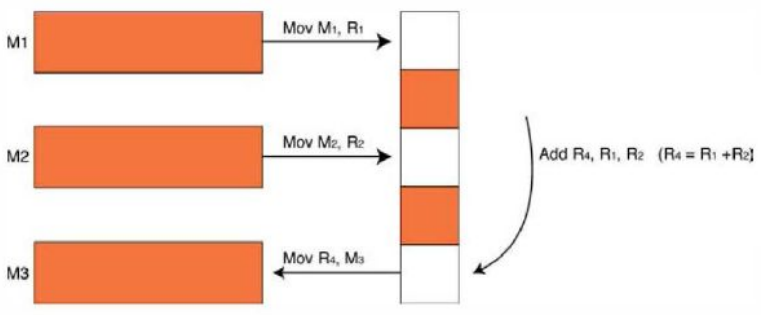
Pipe-line



Elágazás becslés: feltételes elágazásnál mindkét iránynak az utasításait elkezdi felhozni (jelentősen növeli a teljesítményt az ARM7-hez és az ARM9-hez képest).

Load and Store

Minden adatot be kell mozgatni a memóriából a regiszterbe és csak utána lehet a műveletet végrehajtani. A végén a művelet eredményét a regiszterből a memóriába kell írni. Így működik az ARM7 és az ARM9 is.



Hozzáférési módok:
- privileged: reset után ez aktív, minden processzor erőforrásához hozzáférést biztosít, kivétel vagy interrupt hatására is ebbe a módba lép be a processzor.
- unprivileged: korlátozott hozzáférési mód, nem lehet pl. az NVIC és a SysTick regiszterekhez hozzáférni

Működési módok:
- Thread mód: normál működési mód, lehet privilegizált, vagy unprivileged
- Handler mód: kivételezés, interrupt kezelés, mindig privileged

Az ARM7-nek 6 működési módja van: User, Fast Interrupt (FIQ), Interrupt (IRQ), Supervisor, Abort, System

8. Mutassa be a Cortex M3 standard perifériáit (System timer, NVIC, debug blokkok) és az ezek által nyújtott előnyöket! Mutassa be a piacon kapható legelterjedtebb Cortex M3 alapú mikrovezérlő sorozatokat!

System timer:
- 24 bites lefelé számláló
- rendszer órajelével, vagy annak 1/8-áról működhet
- 3 regiszter: számláló, reload, status (IT engedélyezése, Start, stop, Timer config)
- egyesített rendszer számláló a Cortex M3-ra épülő mikrovezérlőkhöz (RTOS Heart-beat timer)

Memórizálásra segítség:
24 * 1/8 egy. 3

NVIC (Nested Vector Interrupt Controller):
- Standard, gyártó független (könnyű portolhatóság)
- THUMB2 több órajelig tartó utasítások megszakíthatóak (determinisztikus IT kezelés)
- támogatja a Nested Interrupt-okat (magasabb prioritású interrupt-ok megszakíthatják az alacsonyabbakat, preemptív)
- processzor független, de a tervezők megszabhatják a bemenő vonalainak számát)

Memórizálásra segítség:
stpt

Debug blokkok

Piacon kapható legelterjedtebb Cortex sorozatok:
- STM32 F1, F2, L1, W
- NXP - LPC 17XX, 18XX

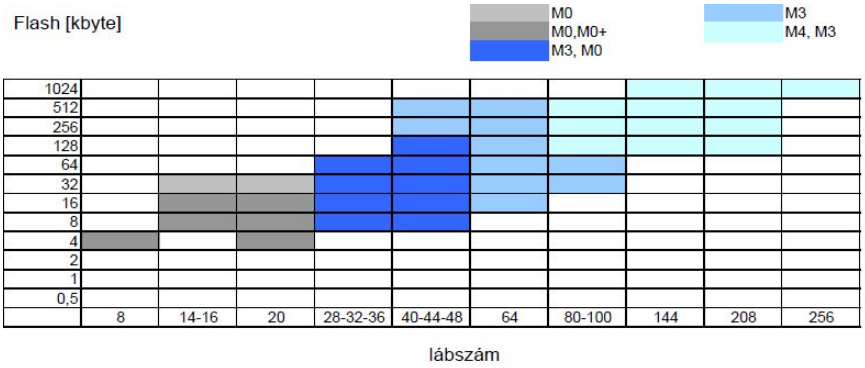
9. Mutassa be a legelterjedtebb ARM7 magú és Cortex M3 magú mikrovezérlő sorozatok belső felépítésének fejlődését az elmúlt 10 évben! Milyen változásokat mi indokolt?

ARM7:
1. Gen:
- 1 AHB (Advanced High-performance Bus)
- 1 APB (Advanced Peripheral Bus)

2. Gen:
- Dual AHB busz
- DMA lehetőség

- több periféria
- fast I/O

Cortex M3:



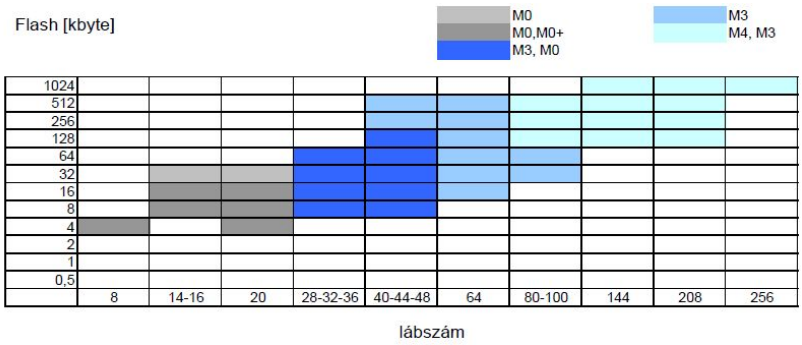
1. Gen.: 1 APB busz
2. Gen.: STM32F103:

- Két APB busz, a gyorsabb és a lassabb perifériáknak max 72 MHz
- AHB busz mátrix

3. Gen.: STM32F107
4. Gen.: STM32F2xx

LPC210x
- 1 AHB (Advanced High performance Bus)
- Neumann architektúra
- 1 APB (Advanced Peripheral Bus)

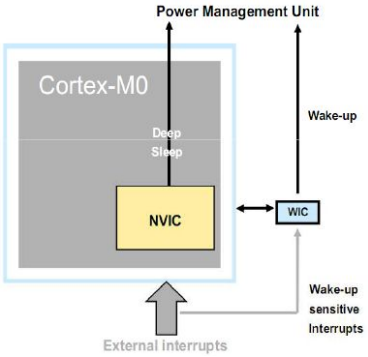
10. Mutassa be röviden a Cortex M0 mag jellegzetességeit, miben különbözik a Cortex M3 magtól és az ARM7 magtól! Mi az a WIC (Wake-up Interupt Controller) és miért fontos az energiatakarékosság szempontjából?



Fő célja, hogy kiváltsa a 8 bites mikrovezérlőket, ezért fontos szempont a kis fogyasztás és ár.

- 3 lépéses pipe-line (M0+ 2 lépéses)
- kis fogyasztás
- kisebb lábszám
- Neumann architektúra (ARM7 is Neumann, de az M3 Harvard)
- kisebb számítási kapacitás (M3-hoz képest)
- egyszerű felépítés
- csak 4 interrupt priority level
- 16-bit Thumb utasításkészlet kiegészítve a Thumb-2 technologiával.
- 2 működési mód (Thread mód és Handler mód) az ARM7-el ellentétben.

WIC blokk lehetővé teszi a deep sleep-ből való felébredést (NVIC és a mag is leáll)
Deep sleep - kisebb fogyasztás, a systick sem dolgozik

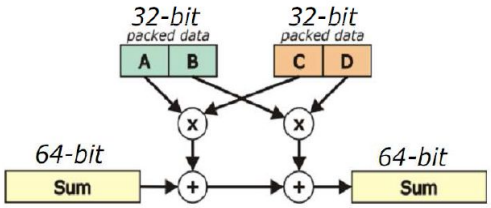


11. Mutassa be a Cortex M4 mag jellegzetességeit! Mik azok a SIMD és MAC utasítások és miért lehetnek ezek hatékonyak jelfeldolgozásra? Milyen további fejlesztéseket hoz az M7 mag az M4-hez képest?

- Thumb2 utasítások
- DSP, SIMD és MAC utasítások
- 100 MHz feletti órajelre tervezték
- Harvard architektúra
- FPU (Floating Point Unit)
- 3 fázisú pipe-line
- 3 AHB-lite bus
- Debug, Trace
- Memória védelmi egység
- kód kompatibilis az M3-al

A SIMD (Single Instruction Multiply Data) akkor hasznos ha nagyobb mennyiségű ugyan olyan típusú adatokra ugyan azt a műveletet kell elvégezni, pl vektor műveletek.

Sum = Sum + (A x C) + (B x D)



MAC utasítás - egy órajel segítségével képes összeadni és megnövelni az akkumulátor értékét. Igen hasznos mivel a FIR szűrőnél is ezek a műveletek vannak (szorzás és összeadás)

$a = a + b \cdot c$ - ezt elvégzi egy órajel alatt

Cortex M7

- megközelítőleg kétszeres teljesítmény az M4-hez képest
- 6 állapotú pipe-line
- branch prediction
- cache
- super scalar
- TCM

12. Mutassa be, hogy az ARM Cortex M0 és M4 magú vezérlők tipikus belső felépítését. Mik a hasonlóságok és különbségek? Mutassa be az NXP LPC4300-as sorozatának felépítését! Milyen szerepet szánnak az M0-ás és az M4-es magnak, mutasson erre példát!

Cortex M0:
- egyszerű felépítés
- kis fogyasztás
- Neumann architektúra
- 3 lépéses pipeline (M0+ 2 lépéses)
- alacsonyabb lábszám
- WIC blokk (Wake up Interrupt Controller)
- 8 bites mikrovezérlők kiváltására szolgál

Cortex M4
- összetettebb felépítés (M0-hoz képest)
- nagyobb teljesítmény (M0-hoz képest)
- Thumb2 utasítások
- DSP, SIMD és MAC utasítások
- 100 MHz feletti órajelre tervezték
- Harvard architektúra
- FPU (Floating Point Unit)
- 3 fázisú pipe-line
- 3 AHB-lite bus
- Debug, Trace
- Memória védelmi egység
- kód kompatibilis az M3-al
- WIC blokk (Wake up Interrupt Controller)

NXP LPC4300
- Cortex M4 és M0 magok vannak összeintegrálva
- 2 bankkos flash
- szétválasztható a feldolgozás a real-time vezérlés
- osztott memórián keresztül képesek kommunikálni

M4 (jelfeldoglozás) + M0 (periféria kezelés, kifelé kommunikáció)

13. Mutassa be a Reset és elindulás folyamatát egy tipikus ARM Cortex magú vezérlőn! Milyen műveletekre és miért van szükség ahhoz, hogy a C program main függvénye elindulhasson?

Reset forrása lehet:
- Power on
- Watchdog
- külső láb
- szoftveres

A reset forrása kiolvasható egy regiszterből

A Cortex sorozatnál a 4-es címről indulunk, ahol végrehajtódik a Reset vektor. Ez után már megvan a stack pointer, tehát nem kell inicializálni.
Sok esetbe na Reset után egy boot kód szokott következni, amely végrehajtása után átadja a normál kódnak a vezérlést.

A C program futásához szükségünk van stack pointer-re (0x00000000-án) és az inicializált adatokat a flash memóriából átmásoljuk a RAM memóriába.

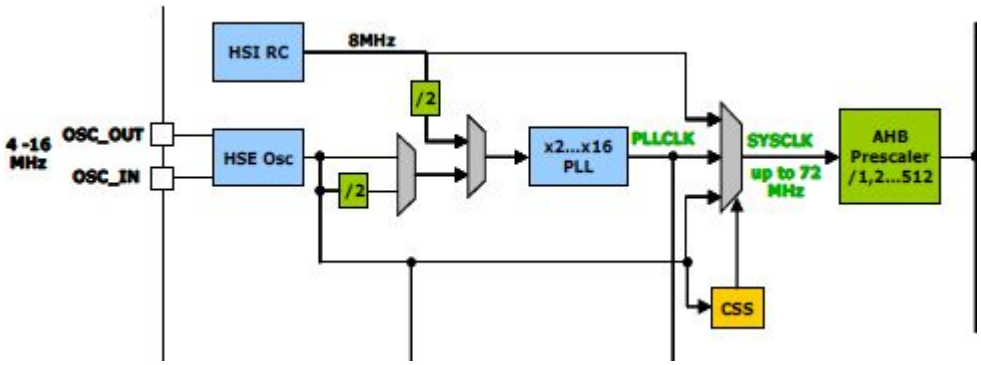
14. Mi az a Flash gyorsító modul, miért szükséges. Mutassa be röviden a hasznát és a működését! Mutasson rá példákat a különböző ARM magú mikrovezérlő generációkból!
A Flash memóriából történő olvasás lassú művelet (25-30 MHz) ami 200 MHz-hez képest nem lég.
Tipikus megoldás, hogy a flash memóriából átmásoljuk a RAM-ba és ott hajtjuk végbe a kódot, de ez az M3-nál nem megfelelő (kevés a SRAM a Flash-hez képest).

Egy célszerű megoldás a flash bit számát megnövelni (32 bit helyett 64 vagy 128 bit) -> NXP fejlesztése, hogy 128 biten lehetett hozzáférni (utána az ST 2x64-bit)

Itt a probléma az ugrásnál jelentkezik, vagy ha olyan dolog (pl. megszakítás) jelentkezik, melyre nem számítottunk. Erre az NXP egyik megoldása 8 darab 128 bites gyorsító modul alkalmazása amely a kisebb ugrásokra megfelelő
ST-ART gyorsító - több 128 bites szavat tárolnak ez mellett arbitráció és branch prediction.

15. Mutassa be egy tipikus ARM Cortex M magú vezérlő órajel hálózatát. Magyarázza meg az egyes órajel osztások értelmét és szükségességét. Milyen előnyökkel és hátrányokkal járnak ezek a beállítási lehetőségek.
A bonyolult architektúra miatt az órajel elosztás sem egy egyszerű feladat.
Több fajta oszcillátorunk lehet mint a Quartz (drága, pontos, lassú stabilizálódás) és RC (pontatlan, olcsó, gyorsabban stabilizálódik). Egy bemenet ahova külső oszcillátor tudunk kötni, és belső RC oszcillátor.
A fő oszcillátor jelét különféle konfigurációkkal rendelkező leosztásainkat rátudjuk rakni a PLL-re. A PLL-ről üzemel a SysClk, Ezt az órajel segítségével üzemeltetünk perifériákat és buszokat.
Mivel a mag több fajta perifériával rendelkezik ezért több fajta órajel leosztást is igényel.

Egy áramkör a probléma esetén képes a PLL bemenetről a belső RC oszcillátor bemenetére kapcsolódni.
A hátránya az összetett felépítésnek, hogy ha pl egy timer-t akarunk beállítani nehéz végig kell követni az előtte levő blokkokat.
A CMSIS lehetőséget nyújt az órajelek magasabb szintű kezelésére.



16. Mutassa be röviden a CMSIS (Cortex Microcontroller Software Interface Standard) bevezetésének szükségességét. Ismertesse a CMSIS core szerepét és az általa nyújtott szolgáltatásokat.
Szoftver fejlesztéssel általában több időt töltünk általában mint hardver fejlesztéssel.
A CMSIS egy szabványos alap definíciós réteg, amely egy szolgáltatást nyújt mely segítségével a belső ismert periféria rétegeket le tudjuk könnyebben kezelni.

- több gyártó támogatja (CMSIS v1-et biztos)
- komplex hardver letakarása

Tartalmaz:
- hardver absztrakciós réteget (HAL)
- regiszter leírást, hogy hogyan kell kezelni a kivételeket
- header file szervezést (mely segítségével csak 1 header filet kell berakni)
- SystemInit() függvény
- Speciális utasítások

Device.h - tartalmazza ami a perifériákhoz kell

core_cm3.h (vagy 0) - regiszter leírás
startup_device - interrupt vektor tábla, kivételek felsorolása
system_device - PLL és az órajelforrások inicializálása

17. Mutassa be röviden a CMSIS v 4 (Cortex Microcontroller Software Interface Standard) szerkezetét és újdonságait. Mutassa be a piacon kapható fejlesztést segítő szoftver tool-okat, és hogy a CMSIS milyen hatással volt ezekre.
Ami maradt az előző CMSIS verziókhoz képest:
- hardver
- CMSIS Core
- v3-as CMSIS DSP library

Nagy változás:
- CMSIS Driver API - alap dolgok konfigurációjának egységesítése (I2C, SPI, UART stb.)
- CMSIS DAP - debug felület és eszközök egyesítése (alap szintű USB és flash driverek

Mindegyik fejlesztő környezet CMSIS-re alapul, de nem vették át (nincs teljes támogatás) feltétlen az összes újítást (pl. a v4-et)

18. Mutassa be egy általános 32 bites mikrovezérlő GPIO lábainak kezelését. Hogyan támogatja ezt a CMSIS, majd a Firmware library-k? Milyen a 8 bites vezérlőkre nem jellemző problémák léphetnek fel a GPIO lábak kezelésénél?
ST esetén egy GPIO kezelése:
- A hozzá tartozó RCC bekapcsolása - csak 32 bitesnél szükséges
- melyik pin
- milyen mód
- felhúzó ellenállás
- sebesség
- irány

} 8 bitesnél is van

A Firmware library magas szintű kezelést biztosít, így nem kell közvetlen a regiszterekbe írnunk.
CMSIS fő célja az egységesítés

Probléma: ha nem kapcsoljuk be a GPIO-hoz tartozó RCC-t akkor nem fog működni.

19. Hasonlítsa össze a Cortex M3 NVIC-ét az ARM7 megszakításkezelési lehetőségeivel. Mutassa be az NVIC interrupt vektor tábla szervezésének főbb jellegzetességeit (nem kell tudni fejből az IT tábla felépítését)! Hány periféria megszakítást támogat az NVIC, mindegyiket ki szokták ezek közül használni?
ARM7
- IRQ - normál IT (interrupt) kezelés
- FIQ - Fast IT
- Vektorok megszakítás gyártó specifikus
- nincs támogatva a Nested IT
- IT kiszolgálás nem volt determinisztikus

M3 NVIC
- gyártó független
- Thumb2 utasításokat ha több órajelig tartottak akkor megszakíthatóak (determinisztikus)
- támogatja a Nested IT-ket (magasabb prioritású IT megszakíthatja az alacsonyabbat - preemptív)
- NVIC processzor független de a gyártó megszabhatja a bemenő vonalainak számát
- 1 nem maszkolható +240 külső periféria + 15 belső IT-t támogat, de nem szoktak 50től többet használni

No.	Exception Type	Priority	Type of Priority	Descriptions
1	Reset	-3 (Highest)	fixed	Reset
2	NMI	-2	fixed	Non-Maskable Interrupt
3	Hard Fault	-1	fixed	Default fault if other hander not implemented

20. Mutassa be az Cortex M3 NVIC prioritás kezelésének alapjait. Mire jók a megszakításmaszk regiszterek, és mi az értelme a Vector Table offset regiszternek? Mutassa be a Cortex M3 NVIC-jének megszakítás végrehajtási folyamatát! Mi az a tail-chaining, mi történik ilyenkor?
- alap belső kivételek -> fix prioritás
- többi külső megszakítás -> prioritás rendszer
- max 8 bites, de min 3 bites prioritási rendszer
- szint korlátozás
- MSB-től kezdődik az implementálás (ha több bites prioritásra áttérek, nem kell a magasabb prioritás újra implementálni, csak 2x annyi lehetőségem lesz)
- sub prioritás: azonos időben azonos prioritással rendelkező IT-nél az alacsonyabb sub prioritású fut le először
- magasabb prioritású IT megszakíthat alacsonyabb rendűt

- MASK-ok - le lehet tiltani vele a megszakításokat
- Vector Table Offset Regiszter - eltudjuk az IT táblát tolni, áthelyezni egy másik helyre - a bootolást segíti

IT kezelés folyamata:
- IT hatására a Cortex M3 It kezelő üzemmódba megy és elmenti a regiszter készletet a stack-re
- eközben az IT kiszolgáló címét elkezdi felhozni az utasítás buszon
- IT kezdő címek után aktualizálás
- IT kiválasztása után 12 órajelel elkezdődik az IT kiszolgálás
- IT után visszatérés szintén 12 órajel ciklus

Tail chaining: A megszakítások láncban történő végrehajtása egyidejű IT esetén

21. Mutasson példát a DMA kezelés használatára! Milyen átviteli és működési lehetőségeket kínál egy általános DMA blokk. Milyen paramétereket kell általában beállítani egy DMA átvitelhez?
DMA (Direct Memory Access)
Perifériák és a memória blokkok hozzáférhetnek a rendszerbuszokhoz a processzor beavatkozása nélkül. Evvel csökkenthetjük a processzor által kiszolgálandó megszakítások számát. A DMA vezérlő Semmilyen feldolgozást nem végez az adatokon.

Működési lehetőségek:
- periféria - periféria
- memória - periféria
- memória - memória

A következő paramétereket kell megadni általában:
- átvitel forrás báziscíme
- cél báziscíme
- átvinni akart blokk hosszának beállítása
- ciklus végéhez tartozó megszakítás jelzés
- Burst-ös vagy egy ciklusos hozzáférés

22. Mutassa be röviden a mikrovezérlőkben alkalmazott DMA-k tulajdonságait: milyen előnyökkel ad a DMA a hagyományos elrendezéshez képest? Milyen beállítási lehetőségei vannak egy tipikus mikrovezérlős DMA-nak? Mi az a Scatter and gather DMA? Hogyan próbálják a DMA hatékonyságát növelni az új sorozatú vezérlőknél (memória elrendezés)? Hogyan függ a DMA sebessége attól, hogy memória – memória, vagy periféria memória átvitelre van szükség?
A DMA (Direct Memory Access) csökkentheti a processzor által kiszolgálandó megszakítások számát továbbá a hardvert is egyszerűsítheti (nem kell minden perifériához önálló FIFO-t rendelni).

Folyamata:
- az adat betöltése a belső címregiszter segítségével
- ez letárolódik a megadott memória vagy periféria címre
- szükséges DMA ciklusok csökkentése
- ki kell kapcsolni a csatornát ha az utolsó ciklus végrehajtódott és nem cirkuláris módba volt konfigurálva

Scatter-Gather DMA - nem szükséges folytatólagos buffer, linkelt lista is lehet

Újabb generációknál megjelenik több master és nagyobb sebességű perifériák is szerepelhetnek master-ként.
Nagyobb hatékonyság miatt a SRAM is több blokkra van szervezve.
A DMA-nál mindegyik stream-hez egy FIFO kerül beépítésre

Memória - memória 5 AHB ütem
Periféria - memória 2 APB + 5 AHB ütem - a legtöbb esetben itt nem elég csak az AHB-n kommunikálni, hanem az APB-n is kell ami lassabb

23. Mutassa be röviden a beágyazott rendszereknél tipikusan alkalmazott szoftver architektúrákat!
Round-Robin
- nincs megszakítás
- nagyon egyszerű
- főciklus végzi az ütemezést

```
void main(void)
{
  while(1)
  {
        if ( Device 1 needs service )
        {
          // Handle Device 1 and its data
        }
        if ( Device 2 needs service )
        {
          // Handle Device 2 and its data
        }
        if ( Device 3 needs service )
        {
          // Handle Device 3 and its data
        }
        ...
```

- Worst case - válaszidő = job-ok össz válaszideje
- Worst case lineárisan nő a job-ok számával
- nagy jitterű válaszidő
- új job felborítja az eddigi időzítést

Round-Robin + megszakítások
- valamennyivel jobban kezeli az időkritikus részeket
- jelentkezhet az osztott változó probléma IT és a főprogram között

```
BOOL Device1_flag = 0;
BOOL Device2_flag = 0;
BOOL Device3_flag = 0;

void interrupt vDevice1(void)
{
  // Handle Device 1 time critical part
  Device1_flag = 1;
}
void interrupt vDevice2(void)
{
  // Handle Device 2 time critical part
  Device1_flag = 2;
}
```

```
void main(void)
{
  while(1)
  {
    if ( Device1_flag )
      {
        // Handle Device 1 and its data
      }
    if (Device2_flag )
      {
        // Handle Device 2 and its data
      }
```

- Esetleges flag-ek helyett használható számláló is

- Worst case válaszidó = a job-ok össz válasza + IT
- Worst case válaszidő lineárisan nő a job-ok számával
- új job felborítja az eddigi időzítést

Függvénysor alapú nem preemptív ütemezés

Olyan függvények használhatóak interrupt-ból és a főprogramból is egyszerre amelyek globális változókat, static kulcsszóval ellátott változókat vagy közös erőforrást használnak
- képes a prioritások kezelésére
- jelentkezhet az osztott változó probléma (IT és főprogram között)

- Worst case válasz idő = a leghosszabb job válaszideje + IT
- Worst Case válaszidő nem nő lineárisan a job-ok számával
- jitter kézben tarthatóbb
- új job nem borítja fel az eddigi időzítést

```
void interrupt vDevice1(void)
{
   // Handle Device 1 time critical part
   // Put Device1_func to call queue
}
void interrupt vDevice2(void)
{
   // Handle Device 2 time critical part
   // Put Device2_func to call queue
}
void main(void)
{
  while(1)
  {
     while(Function queue not empty)
           // Call first from queue
  }
}

void Device1_func (void)
{   // Handle Device 1   }

void Device2_func (void)
{   // Handle Device 2   }
```

RTOS, preemptív ütemezés
- erősen prioritásos
- jelentkezhet az osztott változó probléma (IT, főprogram és taszkok között)
- Worst case válasz idő = a task váltási idő + IT
- Worst case válasz idő nem nő az új job-ok hozzáadásával
- A válaszidő jitter alacsony a magas prioritású szálakra
- jelentős kód overhead

```
void interrupt vDevice1(void)
{
   // Handle Device 1 time critical part
   // Set signal to Device1_task
}
void interrupt vDevice2(void)
{
   // Handle Device 2 time critical part
   // Set signal to Device2_task
}

 void Device1_task (void)
 {
   // Wait for signal to Device1_task
   // Handle Device 1
 }

 void Device2_task (void)
 {
    // Wait for signal to Device2_task
    // Handle Device 2
 }
```

24. Röviden mutassa be a beágyazott operációs rendszerek és az asztali operációs rendszerek közötti különbségeket! Tipikusan hogyan szerveződnek az operációs rendszerek forrás file-jai, mik az alapszolgáltatások, amiket mindegyik tud, és hogyan konfigurálhatóak?
Különbségek:
- Footprint
- Konfigurálhatóság
- Real-time viselkedés
- Nem az OS indítja az alkalmazást hanem az alkalmazás az OS-t
- Nincs memória védelem

Forrás file elrendeződés:
- Demo
- Source:
    -include
    -portable: (portokhoz tartozik, architektúra függő)
        - task váltás
        - systick timer
        - toolchain:
            -port
            -port macro

Szolgáltatások: mailbox, queue, mutex, semaphore, fix méretű memória partíció, idő szolgáltatások, interrupt management

Konfigurálható:
- preemptció
- CPU Hz
- tick rate stb.

25. Röviden mutassa be a FreeRTOS felépítését és jellegzetességeit! Ismertesse a FreeRTOS könyvtárszerkezetét és a portolásánál végrehajtandó lépéseket.
- nyílt forráskódú
- dinamikusan feljődő
- portok: PIC32, PIC32, Atmel, ARM, AVR
- alap szolgáltatások - mutex, semaphore, queue, mailbox, fix méretű partíció, idő szolgáltatások, taszkok (stack-el, prioritással, státusszal rendelkeznek), timer-ek

Portolásának lépései:
- 3 interrupt vektor beállítása
    - Systick: op.rendszer heartbeat timer-a
    - SVC: elindulásánál az első szál elindítása
    - PendSVC: taszkváltás

CMSIS startup részében történő módosítás

26. Tipikusan milyen felhasználási módjai vannak a mutex, semaphore, queue objektumoknak? Mindegyikre adjon példát. Kell e különbséget tenni a kezelésben, ha interrupt-ban használjuk ezeket az objektumokat?
- Mutex: - prioritás inverzió ellen védettek
        - ne használjuk megszakításokból

- Queue - üzenetek küldése taszkok között
- Semaphore - erőforrás menedzsment, ahol egyszerre többen férhetnek egy erőforráshoz
    - bináris - 0/1 érték
    - számláló - nem csak 0/1 érték hanem lehet egész szám
    - esemény számolás

27. A FreeRTOS milyen memóriakezelési lehetőséget nyújt, ezek milyen viszonyban vannak a normál malloc C utasítással és a heap-el? Milyen lehetőségeink vannak a Nagyteljesítményű mikrovezélők FreeRTOS-ben a stack méretének monitorozására? Hogyan tudjuk a FreeRTOS esetében követni az ütemező működését?

Menmang stílus kiválasztása:
- heap_1: csak foglalás nincs felszabadítás
- heap_2: foglalás és felszabadítás
- heap_3: gyakorlatilag malloc, free thread safe-en

Monitorozás: Stack Water mark felkonfigurálásával lehetséges

Ütemezés működésének követését Run Time Stats segítségével követhetjük ahol egy nagy felbontású Timer-re van szükség.

28. Mutassa be, hogy milyen lehetőségeink vannak a mikrovezérlők aktív fogyasztásának befolyásolására 8 bites és 32 bites mikrovezérlők esetében. Hasonlítsa össze nagyvonalakban a 8 bites és 32 bites mikrovezérlők aktív fogyasztásának alakulását az elmúlt pár évben.

Az aktív fogyasztás befolyásolása:
- működési feszültség tartomány (5 V-os logikán megközelítőleg kétszeres a fogyasztás a 3.3 V-os logikához képest)
- lábszám
- flash hozzáfárés és periféria órajelelosztás
- perifériák lekapcsolása
- hőmérséklet (ahogy nő a hőmérséklet nő a fogyasztás is)
- felébredés ideje - aktivitás ideje

A 8 bites mikrovezérlőknek a minimális µA/MHz aránya maradt, a maximális pedig 1/3-ára csökkent (300, 480 µA/MHz).

A 32 biteseknél az alacsony órajelű fogyasztás µA/MHz mértékegységben kb 1/4-re csökkent.
Magas frekvencián pedig 1/3-ára csökkent (valamelyiknél akár 90 µA/MHz-re is)

Aktív fogyasztás - nagyobb órajelnél gyorsabban elvégizi a feladatot és többet aludhat.

29. Mutassa be, hogy egy általános 8 bites és 32 bites mikrovezérlőnek milyen energiatakarékos módjai vannak (általánosan jellemző módok kellenek, nem kell tudni, hogy melyik vezérlőnél hogy hívják ezeket). Milyen újdonságokat tartalmaz a Silabs EFM32 sorozata az energia takarékos

módoknál?
8 bites AVR mikrovezérlő:
1. Csak a CPU órajele van kikapcsolva
2. Perifériák órajele is ki van kapcsolva, és az megszakítások is ki vannak kapcsolva amihez kell órajel
3. System Clk is ki van kapcsolva
4. RTC is ki van kapcsolva
5. Csak pár periféria van bekapcsolva

A 32 bites mikrovezérlők hozzá jön meg egy olyan állapot ahol a SRAM is kivan kapcsolva (elveszti tartalmát).
Passzív üzemmódban a SRAM miatt magas a 32 bites mikrovezérlők fogyasztása.

EFM32 újításai:
- olyan sleep mód ahol csak az aszinkron vagy az alacsony órajelű perifériák futnak
- PRS (Periféria reflex system) - DMA-hoz hasonló, kevesebb fogyasztás. Alvás közben képesek a perifériák egymást "ébreszteni" a CPU bekapcsolása nélkül

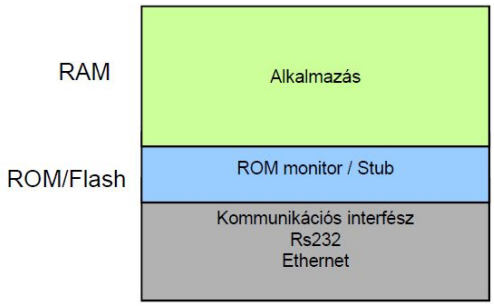
30. Tipikusan miért vannak hátrányban a 32 bites vezérlők a 8 bites társaikhoz képest az energiatakarékos fogyasztás (passzív állapotok) tekintetében. Milyen technikákat, trükköket vetnek be ezek kompenzálására? Milyen előnyei lehetnek egy 32 bites vezérlőnek egy 8 bites vezérlővel szemben energiatakarékos alkalmazásoknál (miért lehet fontos a hatékony utasítás végrehajtás)?
32 bites mikrovezérlők hátránya a 8 bitesek-hez képest passzív fogyasztás szempontjából:
- nagyobb SRAM-al rendelkeznek
- több láb
- több periféria

Megoldások:
- SRAM-ot is ki lehet kapcsolni ha nincs rá szükség
- nagy frekvenciájú perifériákat kikapcsolni
- nem használt lábak kezelése

Előny: Sokkal hatékonyabban dolgoznak és kevesebbet kell, hogy belegyenek kapcsolva (többet bírnak alvó üzemmódba lenni)

31. Mi az a ROM monitor (más néven GDB stub), tradicionálisan hogy és hol lehet használni, miért nem a legjobb megoldás a modern 32 bites mikrovezérlők esetében?
ROM monitor:

- csak a RAM terület használható az alkalmazás által



GDB (GNU Debugger): parancssori debugger, tradícionálisan C kódot debug-oltak vele. (printf)

Hátránya:
- memória probléma
- nem egyszerű a valósidejű debug-olás (stack)

32. Mi az a GDB, milyen jellegű parancsok használhatóak a GDB-ben a target debuggolására? Mi az a GDB RSP, milyen tulajdonságai vannak, miért játszik fontos szerepet a beágyazott rendszerek debuggolásában!

GDB - GNU Debugger

GDB parancsok: run, continue, list, next, break, print set stb.
GDB RSP (Remote Serial Protocol):
- kapcsolattartás a targettel RS232-vel vagy TCP protokollal
- ASCII jelölésekből álló parancsok

GDB parancsok:
- read register
- write register
- read memory
- write memory
- continue

Ezt az elvet a legfejlettebb debuggerek is berakják

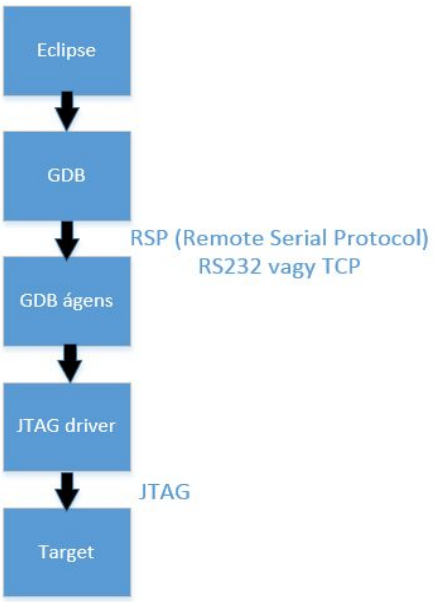
33. Mi az az Open-OCD, milyen főbb komponensekből áll, mire kell odafigyelni, amikor kész Open-OCD forításokat próbálunk használni saját eszközünkhöz?

Open-OCD (Open On Chip Debugger)
- GDB server
- JTAG
- RSP parancsok konverziója JTAG paranccsá

- Külön konfigurációs kód

Fordítás függő részei (konfigurálni kell):
- target modul - minden target core-hoz külön c-file
- flash modul - hardver függő flash támogatás
- portok
- interfészek

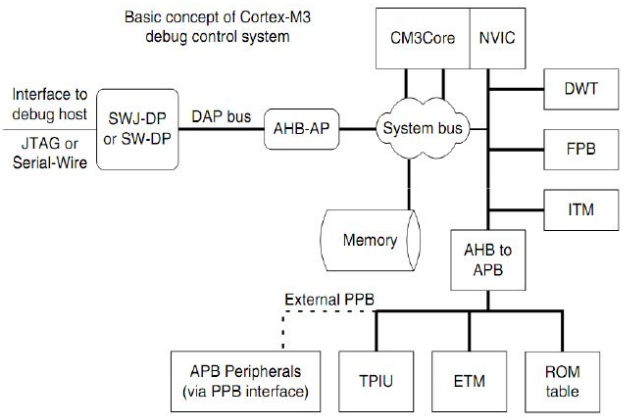
34. Mutassa be nagyvonalakban, hogy milyen lépések játszódnak le akkor, amikor egy Eclipse-es GDB, alapú debugger felületen le szeretnénk egy debug állapotban lévő target egy változójának értékét kérdezni az OpenOCD segítségével (nem kellenek a parancsok, csak az, hogy milyen rétegeken megy át a kommunikáció és azoknak mi a főbb jellemzői)!



Az Eclipse kiadja a parancsot a GDB-nek.
A GDB átalakítja RSP parancsokká
A GDB ágens átalakítja JTAG parancsokká

35. Mutassa be, hogy milyen blokkokból áll a Cortex M3 Coresight debug rendszere, mennyivel nyújtanak ezek a blokkok több lehetőséget egy tradicionális hibakereséshez képest. Mik azok a Trace blokkok, mire képes az ITM, DWT és ETM blokk?

- A debugger megjelenik mint master asz AHB buszon.
- A processzor tudta nélkül hozzá tudunk férni az összes rendszerbuszon elérhető perifériákhoz
- menet közben tetszőlegesen megtudjuk figyelni vagy változtatni a memóriát



- rom táblán keresztül ki lehet olvasni a debug blokkat

2 mód:
- Halt - szokványos debug - minden periféria leáll
- Debug monitor mode - nem áll le az egész rendszer

- ITM - alkalmazás vezérelt trace blokk - Direkt konzol üzenetek (printf)
- DWT- Data Watch Point Trace - 4 komparátor: data address / program counter, számlálók,
- ETM - Embedded Trace Macrocell - Utasítás végrehajtás követés

- trace blokkok

36. Milyen lehetőségei vannak beágyazott rendszerekben a háttértár funkció létrehozására? Röviden mutassa be az SD kártyák típusait és kezelési módjait, illetve a FAT file rendszert, és annak kezelésére alkalmazható szoftver könyvtárak tulajdonságait!

Lehetőségek:
- EEPROM (belső/külső)
- SD kártya

SD kártya típusai: SD, miniSD, uSD

SD kárya osztályai:
- Class2 - 2 Mbyte/sec
- Class4 - 4 Mbyte/sec
- Class6 - 6 Mbyte/sec

Kommunikációs módok:
- One-bit SD mode: különálló parancs és adat csatorna
- Four-bit SD mode: Extra adattáblák
- SPI mode: egyszerűsített kommunikáció, első sorban mikrovezérlők részére

Normal esetben Sector-Block felosztású,
- blokk (hány byte írható/olvasható egyszerre a blokkos adatátvitelnél)
- sector (hány blokk törölhető egyszerre)

FAT (File Allocation Table): A partíciók egyenlő méretű cluster-ekre vannak bontva, a cluster méret függhet az alkalmazott FAT file rendszertől és a partíció méretétől. Általában a 2k és a 32k közötti méreteket preferálják. Minden file méretétől függően egy, vagy több ilyen clustert foglal el.
FAT12, FAT16, FAT32-es file rendszerek léteznek.

ChanFS szoftver könyvtár ami segít kezelni a FAT formátumú SD kártyákat:
- platform független
- ANCI C-ben írodott
- FAT12, 16, 32-t támogatja
- kifejezetten beágyazott rendszerekhez létrehozott FAT file rendszert

Parancsai pl: f_write, f_read

37. Mutassa be egy USB csomag általános felépítését! Milyen főbb csoportokba oszthatók a csomagok, mi ezek jellemzője? Jellemezze az USB négy transzfer típusát! Mikor melyiket érdemes/kell használni? Milyen szabályok vonatkoznak az USB keretek transzferekkel való feltöltésére? Mondjon példákat!

USB (Universal Serial Bus) csomag felépítése:

SYNC	PID	Data	CRC	EOP
Órajel szinkronizáció	Packet ID	PID függő tartalom		End of Packet

Egy tranzakciót csomagokból (packet) épül fel:
- Token: „fejléc”, megadja a tranzakció típusát
- Data: opcionális adat
- Handshake: státusz információ, isochron átvitelnél nincs

Memória segítség:
HaToDa

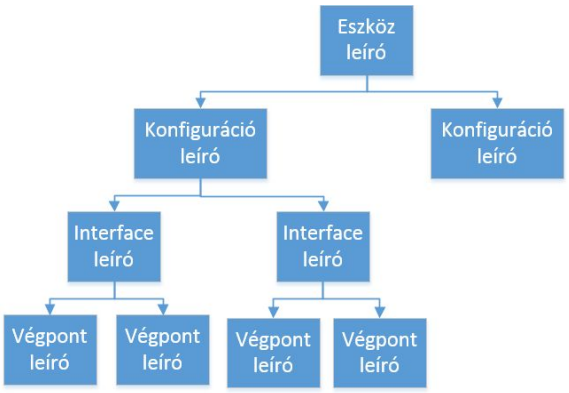
Transzfer típusai:
- Control (vezérlés átvitel) - rövid, host kezdeményezett periféria lekérdezése
- Bulk - nagy tömeg, egy irányú, nem időkritikus - file átvitel
- isochron - állandó sávszélesség - multimédia (webkamera)
- interrupt - megszakításos átvitel, ritkán kevés adatot továbbítunk, megbízható

Keret kitöltése:
- Isochron és interrupt elsőbbségben vannak (keret max 90%-a)
- ha új periféria csatlakozna a és túllépné a 90%-át akkor nem engedi csatlakozni
- fennmaradt 10%-ban elsőbbsége van a Control tranzakciónál
- maradék sávszélesség: bulk

	Sok adat	Hibamentes	Real-time
Bulk	+	+	-
Isochron	+	-	+
Interrupt	-	+	+

38. Mi az az USB pipe és az endpoint, mi a különbség az IN és OUT típusú endpoint között? Milyen viszonyban van az eszköz-, konfiguráció-, interfész- és a végpont leíró? Melyik "mire való"?

USB endpoint: buffer az USB eszközön. A host küldeni vagy fogadni tud a bufferből vagy bufferbe.
Az endpoint iránya a hoszttól függ, IN: eszköz ->host
OUT: host -> eszköz
USB pipe: logikai összeköttetés a host és az endpoint-ok között. Amikor az eszköz küld vagy fogad adatot az endpoint-októl a kliens szerver ezt az adatokat a pipe-on keresztül küldi.



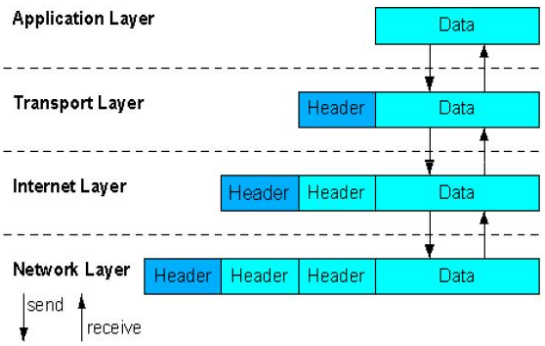
Eszköz leíró: az egész eszköz leírását tartalmazza. Mindegyik eszközhöz egy eszköz leíró rendelhető. A fontos információkat tartalmazza, mint az eszköz USB verziója, max csomag nagysága, vendor és product ID
Konfiguráció leíró: Az specifikálja az eszköz tápforrását, fogyasztását és az interface-ek számát

Interface leíró: leírja a bemenetet/kimenetet, hogy milyen típusú, milyen parancsai vannak
Végpont leíró: Az írja le, hogy milyen adatokat várunk.

39. Tipikusan milyen TCP/IP protokollok szükségesek egy beágyazott megvalósításban? Röviden mutassa be ezeknek a szerepét!

Szükséges protokollok:

- Alkalmazási réteg: a felhasználó programja és a szállítási réteg között reremt kapcsolatot
- Hálózati réteg: szállítási rétegtől kapott header adatokhoz hozzá teszi a saját fejlécét amely megadja hogy az adatot melyik végpont kapja meg
- Adatkapcsolati réteg szintén hozzá rakja a saját fejlécét és az adatokat keretre bontja
- Fizikai réteg továbbítja a kapéott keretet a hálózaton



40. Milyen megvalósítási problémákkal egyszerűsítésekkel találkozhatunk egy beágyazott TCP/IP protokoll stack esetében (IP-, ICMP-, TCP korlátok, memóriakezelés, párhuzamosság)? Kis teljesítményű, kis erőforrású vezérlő esetében miért nem mindegy, hogy TCP, vagy UDP alapú alkalmazási rétegbeli protokollt használunk?

Fő problémák a memóriával vannak.
- IP hossza max 32 kByte (általában 1.5) amit a SRAM-ba kel tudnia tárolni
- 576 Byte-os csomag fogadását minden eszköznek tudni kellene (1 kByte-os memóriánál gond van)
- több csömag fogadására is fel kell tudnunk készülni (8 bitnél van nagy gond)
- sok adminisztráció

BIG ENDIAN (fordítva küldi az adatokat) - swap-olni kell a byte-okat.

TCP: - kapcsolat alapú (megbízható)
- bonyolultabb
- sok memória darabot igényel
- nyugtáig meg kell őrizni a küldőnél az üzenetet
- 4-5x nagyobb az UDP-nél
- nem alkalmazható üzenet szórásra
- állapotgép

UDP:- nem biztos, hogy célba ér az üzenet
- kis erőforrás igényű
- nem kapcsolat alapú