ARM Cortex magú mikrovezérlők

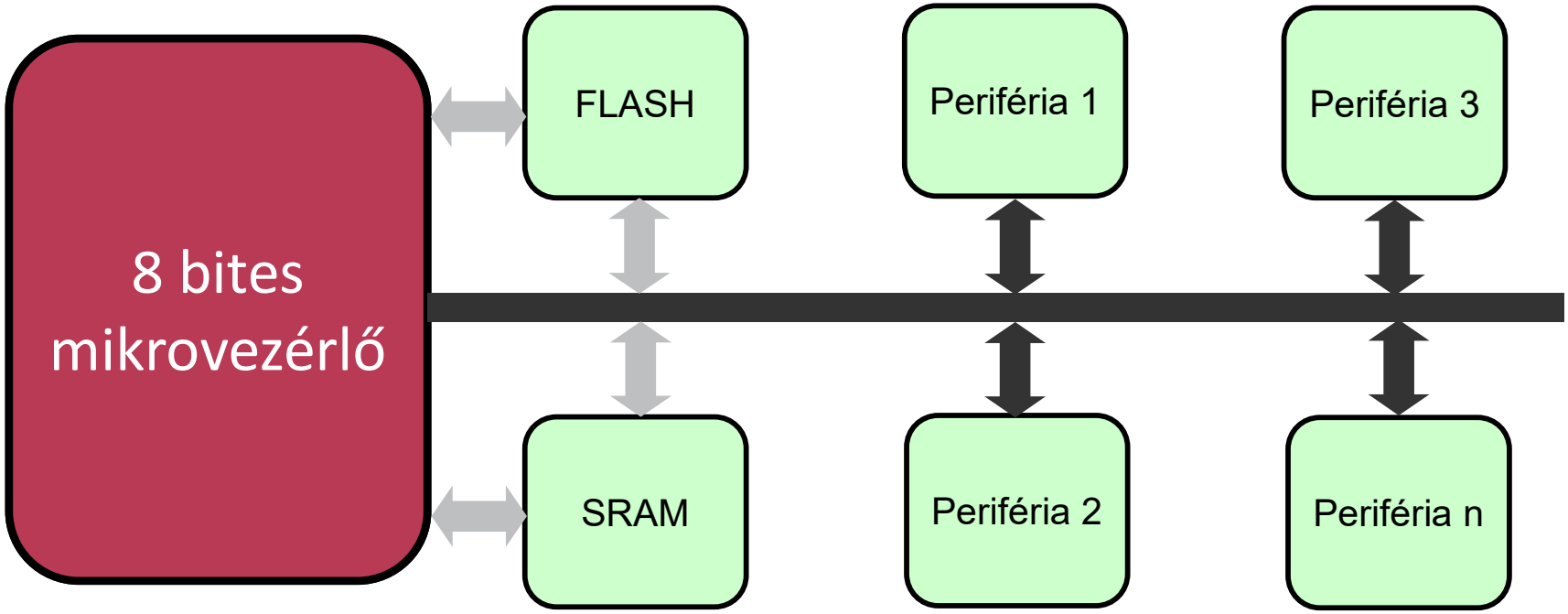
4. Belső felépítés, System Control Block

Scherer Balázs

Méréstechnika és
Információs Rendszerek
Tanszék

Mikrovezérlők belső felépítésének fejlődése

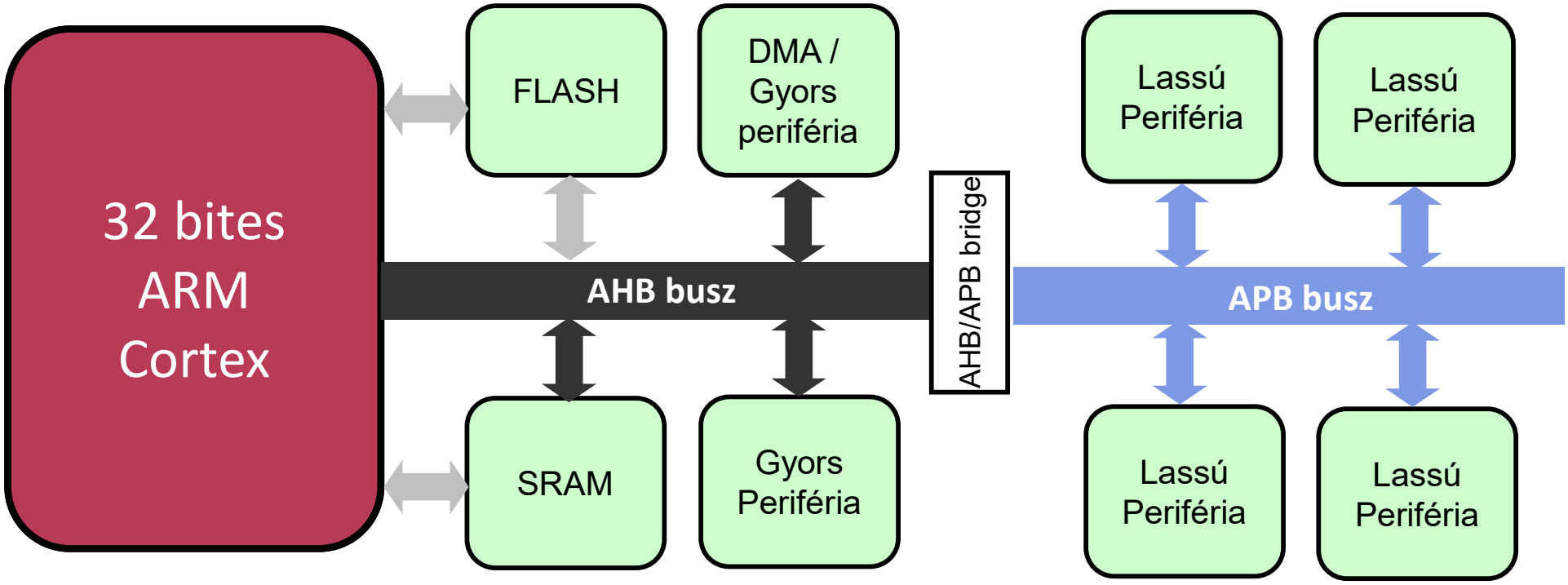
Tipikus 8 bites vezérlők belső felépítése



- Flash, SRAM adatbuszon, vagy gyakrabban direktben csatolva a vezérlőhöz
- Legújabb sorozatoknál 2015+ módosul és kezd hasonlítani a 32-bites vezérlők 2. generációjához

32-bites vezélők

32-bites vezérlők belső felépítése: 1. generáció



- Megjelenik két különböző képességű busz AHB - APB
- Régebbi generáció tagjainál jellemző (2010 elött), utána csak a legegyszerűbb vezérlőknél
  - Kis lábszám, nem komplex perifériák, és 50 MHz alatti sebesség, elsősorban M0 magú vezérlők

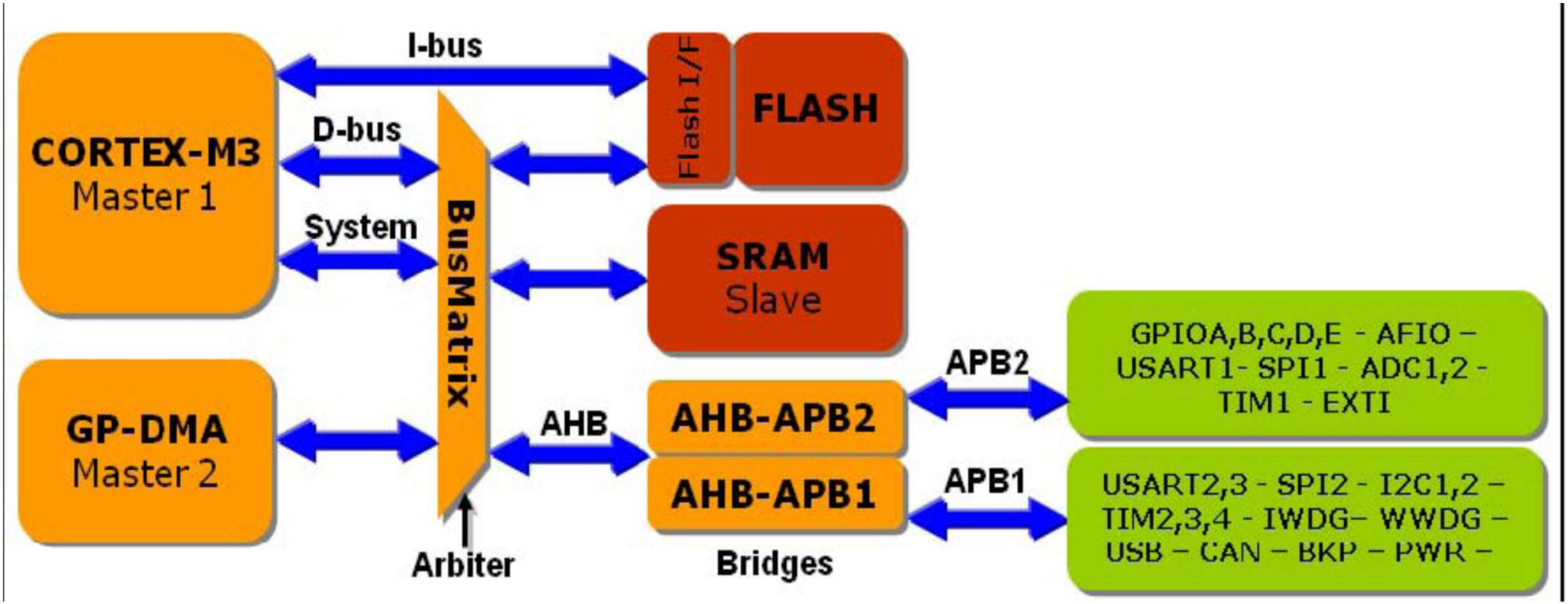
AHB vs APB

- Mindkettő az ARM Advanced Microcontroller Bus Architecture (AMBA) szabványhoz tartozik.

- **AHB** Advanced High-performance Bus
  - Van Pipelining
  - Multiple master
  - Burst-ös átvitel
  - Full-duplex parallel komm.

- **APB** Advanced Peripheral Bus
  - Nincs Pipelining
  - Single master
  - Kis interfész-komplexitás
  - Kis fogyasztás
  - 32 bites buszszélesség

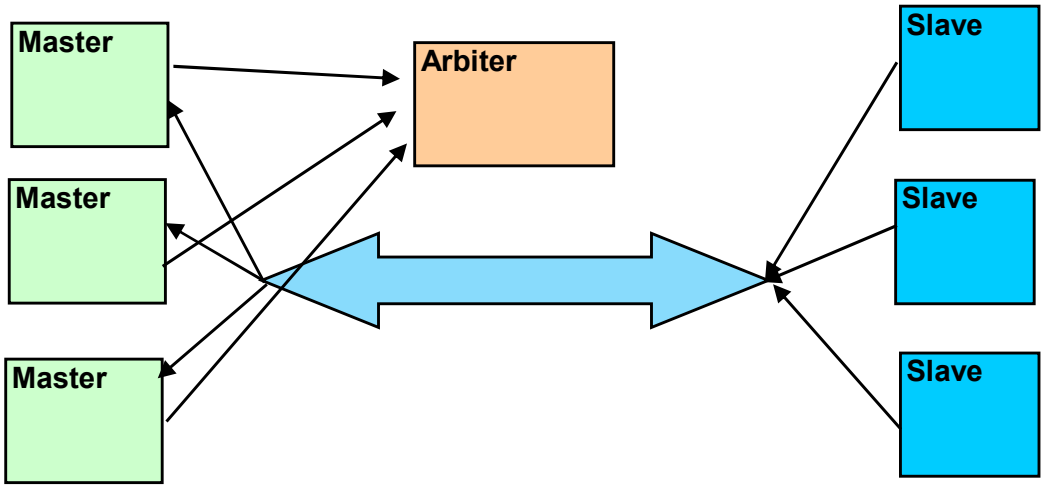
32-bites vezérlők belső felépítése: 2. generáció



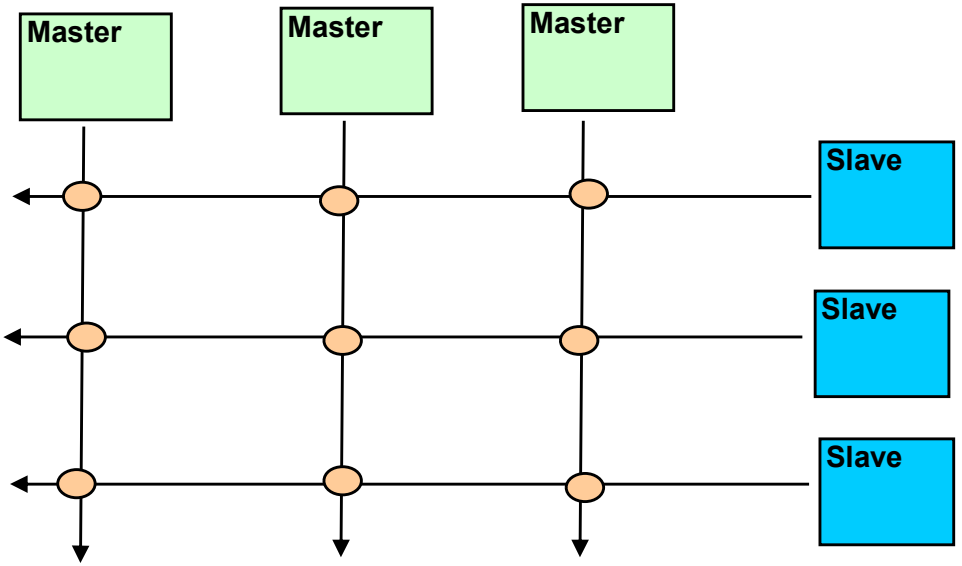
- Busz mátrix
- Különböző sebességű periféria buszok
  - Perifériák sebesség szerint rendezve
- Lényegében eltűnik a 2.1 generáció miatt

Több master a buszon

Megosztott AHB Busz

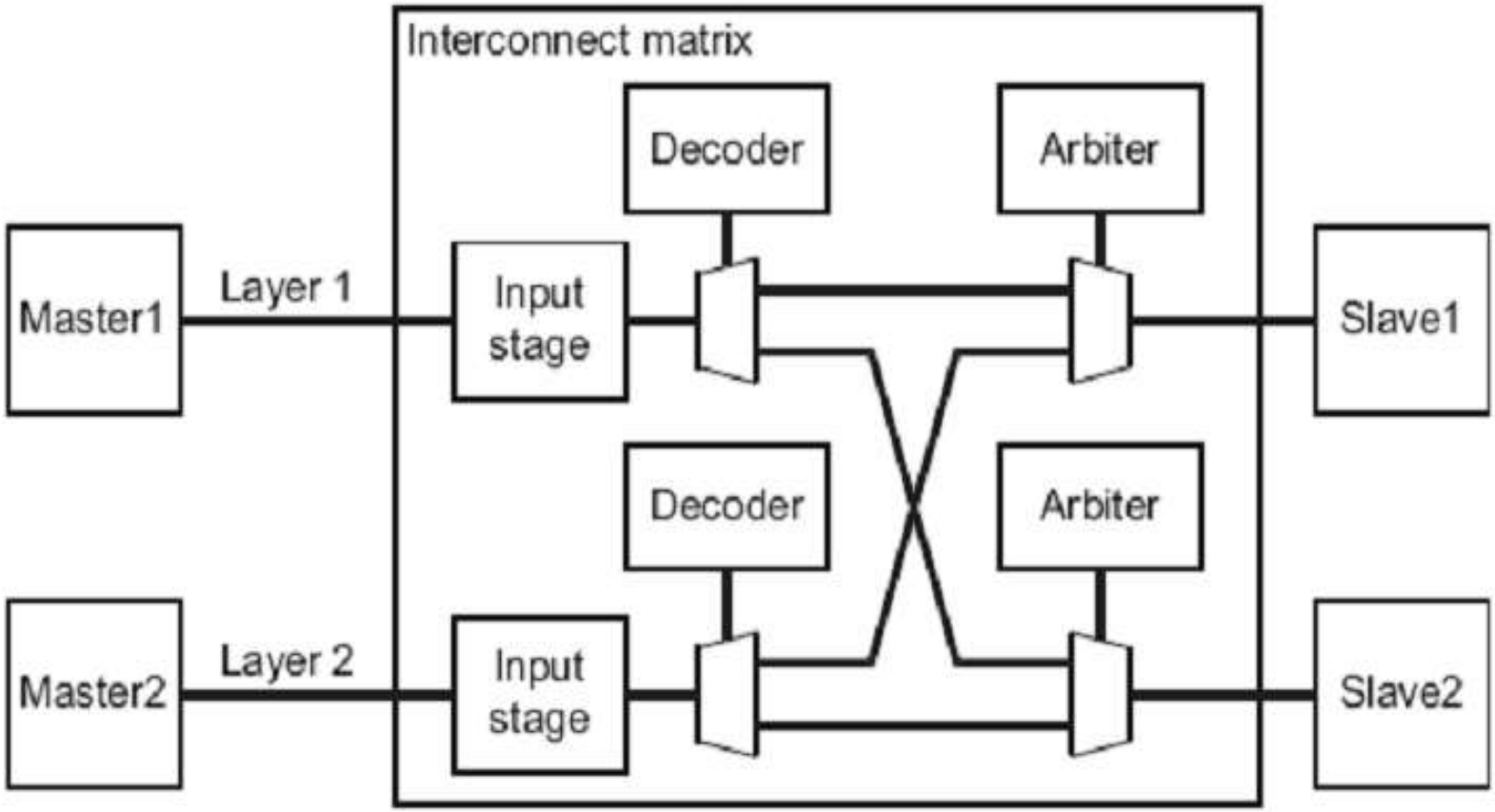


AHB Busz Matrix



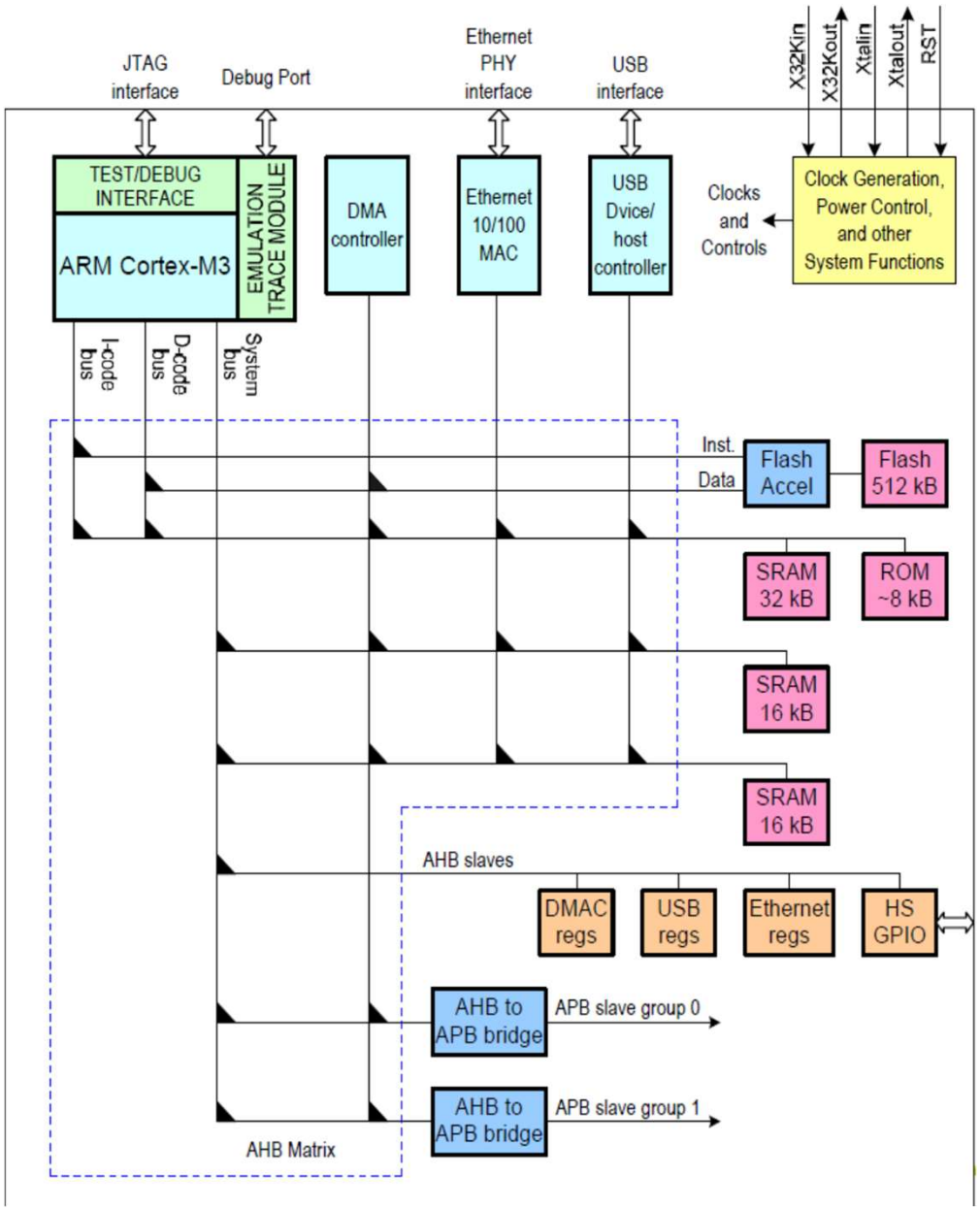
Mi történik a pontoknál?

- Arbitráció: általában round-robin



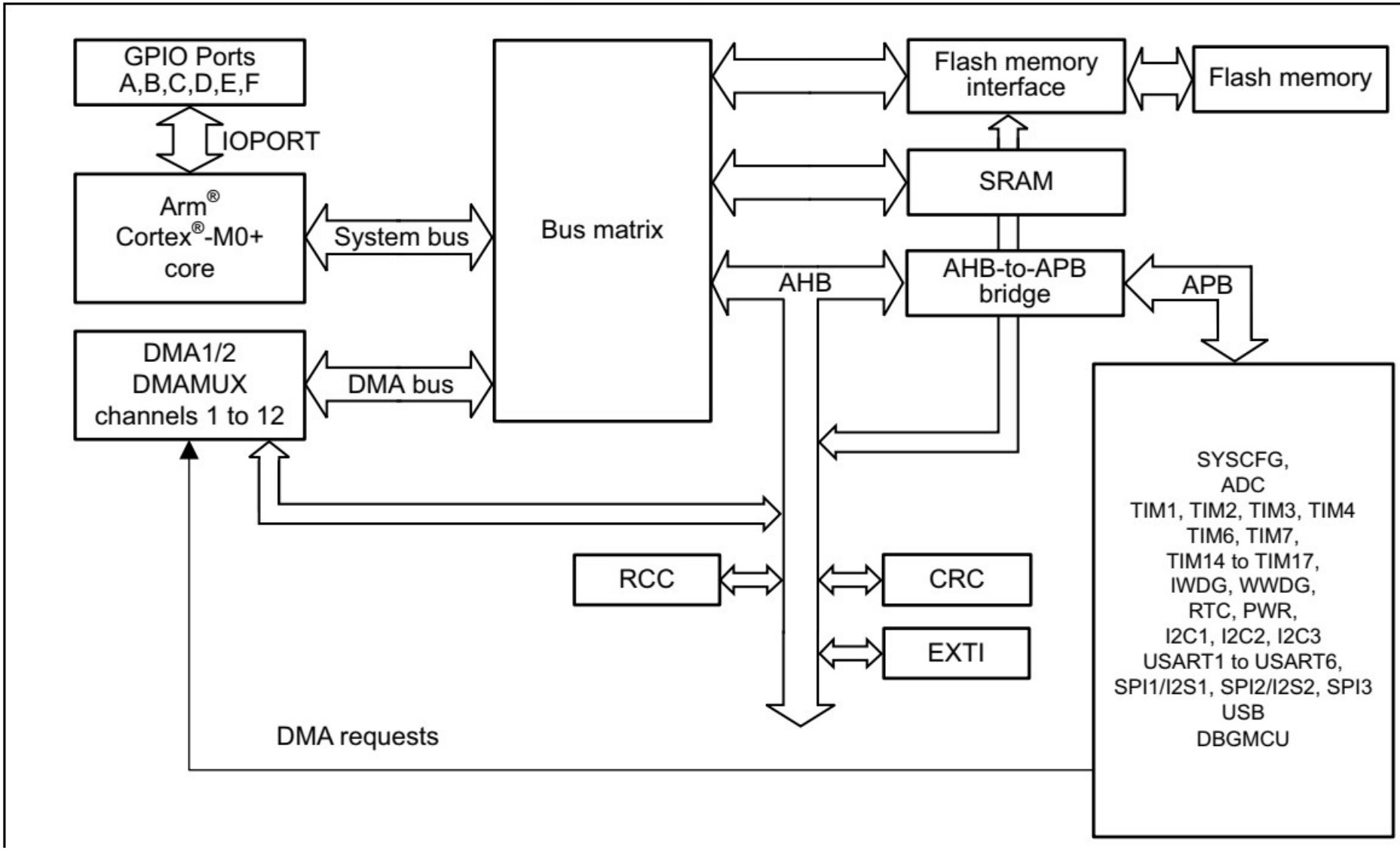
32-bites vezérlők belső felépítése: 2.1 generáció

- 2009 után
- Busz mátrix
- Több részre osztott adatmemória Különálló, külön elérhető memória blokkok
- A legtöbb jelenleg kapható vezérlő architektúra ezen alapul

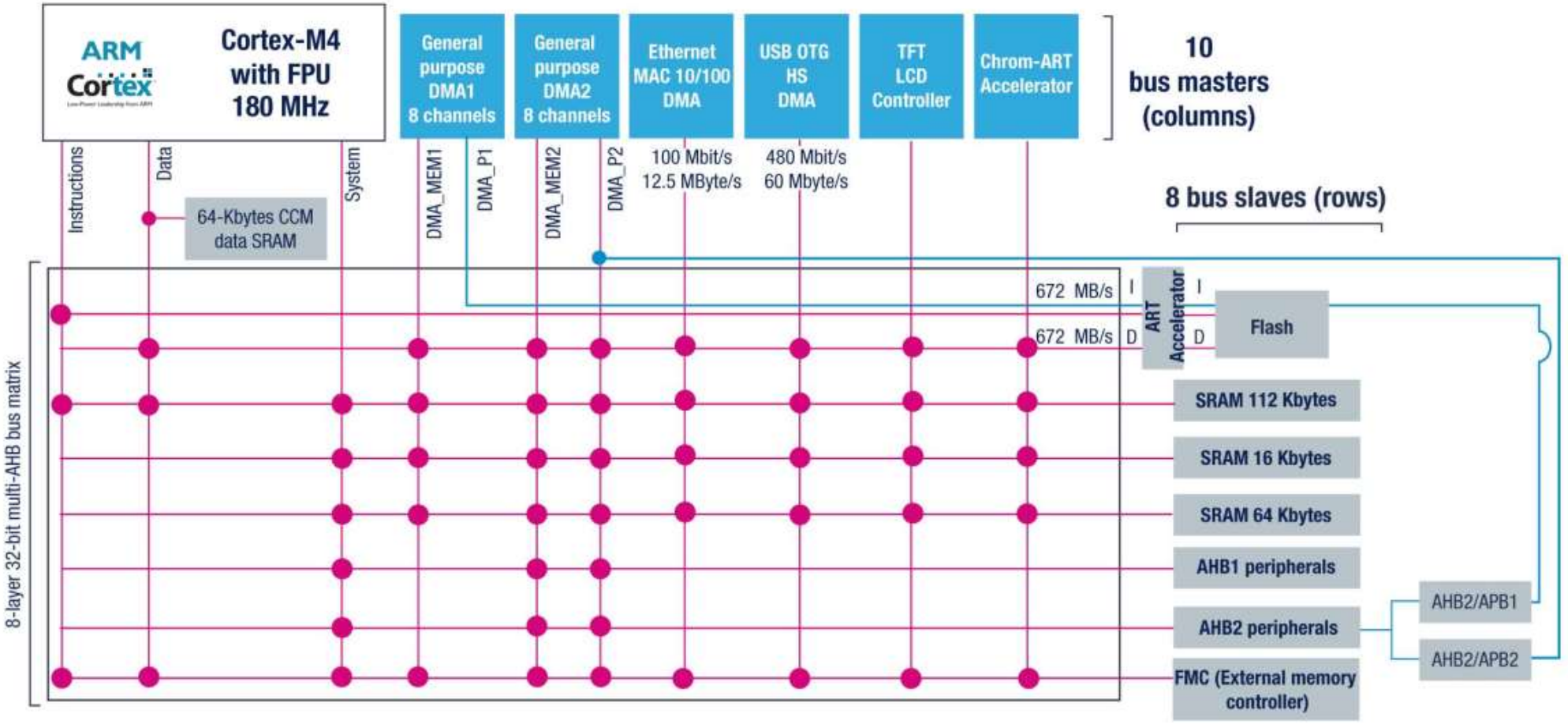


M0+ példa: STM32G0x0 sorozat

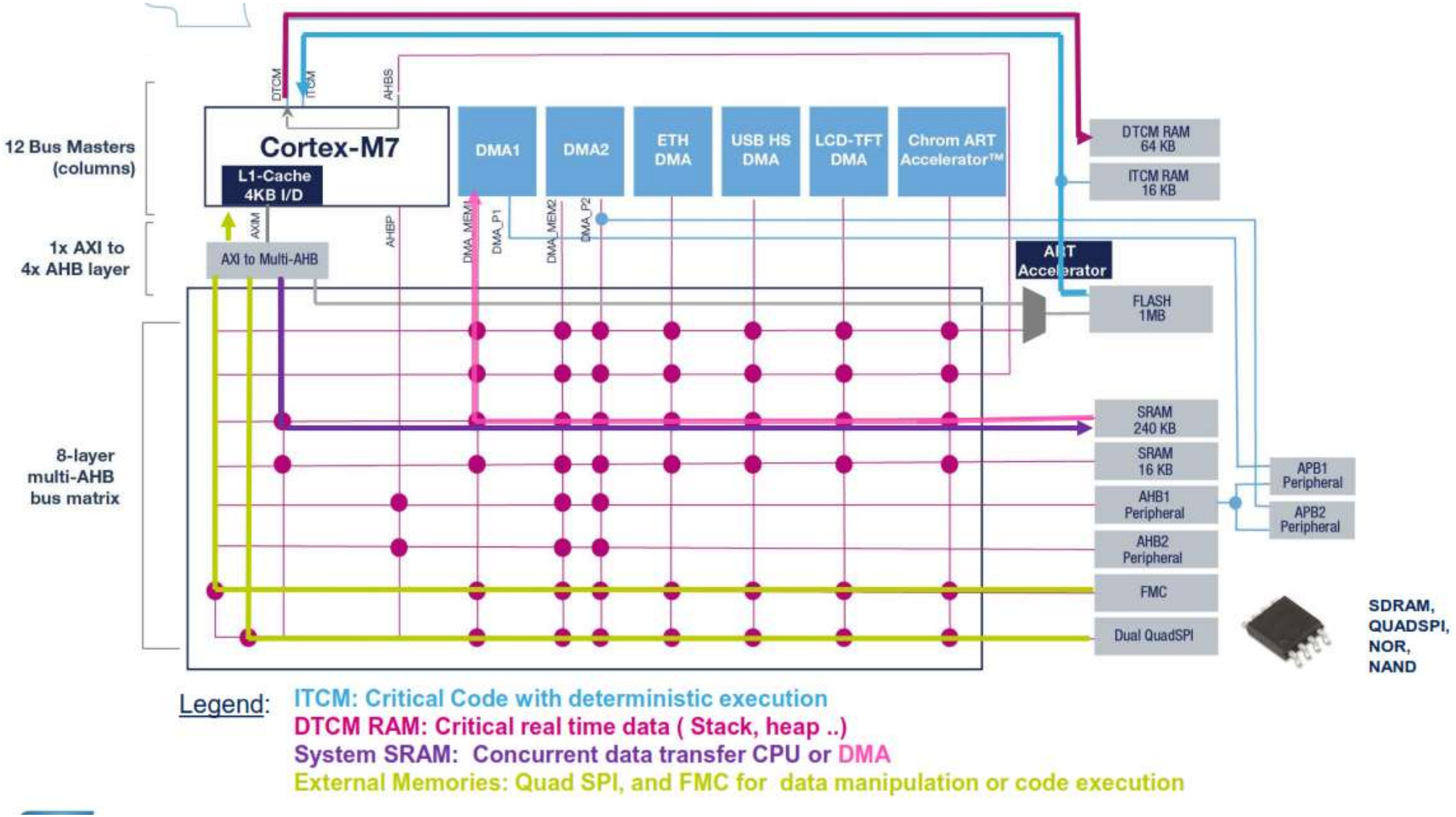
Figure 1. System architecture



M4 példa: STMF4xxx



M7 példa: STM32F7xx belső felépítése



Legend:
ITCM: Critical Code with deterministic execution
DTCM RAM: Critical real time data (Stack, heap ..)
System SRAM: Concurrent data transfer CPU or DMA
External Memories: Quad SPI, and FMC for data manipulation or code execution

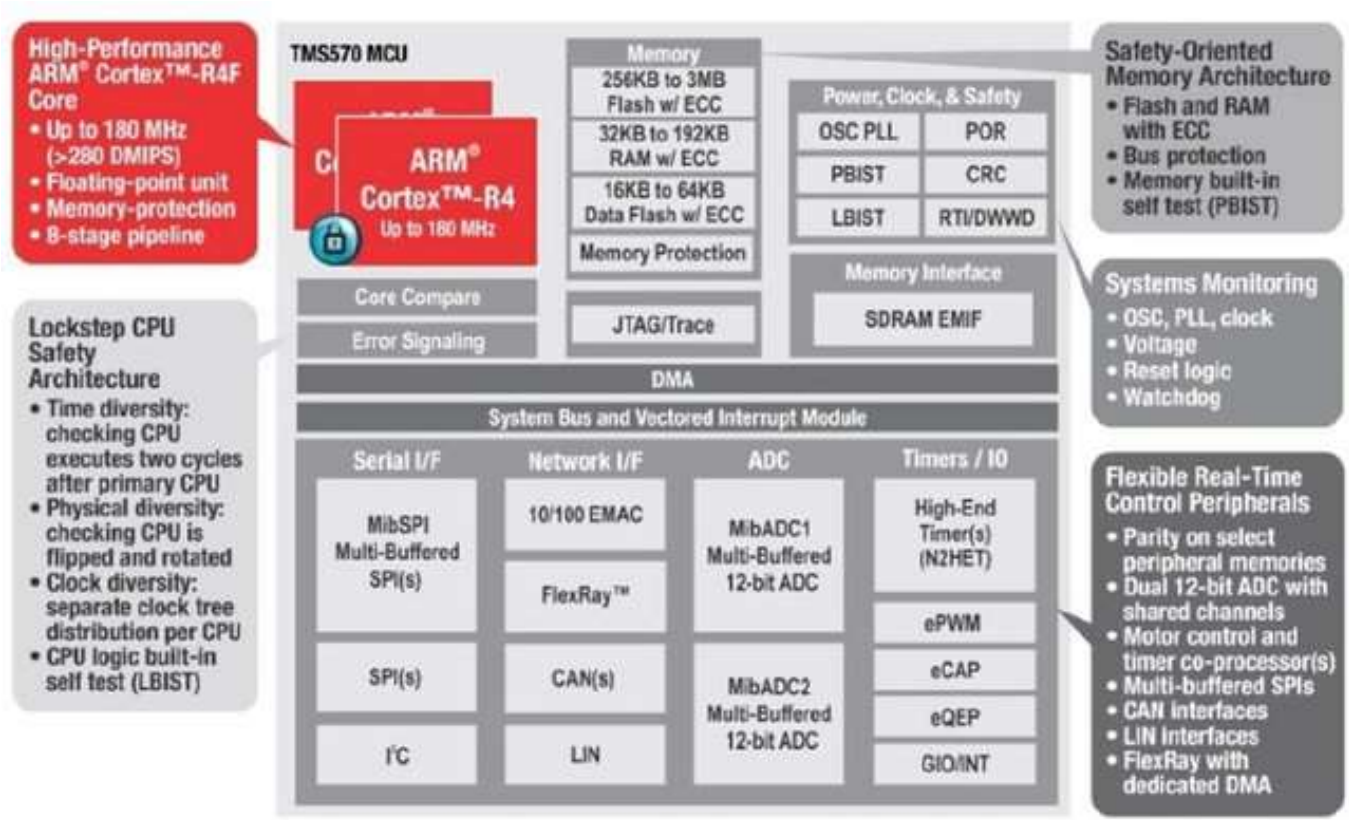
Több magú vezérlők

Több magú mikrovezélők

- Nem az a jellemző, mint az asztali PC-knél
  - Több azonos mag
  - Operációs rendszer szétosztja a feladatokat a magok között
- Itt nincs operációs rendszer ami ezt meg tudná tenni
- Két jellemző alkalmazása van a több magnak
  - Safety célú: Dual lock step processzorok
  - Általános heterogén több magú vezérlők

Safety célú vezérlők

- Jellemzően Dual lock step processzorok
  - Ugyanolyan processzor magok
  - Ugyanaz a program hajtódig rajtuk végre
  - Az eredmény összehasonlításra kerül, és ha nem egyezik meg akkor hibajelzés: véletlen hardware hibáktól véd
  - Más safety megoldással is el vannak látva



Általános célú több magú vezérlők

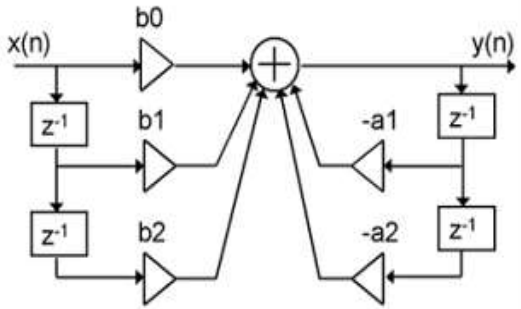
- Jellemzően heterogén
  - Egy M7 egy M4 vagy egy M4 és egy M0 mag
- A két vagy több processzor külön feladatot lát el
- Külön program fut rajtuk: külön project tartozik hozzájuk a fejlesztőkörnyezetben
- A buszmátrixhoz mindegyik vezérlő hozzáfér
- A program és adatmemória több részre van osztva a chippen belül
  - Dedikált területek
  - Közös területek

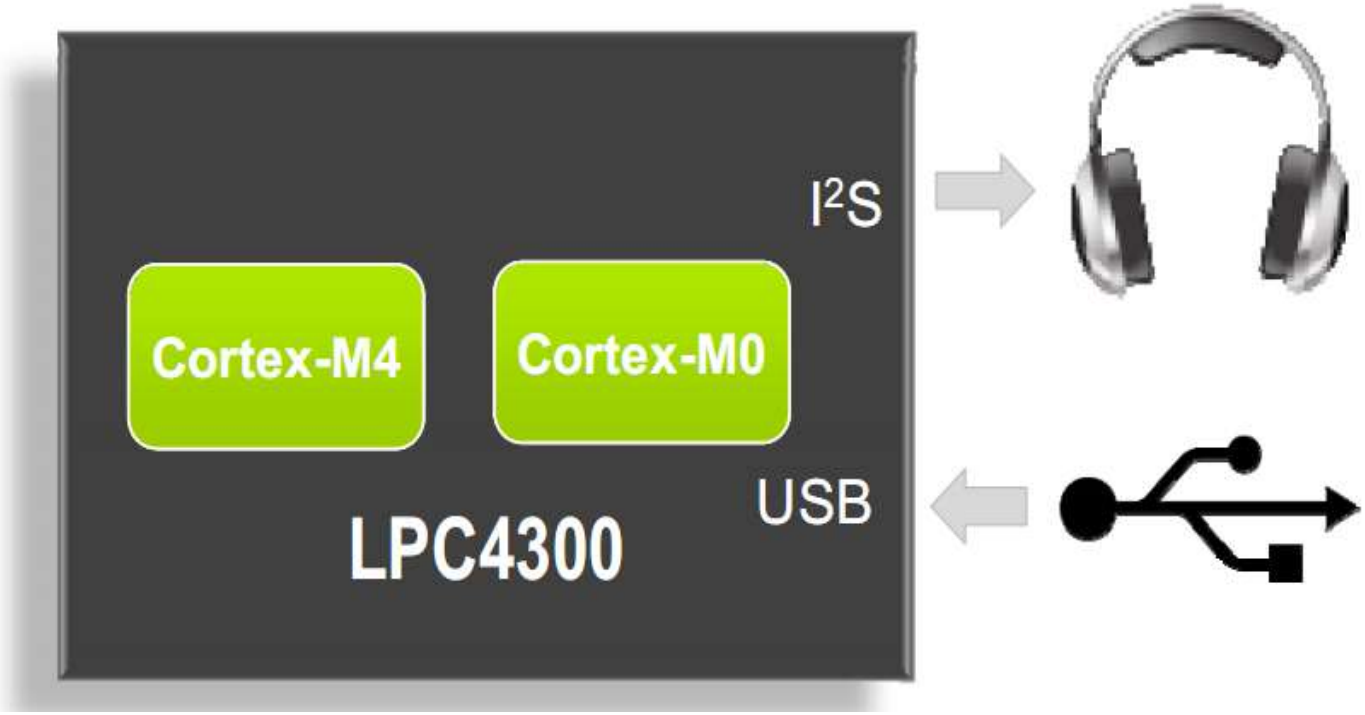
Példa: az LPC4300 család

- Cortex-M4 alapú Digital Signal Controller
- Cortex-M0 alrendszer a perifériafunkciókra
- max. 1 MB Flash
  - Két bankos Flash
- Max. 200 kbyte SRAM
- High speed USB
- Pin kompatibilis az M3 sorozattal
- További tulasdonságok
  - 10/100 Ethernet MAC
  - LCD panel controller (max. 1024H × 768V)
  - 2x 10-bit ADCs és 10-bit DAC at 400ksps
  - 8 csatornás DMA vezérlő
  - Motor Control PWM, Quadrature Encoder
  - 4x UARTs, 2x I2C, I2S, CAN 2.0B, 2x SSP/SPI

Cortex-M0, M4 együttes használata; példa: audiofeldolgozás

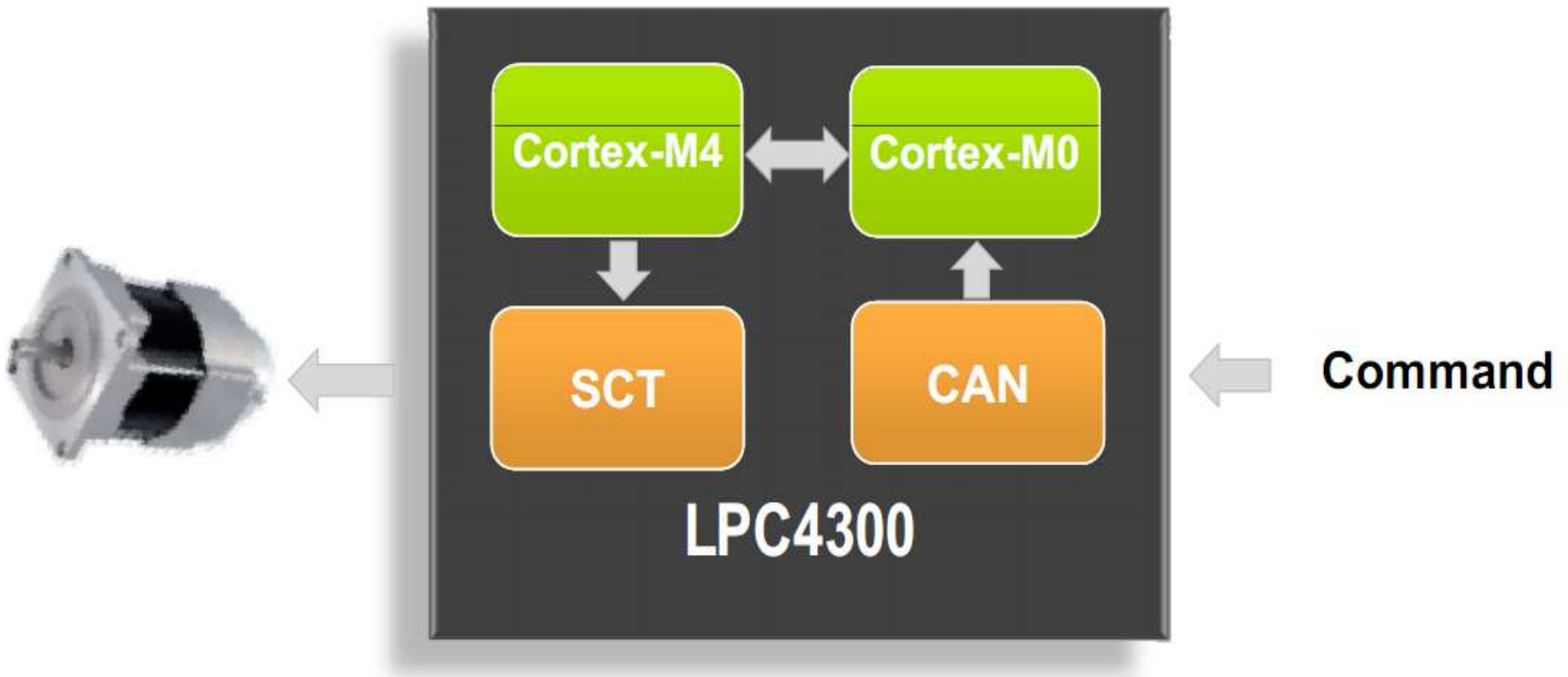
- Cortex-M0: perifériakezelés: I2S, USB
- Cortex-M4: jelfeldolgozás





Motorvezérlés példa

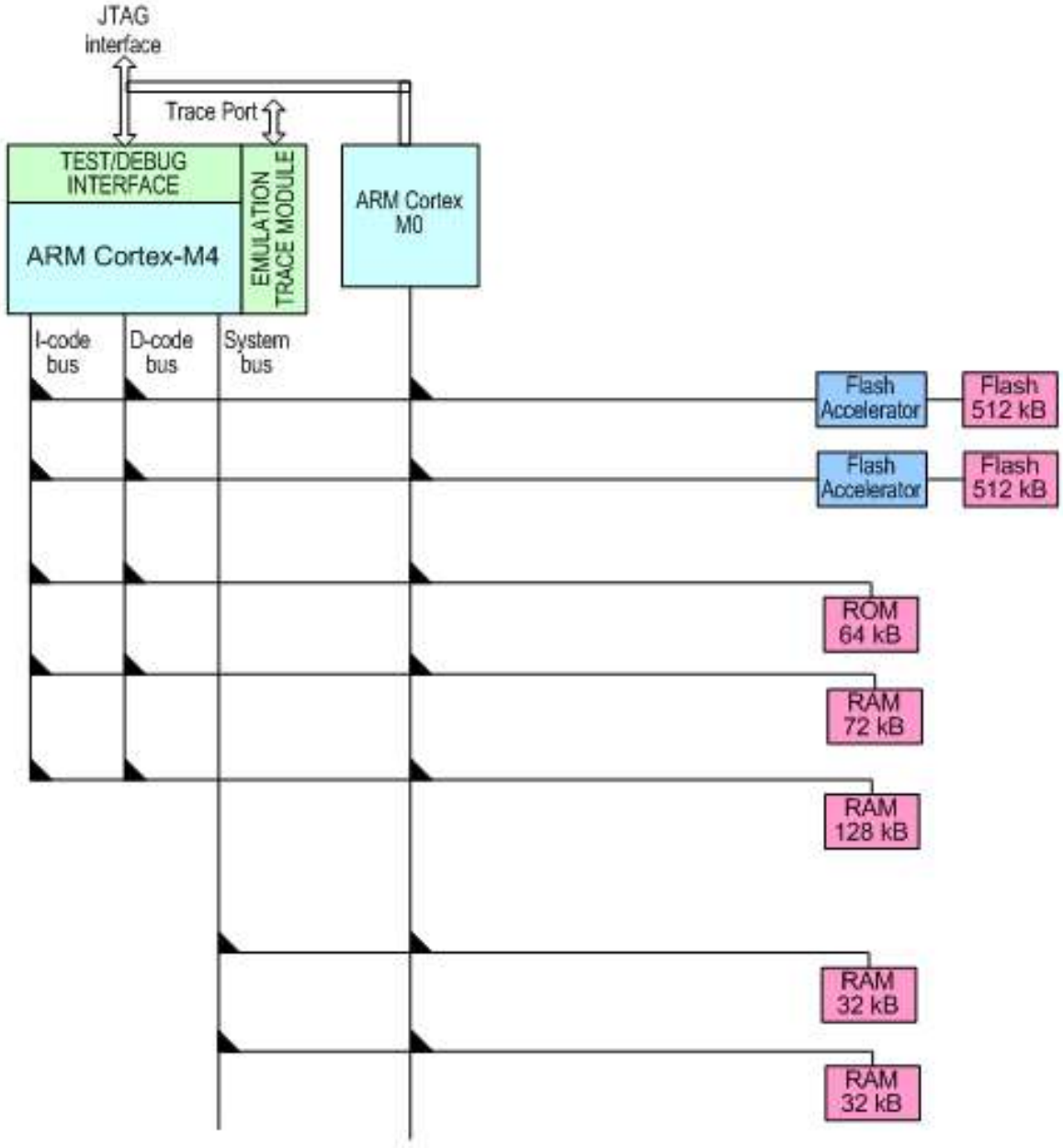
- Cortex-M4: motor control: Field Oriented Control (FOC)
- Cortex-M0: CAN parancsok feldolgozása



LPC4300 belső felépítése

LPC43xx memóriarendszere

- Dual Core
  - Az M4 és M0 is tud Flash-ből kódot végrehajtani összeakadás nélkül
  - M0 a saját RAM-jából futtatni
  - ROM code Thumban van, mindkettő tudja futtatni
  - M4 MPU-ja tudja védeni az M0 által használt memóriákat



System Control Block

System Control Block

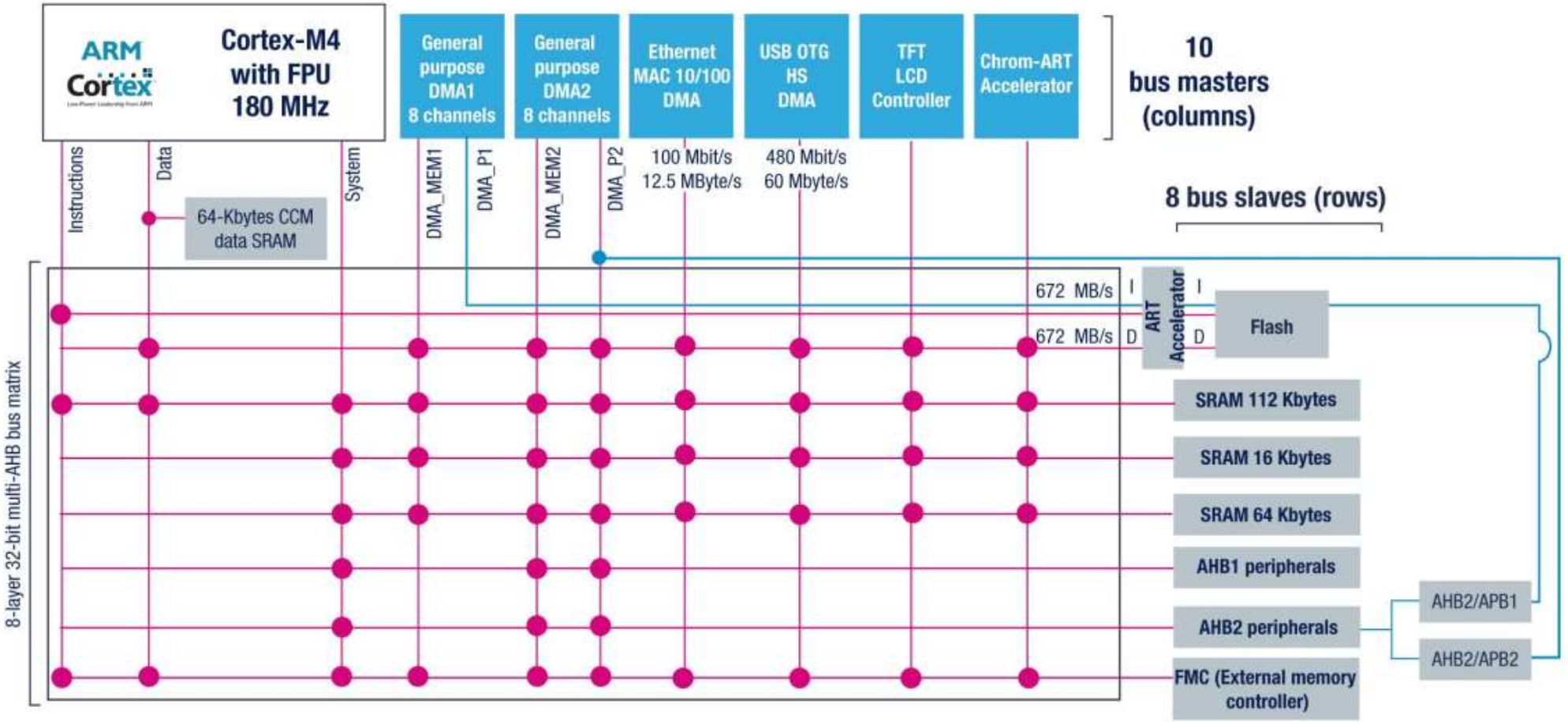
- Órajel rendszer
  - A rendszerórajel forrásának kiválasztása
  - PLL (Phase Locked Loop)
  - Egyes perifériabuszok órajelének meghatározása
- A Flash-hozzáférés szabályozása
  - Flash-gyorsítás, wait-ciklusok száma stb.
- Perifériák tápellátásának engedélyezése
  - A modern vezérlőkben gyakorlatilag periféria szinten kapcsolhatóak le az egységek.
- Lábak alternatív funkcióinak meghatározása

Órajel rendszer

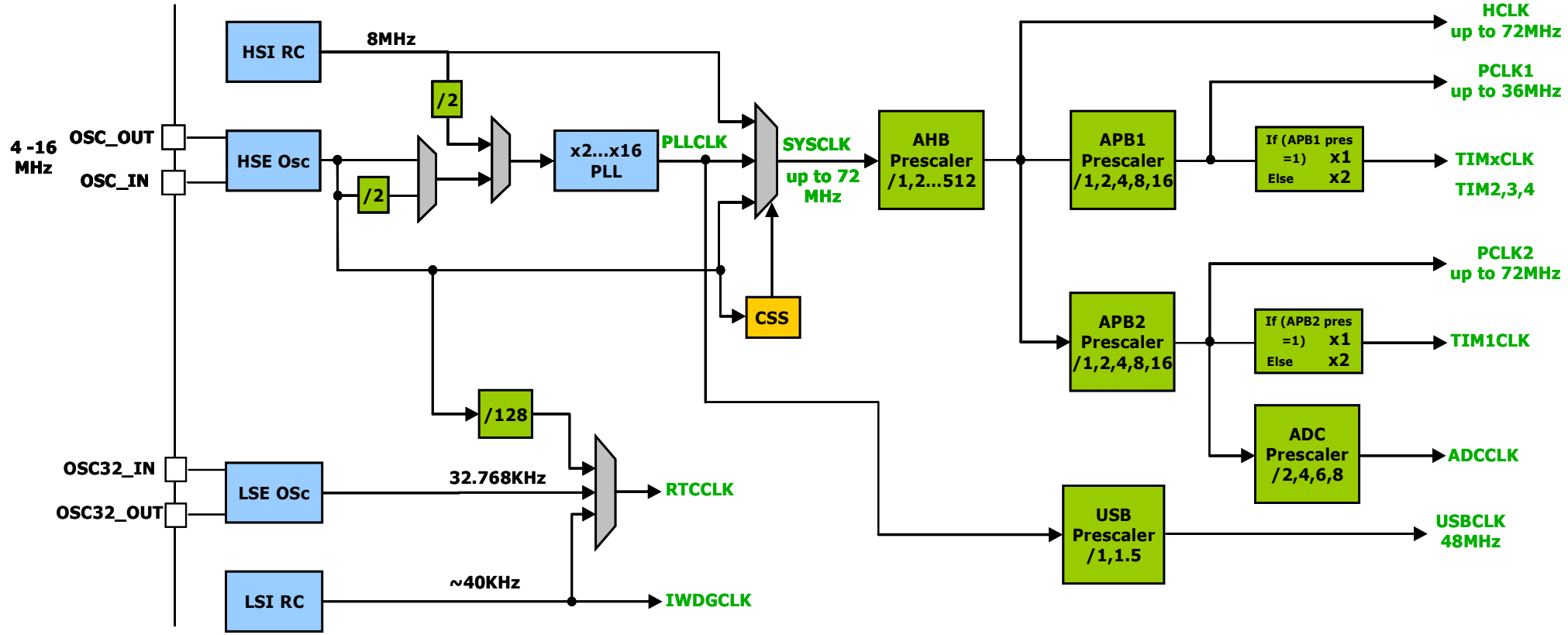
Az órajel rendszer alap problémái

- Követelmény: sokféle alapórajel
  - Kvarc: pontos, stabil, drága
  - RC oszcillátor: pontatlan, olcsó
  - Alacsony sebességű órajel forrás: RTC kvarc, Low speed RC
- Sokféle perifériaigény
  - Alap perifériablokkok
  - I/O lábak
  - Ethernet
  - USB

Kapcsolat a busz architekrúrával



Egyszerűsített tipikus órajel hálózat



Órajelszétosztás problémái

- Egy egyszerű LED villogtatás is minimum 1 órajel engedélyezését igényli, de akár 3-4-et is igényelhet.

7.3.13 RCC APB1 peripheral clock enable register (RCC_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	Reser-ved	CAN2 EN	CAN1 EN	Reser-ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser-ved
		rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWDG EN	Reserved		TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw			rw	rw	rw	rw	rw	rw	rw	rw	rw

Flash-gyorsító modul

Flash-memória

- A Flash-memóriák kisebb helyet foglalnak el, mint a RAM-ok, de lassabbak.
- Flash-memória hozzáférési idők
  - 30 ns – 50 ns (33 – 25 MHz)
- Ez kevés a 60, 72, 120, 180, 200,300 MHz-ről való működéshez.

Flash-memória

- A Flash-memóriák kisebb helyet foglalnak el, mint a RAM-ok, de lassabbak.
- Flash-memória hozzáférési idők
  - 30 ns – 50 ns (33 – 25 MHz)
- Ez kevés a 60, 72, 120, 180, 200,300 MHz-ről való működéshez.
- Tradítcionális megoldás: Futtassunk a kódot a RAM-ból!
  - Mikrovezérlőknél nem működik
  - A RAM drága és sokat fogyaszt
  - Tipikusan 5-ször, 10 szer több Flash vagy egy mikrovezérlőben mint RAM
  - A RAM-ba még adatokat is kellene tárolni nem csak a programot

Első generációs megoldások 2000 - 2010

- Megnövelt bitszámú Flash hozzáférés
  - 64,128 bites Flash interfész

- A wait ciklusok számát fel kell programozni.

zero wait state, if 0 < SYSCLK ≤ 24 MHz
one wait state, if 24 MHz < SYSCLK ≤ 48 MHz
two wait states, if 48 MHz < SYSCLK ≤ 72 MHz

Első generációs megoldások 2000 - 2010

- Megnövelt bitszámú Flash hozzáférés
  - 64,128 bites Flash interfész

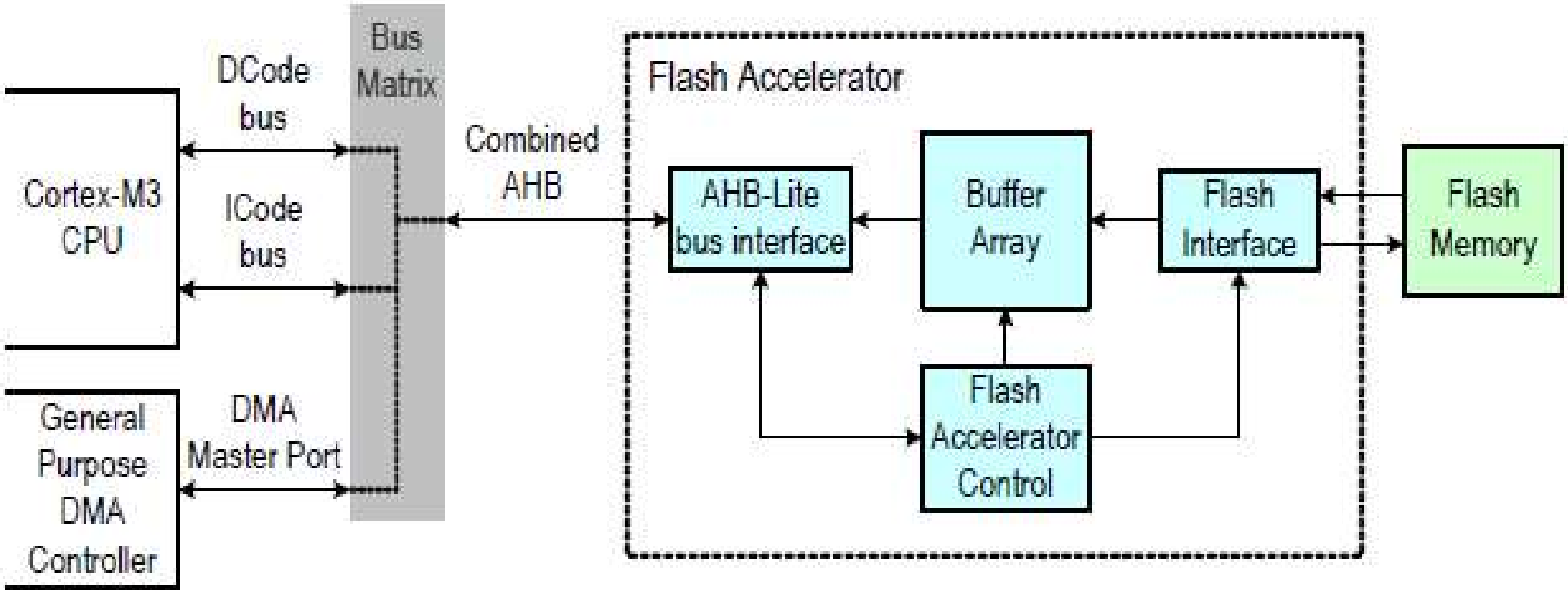
- A wait ciklusok számát fel kell programozni

zero wait state, if 0 < SYSCLK ≤ 24 MHz
one wait state, if 24 MHz < SYSCLK ≤ 48 MHz
two wait states, if 48 MHz < SYSCLK ≤ 72 MHz

- Problémája, hogy 50MHz felett erősen lassul

Második generációs megoldás: 2010 után

- Kis gyorsító tár: 8 darab 128 bites szó
- Nem csak utasításokat, hanem adatokat is fel tud hozni.
- Wait-ciklusok számát itt is fel kell programozni



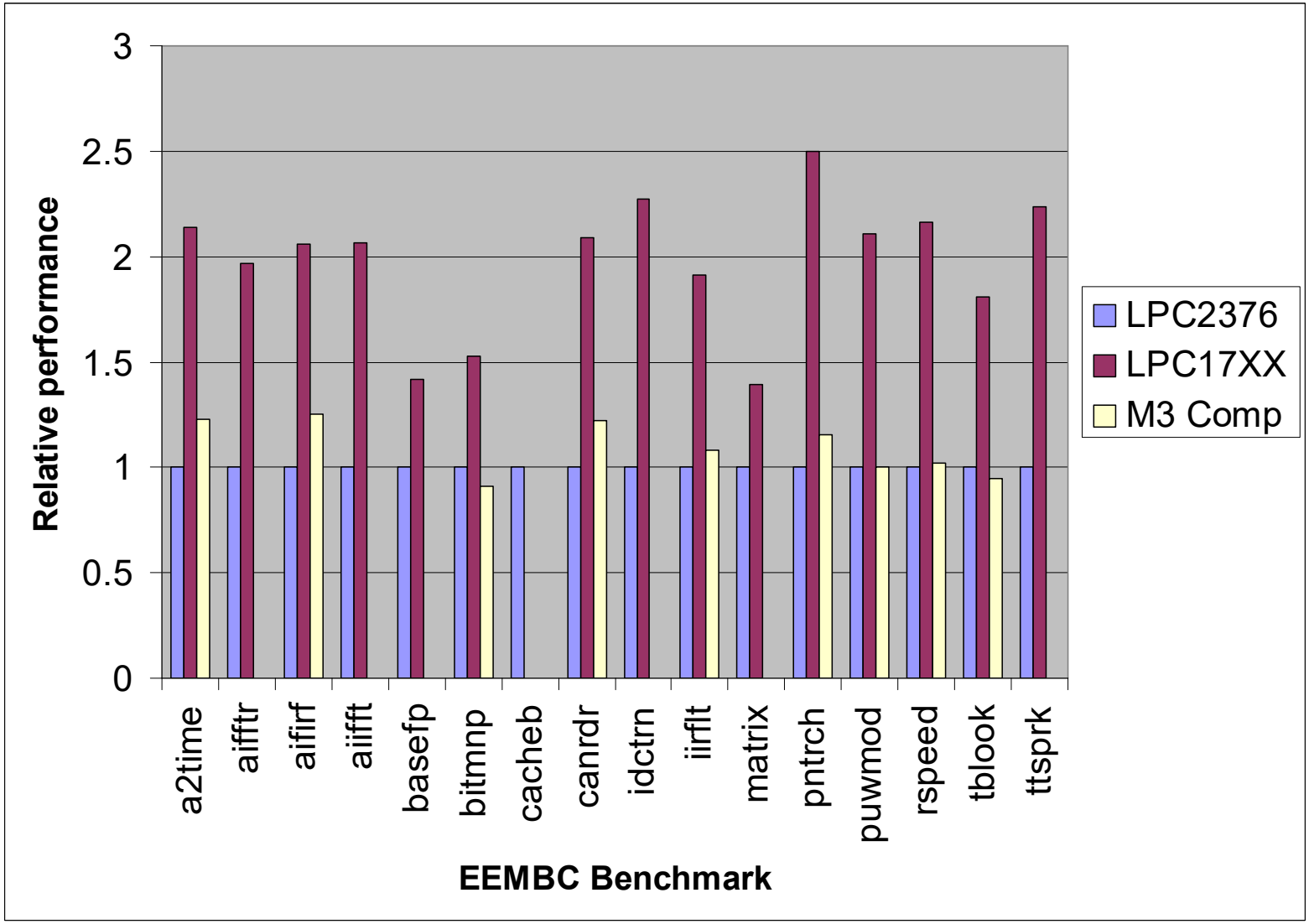
A kis gyorsítótár hatása

- RAM-ból való kódvégrehajtással összevetve
  - A végrehajtási sebesség 16% -on belül marad.
  - A fogyasztás 25%-al kevesebb.

- A régi sima 128 bites Flash interfészhez képest 45% teljesítménynövekedés

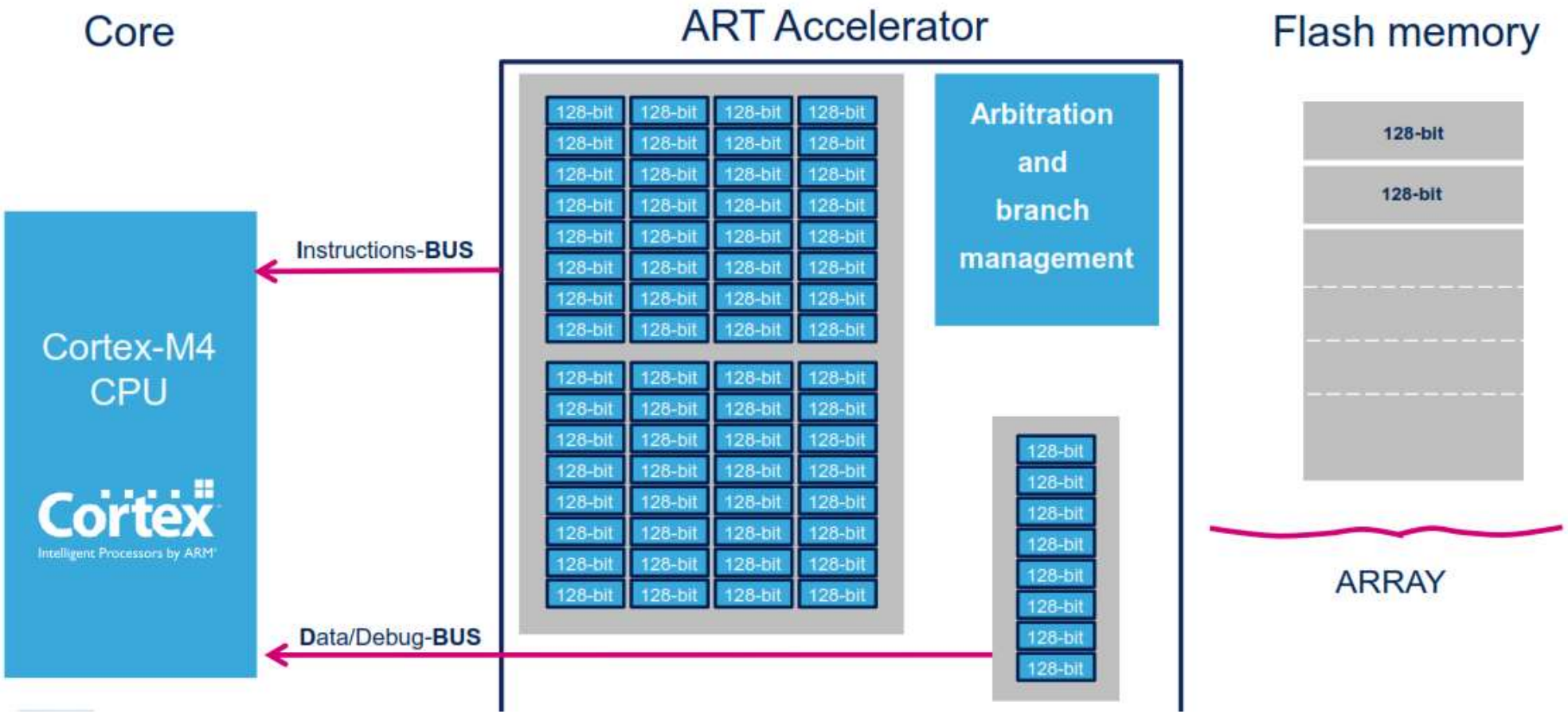
Benchmark eredmények

- LPC2376, M3 Comp: első generáció csak Flash bitszám növelés, LPC17xx-nál már kisméretű gyorsító tár

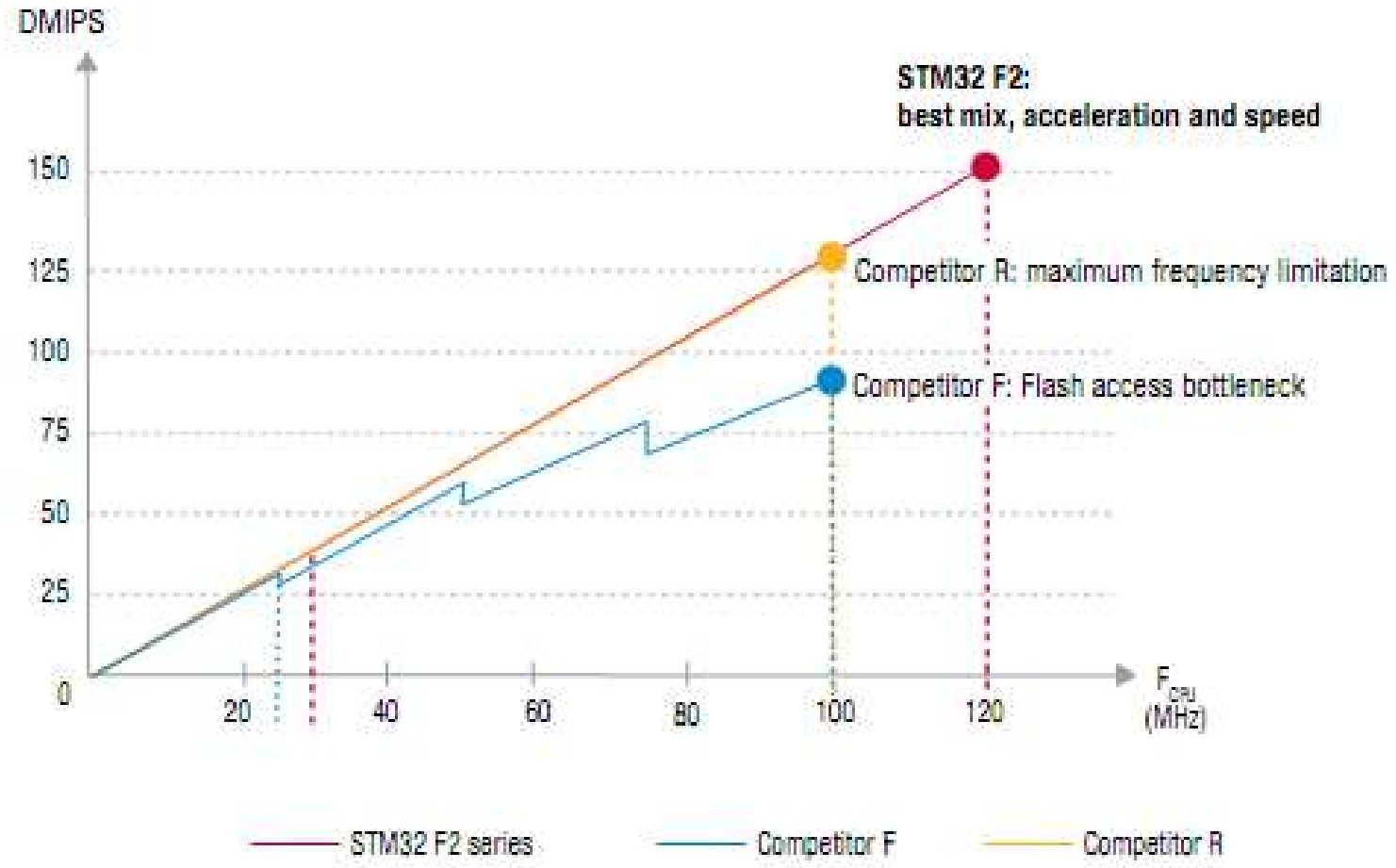


Harmadik generáció

- Nagyméretű gyorsító tár, már majdnem Cache



Második generáció vs. Harmadik generáció



Lábak alternatív funkciója

Alternatív funkciójú lábak

- Egy láb tipikusan lehet digitális, analóg be/kimenet, de általában többfajta perifériához is hozzá van rendelve
- Meg kell adni, hogy ezek közül melyik funkciót tölti be: Nagyon hasonló a filozófia a gyártóktól függetlenül

Table 80. Pin function select register 0 (PINSEL0 - address 0x4002 C000) bit description

PINSEL0	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.0	GPIO Port 0.0	RD1	TXD3	SDA1	00
3:2	P0.1	GPIO Port 0.1	TD1	RXD3	SCL1	00
5:4	P0.2	GPIO Port 0.2	TXD0	AD0.7	Reserved	00
7:6	P0.3	GPIO Port 0.3	RXD0	AD0.6	Reserved	00
9:8	P0.4[1]	GPIO Port 0.4	I2SRX_CLK	RD2	CAP2.0	00
11:10	P0.5[1]	GPIO Port 0.5	I2SRX_WS	TD2	CAP2.1	00
13:12	P0.6	GPIO Port 0.6	I2SRX_SDA	SSEL1	MAT2.0	00
15:14	P0.7	GPIO Port 0.7	I2STX_CLK	SCK1	MAT2.1	00
17:16	P0.8	GPIO Port 0.8	I2STX_WS	MISO1	MAT2.2	00
19:18	P0.9	GPIO Port 0.9	I2STX_SDA	MOSI1	MAT2.3	00
21:20	P0.10	GPIO Port 0.10	TXD2	SDA2	MAT3.0	00
23:22	P0.11	GPIO Port 0.11	RXD2	SCL2	MAT3.1	00

Összefoglalás

- Rendkívül bonyolult belső felépítés
- Rendkívül bonyolult órajel hálózat

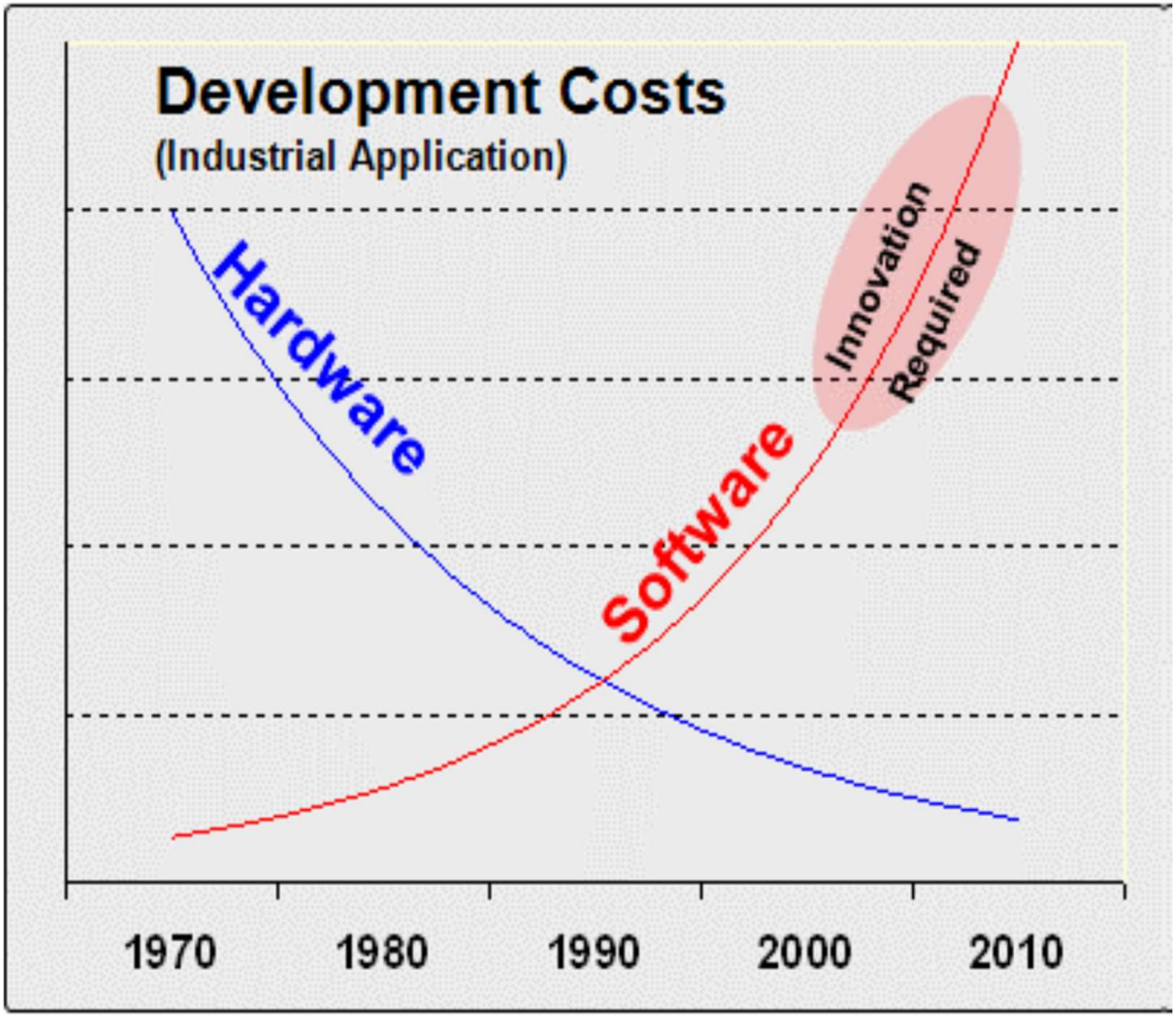
- Már egy LED villogtatás is komoly kihívás alacsony szinten tradicionális módszerekkel
  - Processzor elindítás, órajel beállítás
  - Buszok engedélyezése
  - Periféria engedélyezés
  - Láb alternatív funkció megadása
  - Láb kimenetbe állítása, egyéb tulajdonságok megadása
  - Láb vezérlése

CMSIS

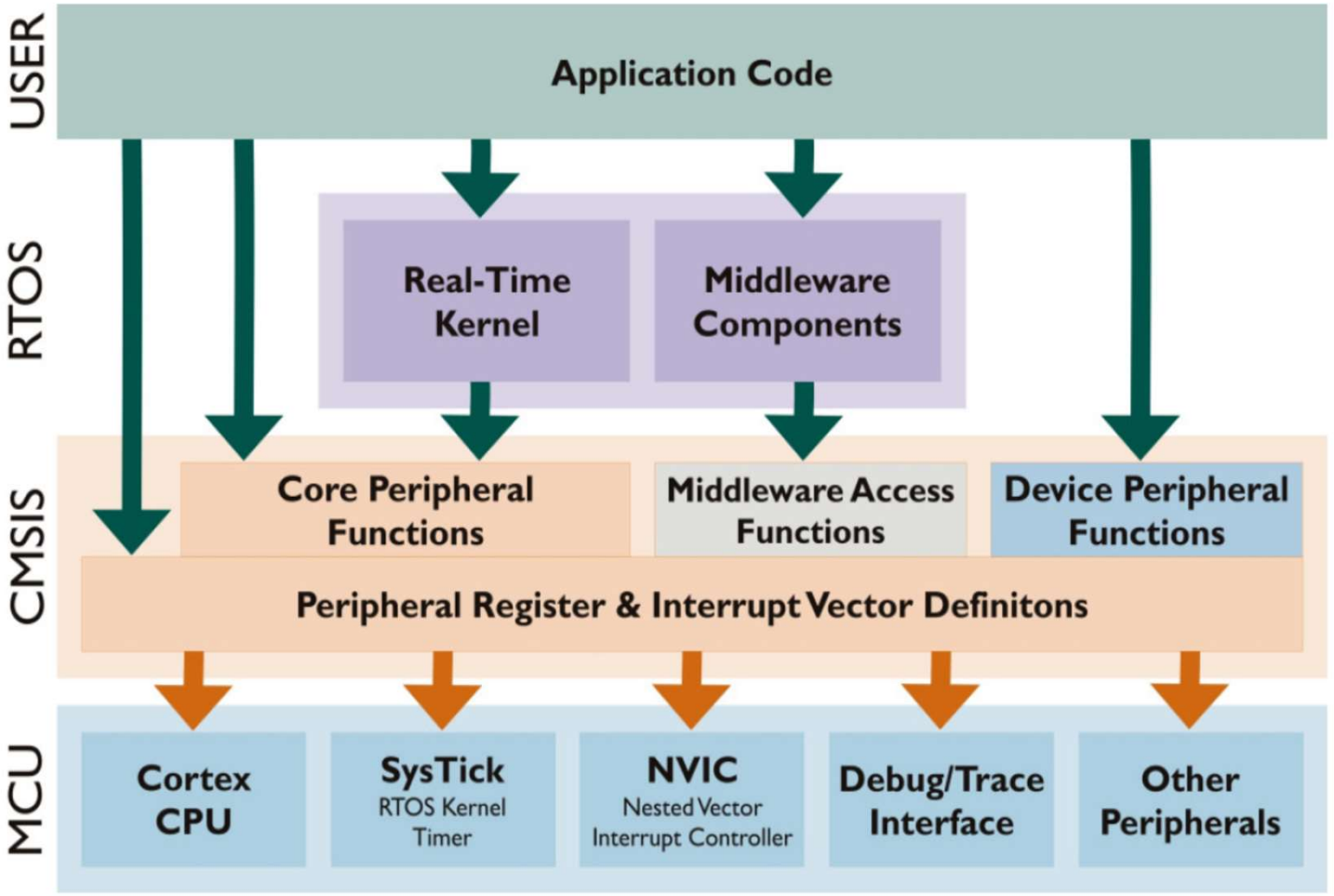
Cortex Microcontroller

Software Interface Standard

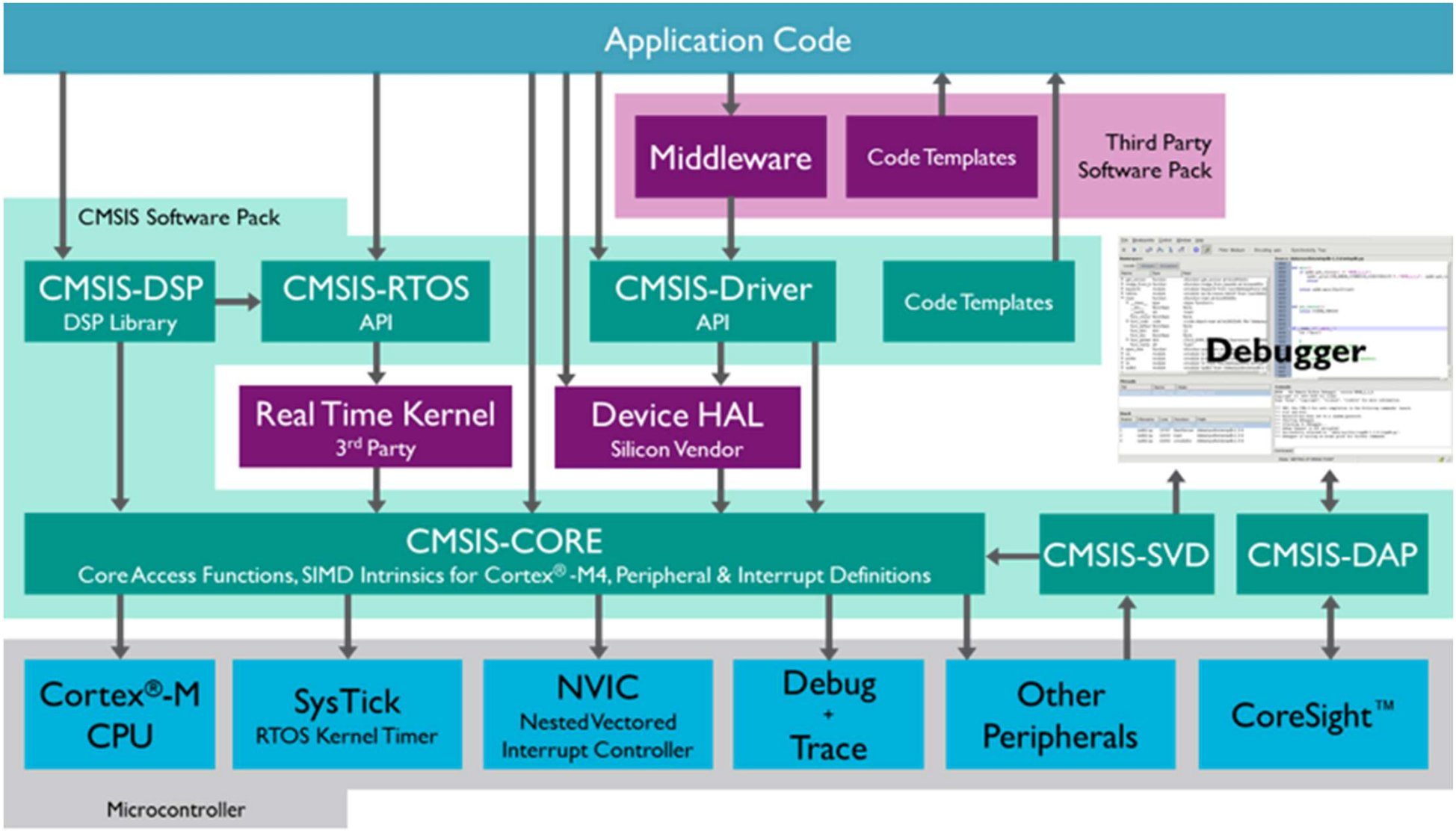
Fejlesztési költségek alakulása



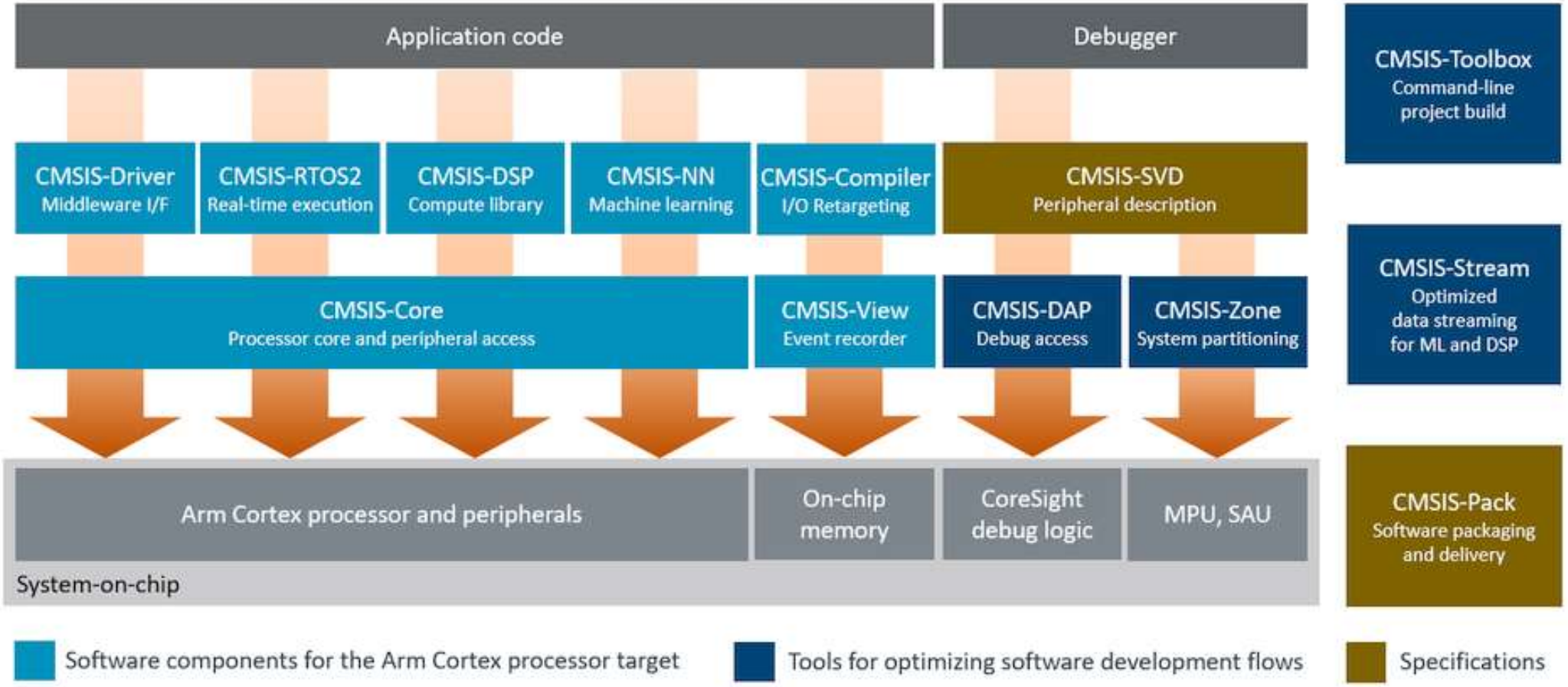
CMSIS szerkezete (v1.3) ~2008



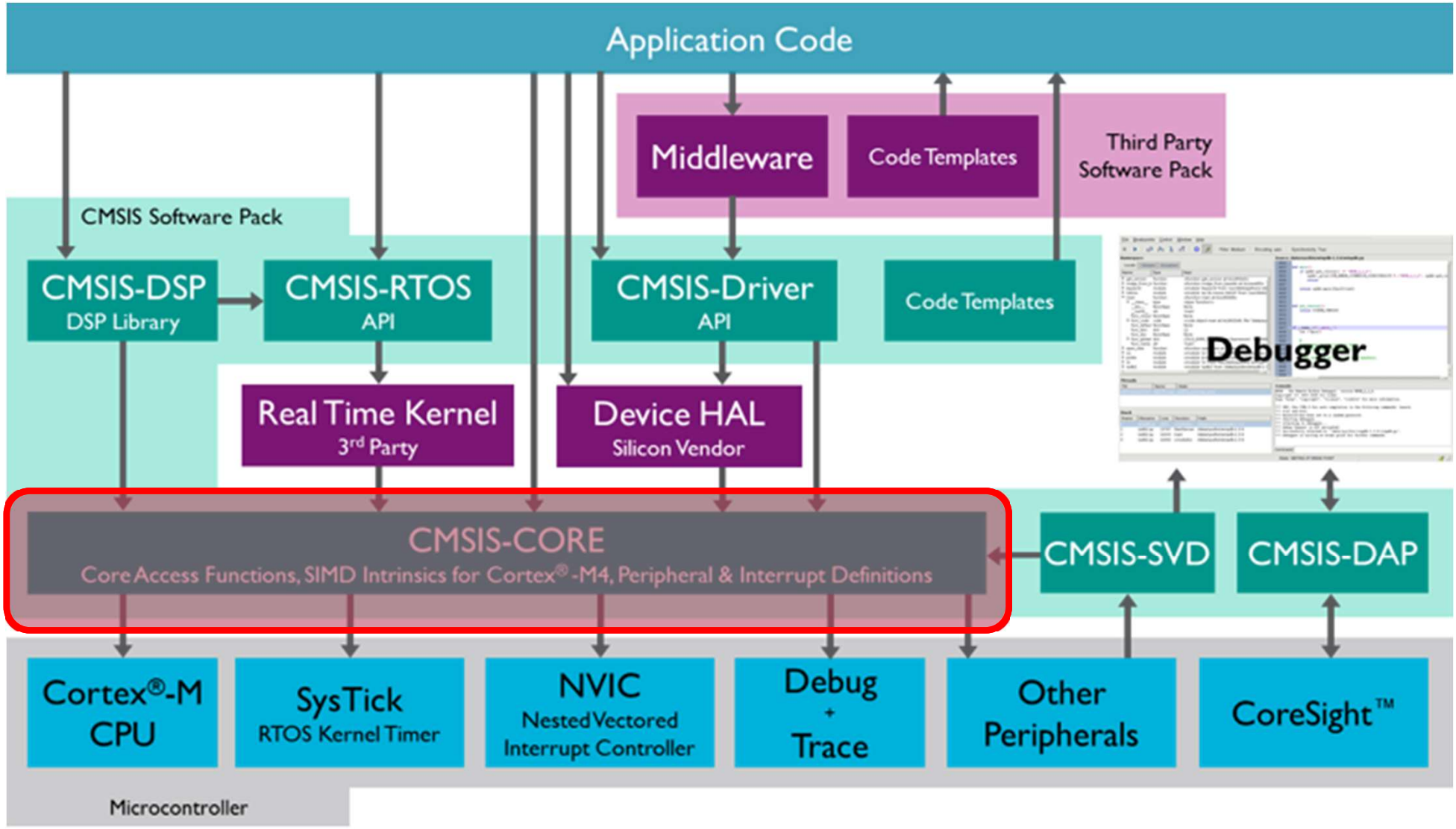
CMSIS szerkezete (v5) ~2019



CMSIS szerkezete (v6) ~2024



CMSIS Core


Application Code
Middleware
Code Templates
Third Party Software Pack
CMSIS Software Pack
CMSIS-DSP
DSP Library
CMSIS-RTOS
API
CMSIS-Driver
API
Code Templates
Debugger
Real Time Kernel
3rd Party
Device HAL
Silicon Vendor
CMSIS-CORE
Core Access Functions, SIMD Intrinsics for Cortex® -M4, Peripheral & Interrupt Definitions
CMSIS-SVD
CMSIS-DAP
Cortex®-M CPU
SysTick
RTOS Kernel Timer
NVIC
Nested Vectored Interrupt Controller
Debug + Trace
Other Peripherals
CoreSight™
Microcontroller

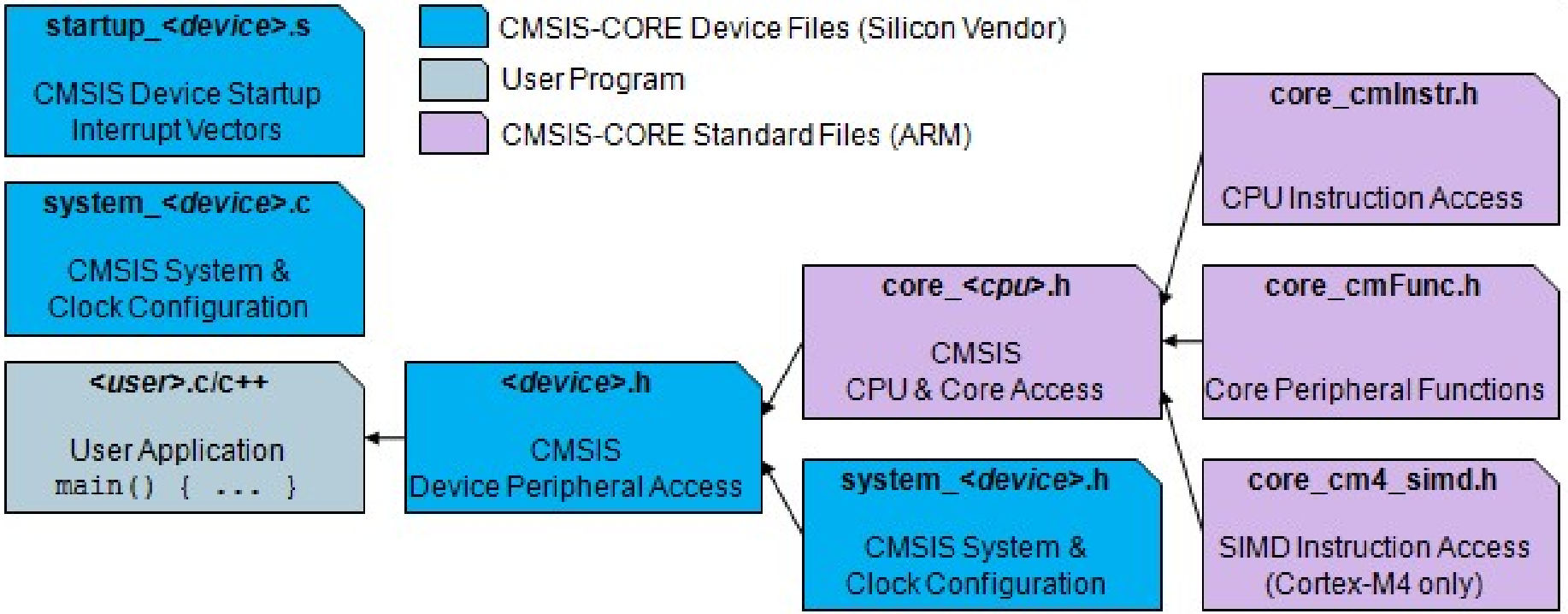
CMSIS Core

- **Hardware Abstraction Layer (HAL):** Az összes Cortex-M variánshoz egy standardizált periféria- és regiszterkészlet-kezelés a SysTick, NVIC, System Control Block, MPU, FPU regiszterszterekhez
- **Rendszerkivétel nevek:** A rendszerkivételekhez, interruptokhoz való hozzáférés definiálása
- **Header file szervezés definíciók:** Elnevezési konvenciók a mikrokontroller-specifikus interruptokhoz és header file-okhoz
- **Rendszerindítás:** Mikrovezérlő-független inicializáló függvény interfész. Standardized SystemInit() függvény
- **Speciális utasítások támogatása**
- Globális változó a **system clock** frekvencia meghatározására

A CMSIS file-jai és könyvtárstruktúrája

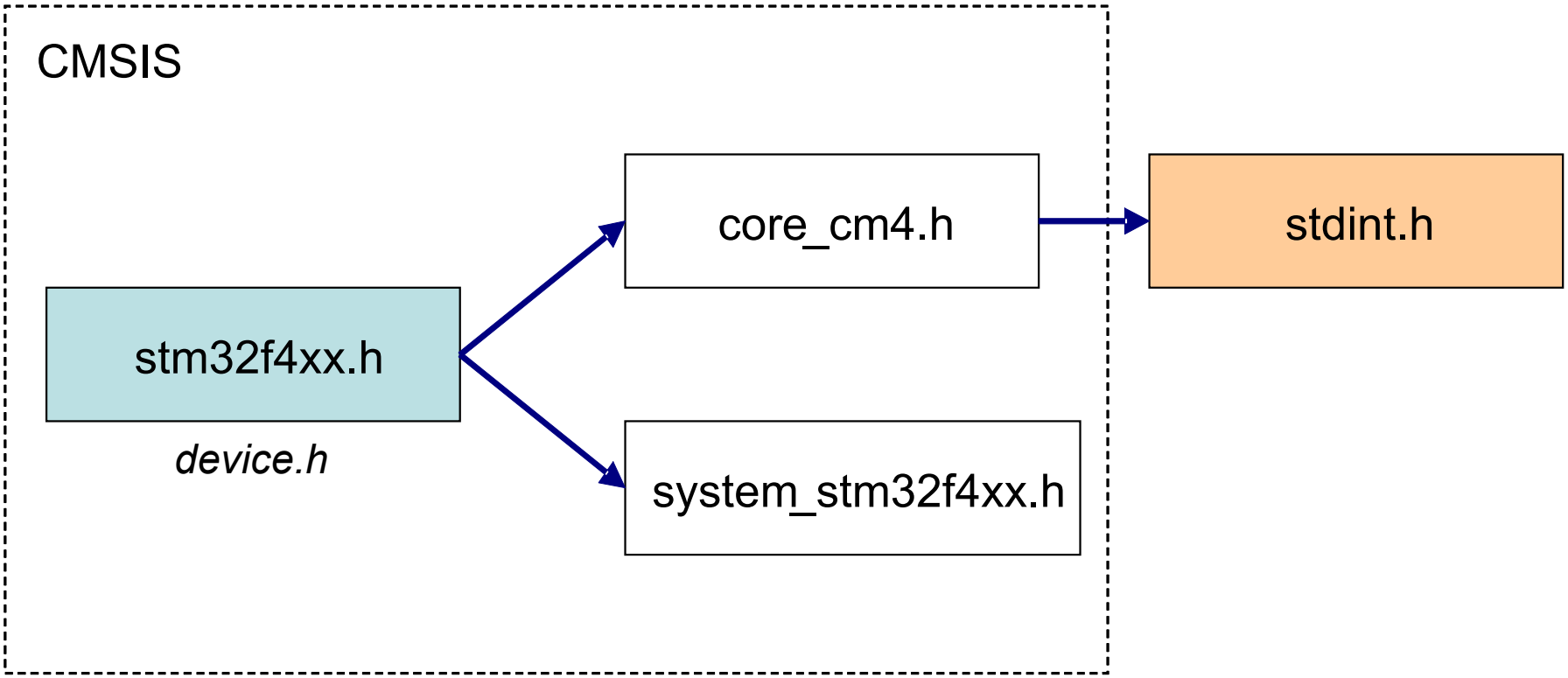
File	Szolgáltató	Leírás
device.h	Mikrovezérlő gyártó	A mikrovezérlő perifériáinak definítciója. Ez a file include-olhatja az összes többi mikrovezérlőhöz tartozó firmware
startup_*device*	ARM, de a mikrovezérlő, vagy compiler gyártó adoptálhatja és testreszabhatja	A Cortex-M mikrovezérlő startupkódja és a teljes mikrovezérlő függő ugrótábla
system_*device*	ARM, de a mikrovezérlő gyártó, adoptálhatja és testreszabhatja	Mikrovezérlő függő inicializációs rész. Tipikusan a PLL és egyébb órajelforrások inicializálása történik itt.
core_cm(x).h	ARM (fordítófüggő részekkel)	A Cortex-M(x) processzor alapperifériáinak és regisztereinek
core_cmFunc.h	ARM (fordítófüggő részekkel)	A Core regisztereihez való hozzáférés (státusz regiszter, IT prirorítás stb.). Sokszor egy külön fordító függő header file-ra
core_cmInstr.h	ARM (fordítófüggő részekkel)	A Core átlal támogatott utasításokhoz nyújt C nyelvű hozzáférést. Sokszor egy külön fordító függő header file-ra
core_cmSimd.h	ARM (fordítófüggő részekkel)	A Core átlal támogatott SIMD utasításokhoz nyújt C nyelvű hozzáférést. Sokszor egy külön fordító függő header file-ra
core_math.h	ARM (fordítófüggő részekkel)	Speciális DSP függvényekhez ad hozzáférést, Inline függvények, vagy egy előre letöltött library-ra hivatkoznak

CMSIS core struktúra



A device.h szerepe

- Az egyetlen szükséges include file a felhasználó számára (az induláshoz)



A *startup_device* file

- Fordítófüggő
- Startup Code, ugrótábla
- A GCC megvalósításnál a ugrótáblákhoz úgynevezett ***weak*** pragmákat használnak.

```
DCD       USART1_IRQHandler,              /* USART1 */
```

Majd az így definiált nevet egy ***weak*** pragmával ráhuzalozzuk egy default handlerre:

```
#pragma weak USART1_IRQHandler = Default_Handler
```

A system_device.c

- Minden Cortex-M vezérlő azonos módon elindítható legyen
- Minimum szolgáltatások

Függvény definició	Leírás
void SystemInit (void)	A mirovezérlő elindulásához szükséges rutinok végrehajtása. Tipikusan ez a függvény konfigurálja a PLL-t. Ez a függvény tipikusan a startup_device-ból hívódik meg, de lehet, hogy a felhasználónak kell meghívnia.
void SystemCoreClockUpdate(void)	Frissíti az aktuális értékre a SystemCore változót. Minden a központi órajelet érintő konfiguráció után meg kell hívni

Változó definítció	Leírás
uint32_t SystemCoreClock	A rendszer órajel frekvenciáját tartalmazza.

A CMSIS coding guideline-jai

- A CMSIS szabvány a MISRA 2004-es guideline-jainak betartásával készül.
- A CMSIS az adattípusokként az **<stdint.h>** -ba definiált típusokat használja.
- A kifejezéseket tartalmazó #define-okat zárójelezni kell.
- A *Core Peripheral Access Layer (**CPAL**)* összes függvénye re-entráns kell, hogy legyen.
- A *Core Peripheral Access Layer (**CPAL**)* nem tartalmazhat blokkoló kódot.
- Az összes interrupt-kezelő függvények az **_IRQHandler** utótaggal kell záródnia.