ARM Cortex Core Microcontrollers

4. System Control block

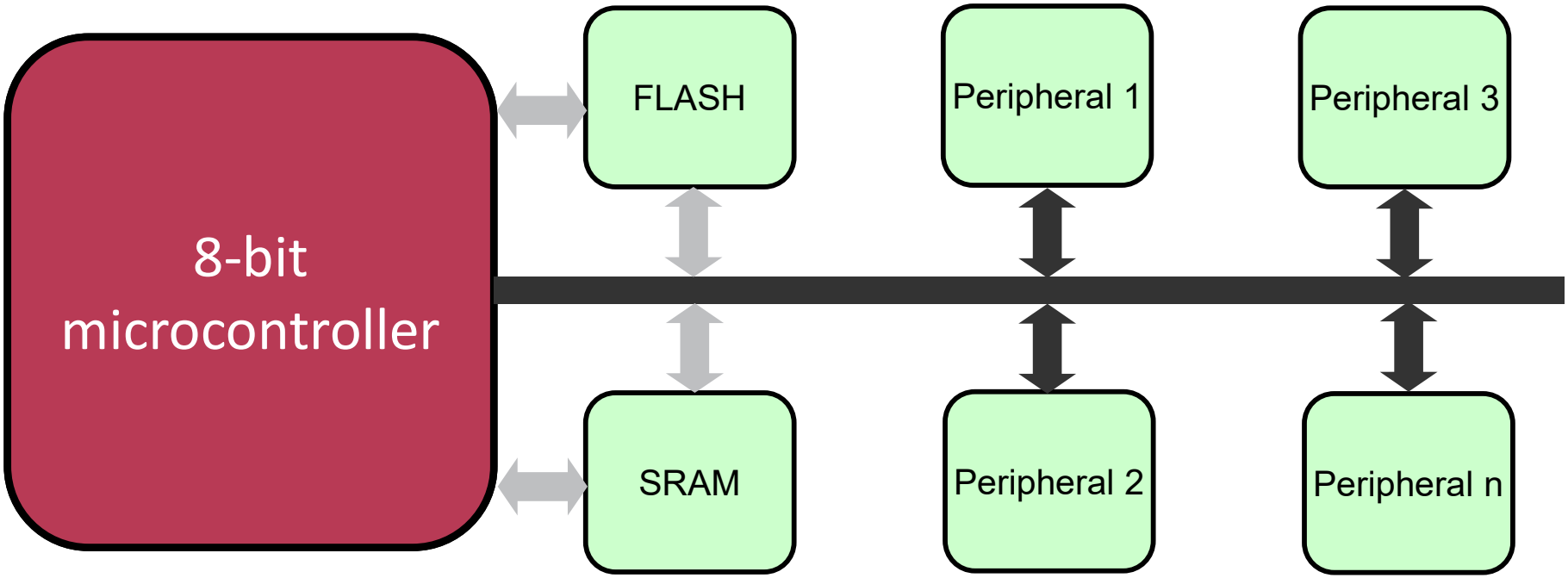
Balázs Scherer

Méréstechnika és
Információs Rendszerek
Tanszék

Budapest University of Technology and Economics
Department of Measurement and Information Systems

Evaluation of internal architecture of ARM core microcontrollers

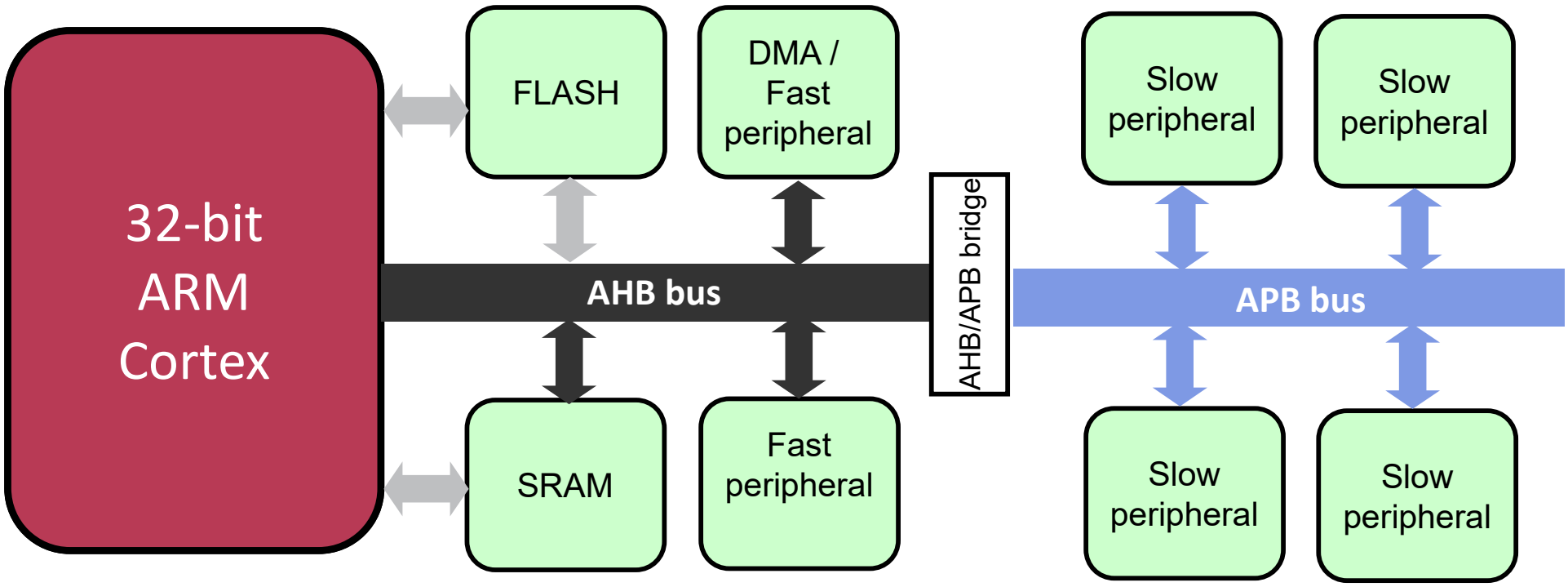
Internal architecture of a typical 8-bit microcontroller



- Flash, SRAM connected to separate data bus, or directly to the system bus
- Newest (2015+) 8-bit micros have improved internal architecture like the 32-bit ones.

32-bites ARM Core based micros

32-bit ARM Core first generation



- Two bus architecture with different capabilities: AHB - APB
- Used for the old generation (before 2010), and for the simplest controllers
  - Cortex M0, with low complexity features

AHB vs APB

- Both part of the ARM Advanced Microcontroller Bus Architecture (AMBA) standard

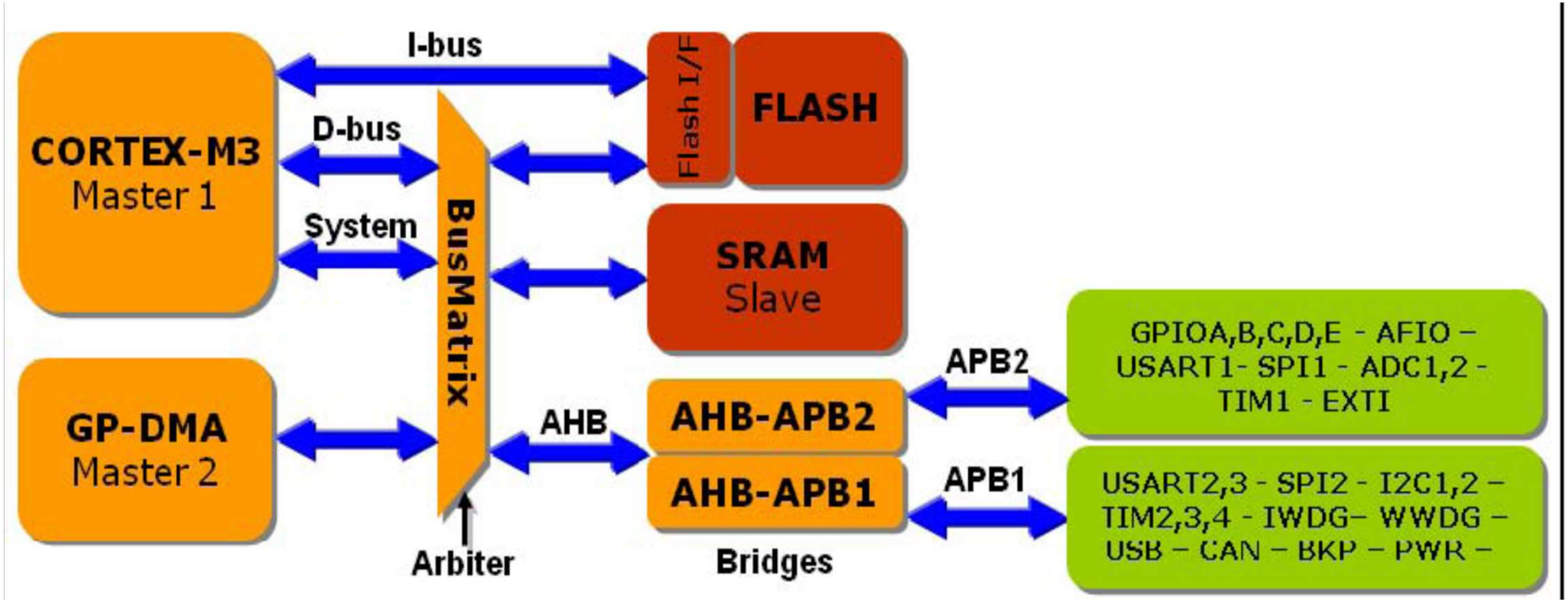
- **AHB** Advanced High-performance Bus
  - Pipelining
  - Multiple master
  - Burst transaction
  - Full-duplex parallel comm.

- **APB** Advanced Peripheral Bus
  - No Pipelining
  - Single master
  - Small complexity
  - Small power
  - 32-bit bus

Second generation of Cortex M3

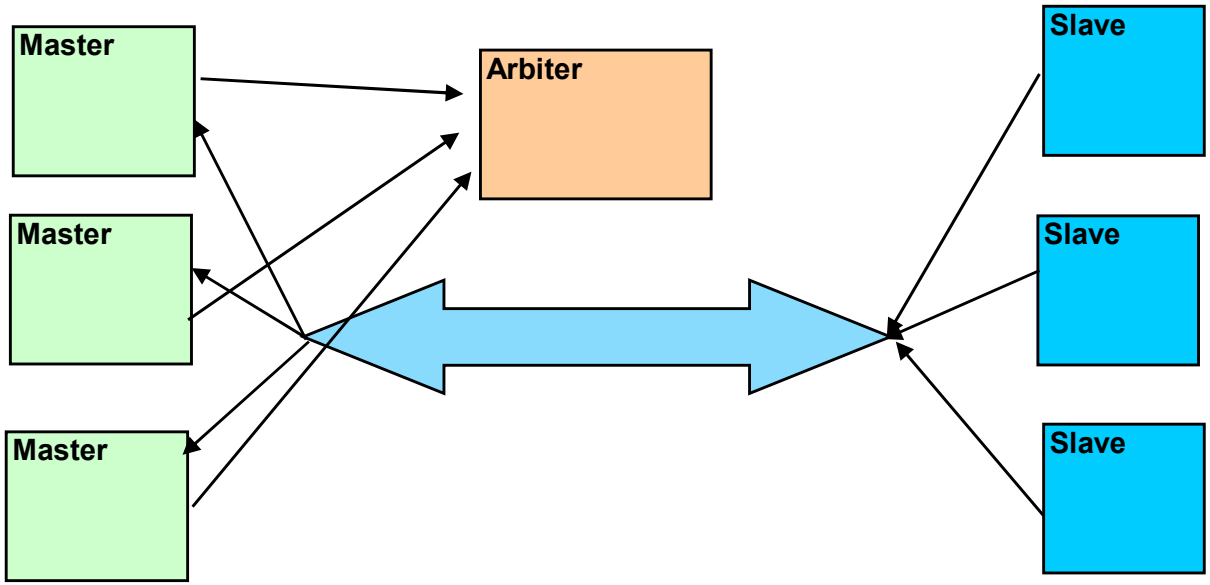
2007: STM32F103 (Max 72 MHz)

- APB1: max. 72MHz
- APB2: max. 36MHz

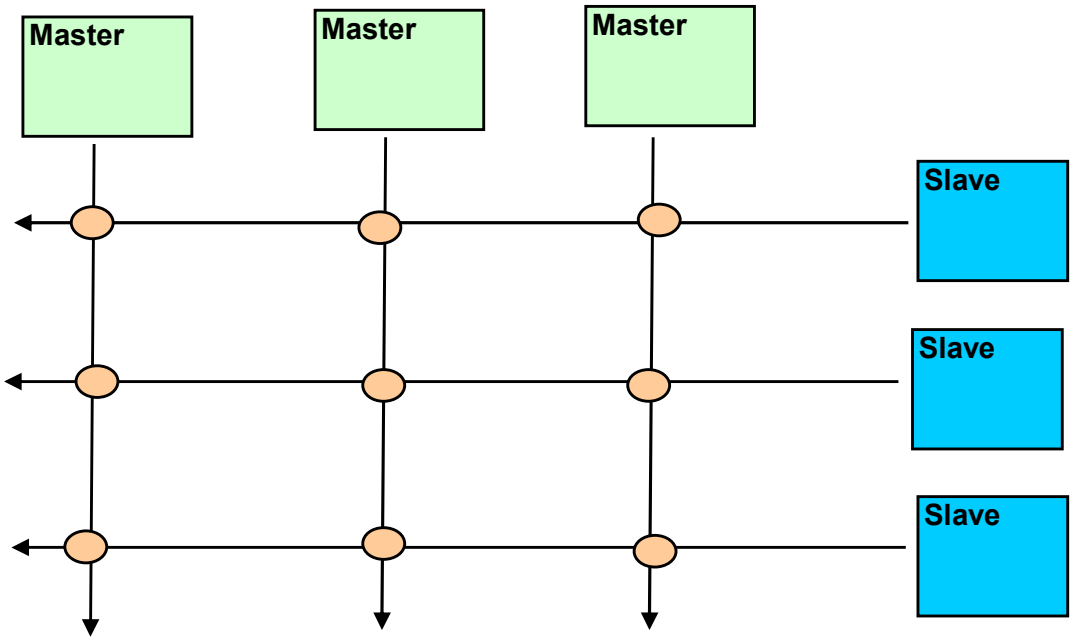


Multi master bus system

Shared AHB Bus

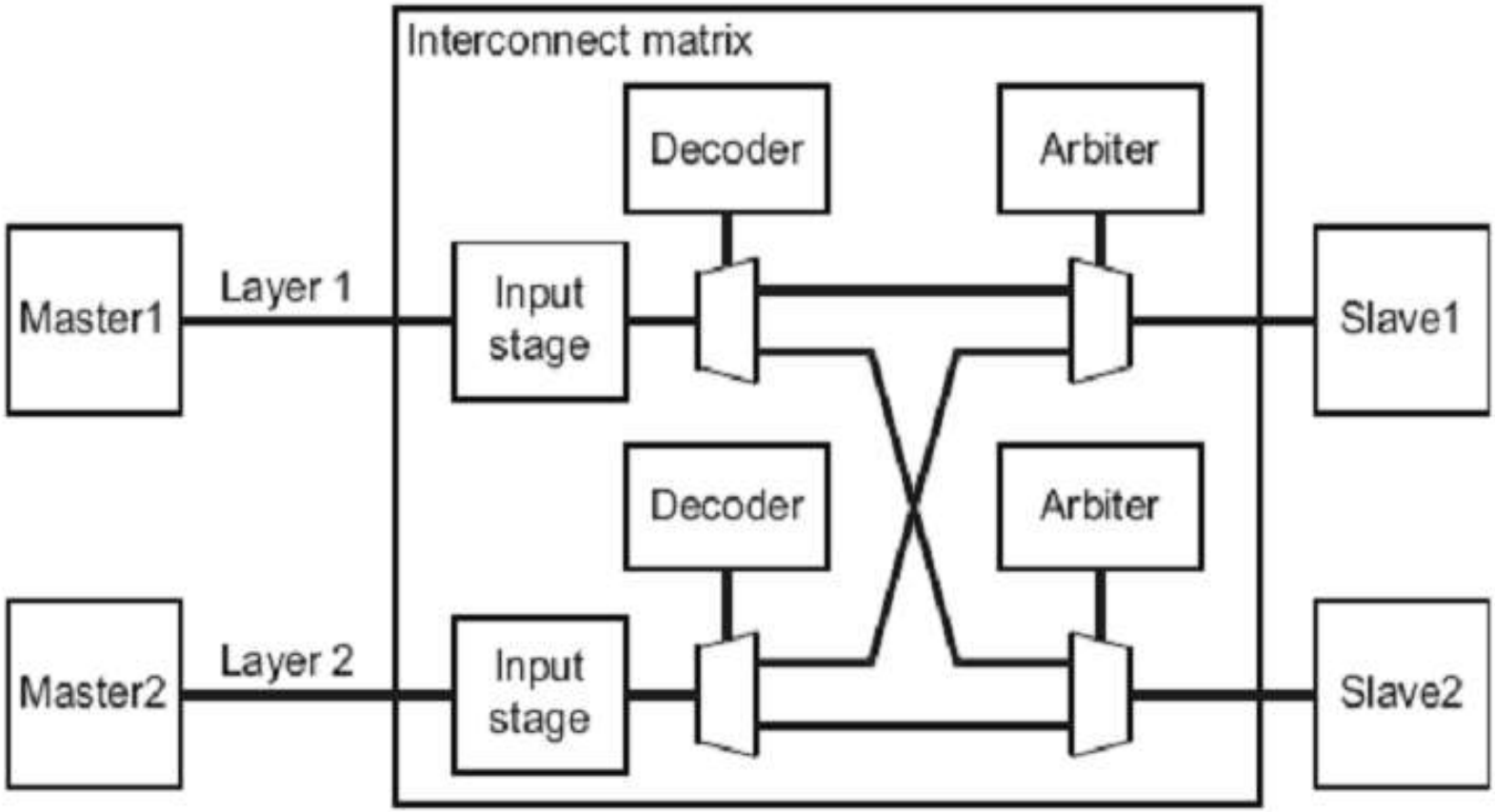


AHB Bus Matrix



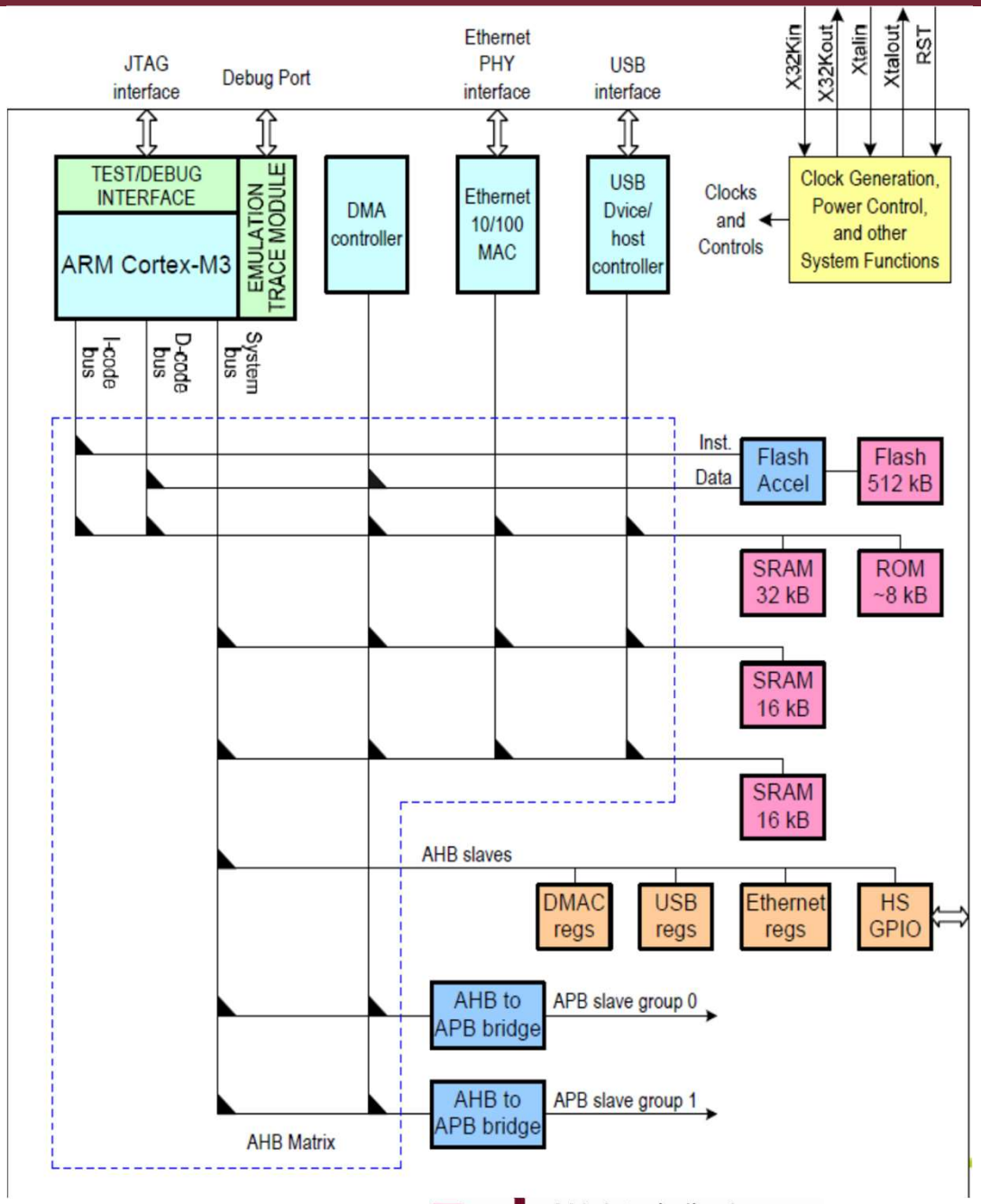
What happens in the matrix

- Arbitration: usually round-robin



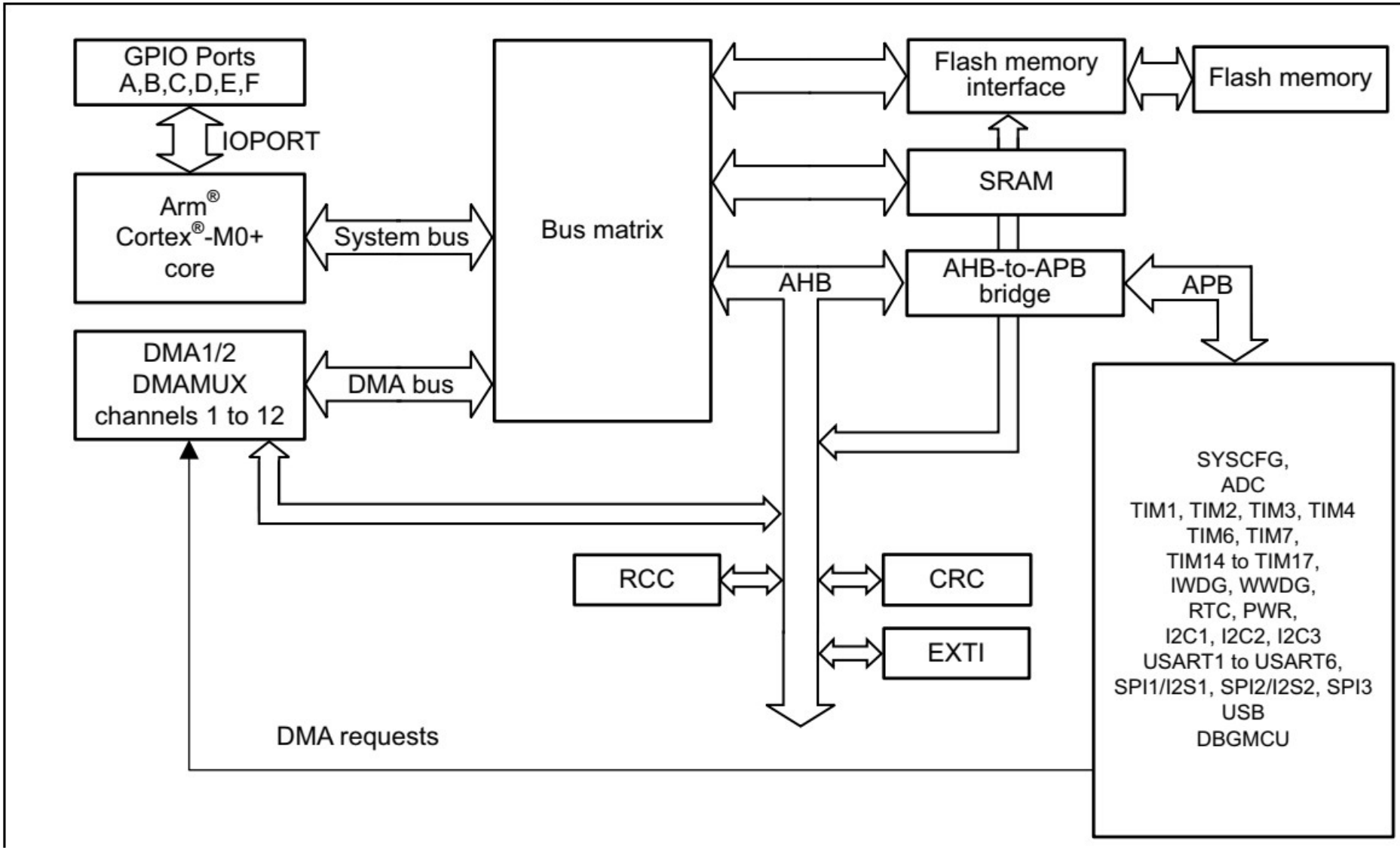
2.1 generation

- After 2009
- Bus matrix
- Separate SRAM blocks supporting multi master operations
- Most of the current micros using this internal architecture

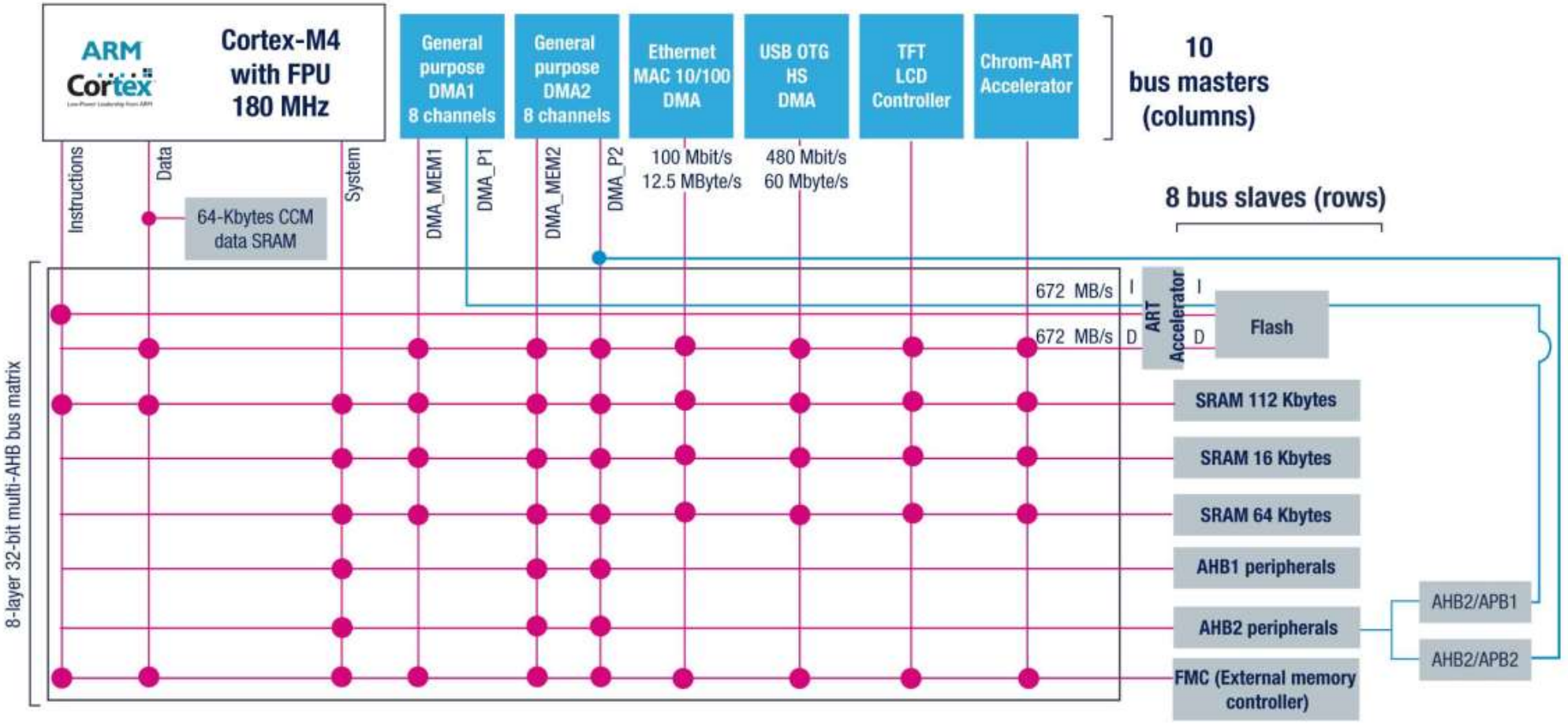


M0+ example: STM32G0x0 line

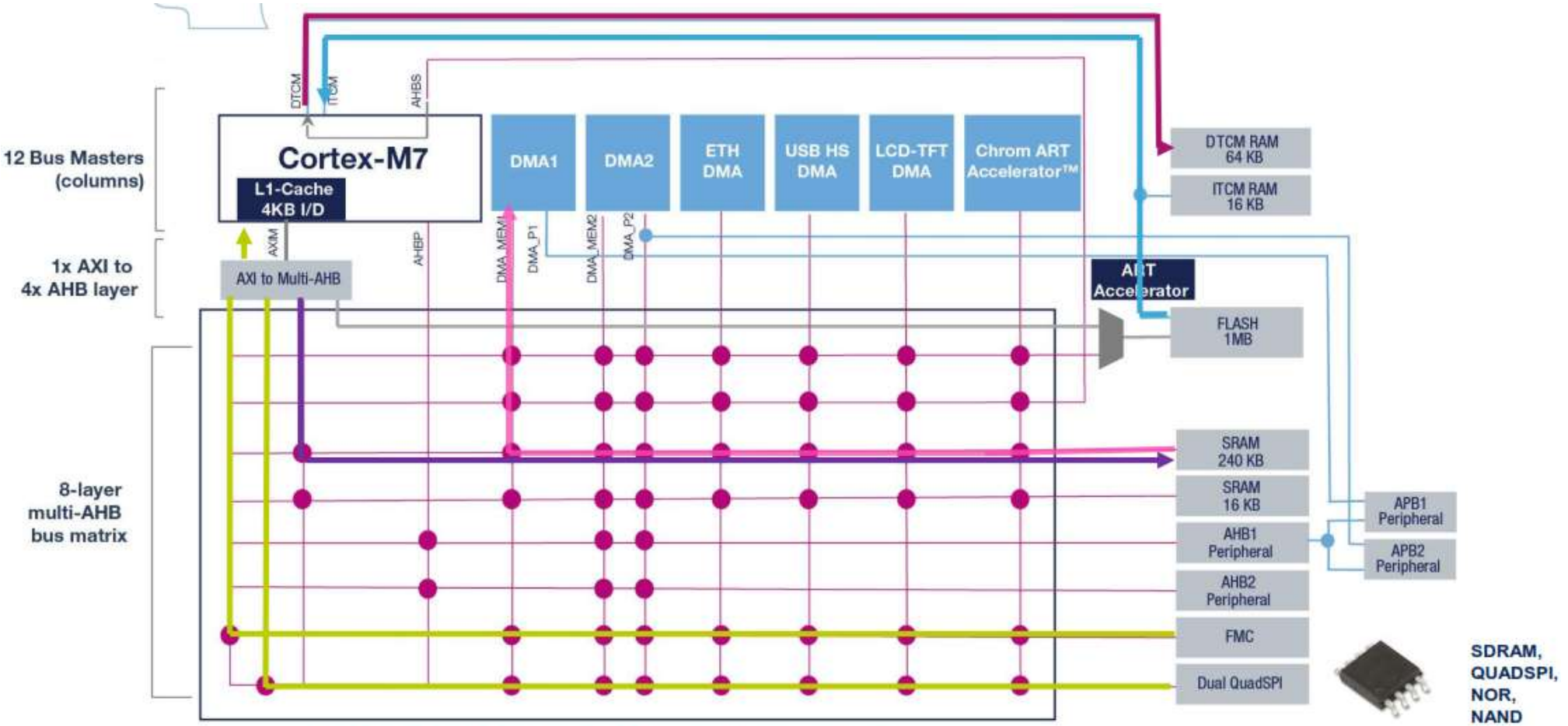
Figure 1. System architecture



M4 example: STMF4xxx



M7 example: STM32F7xx



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

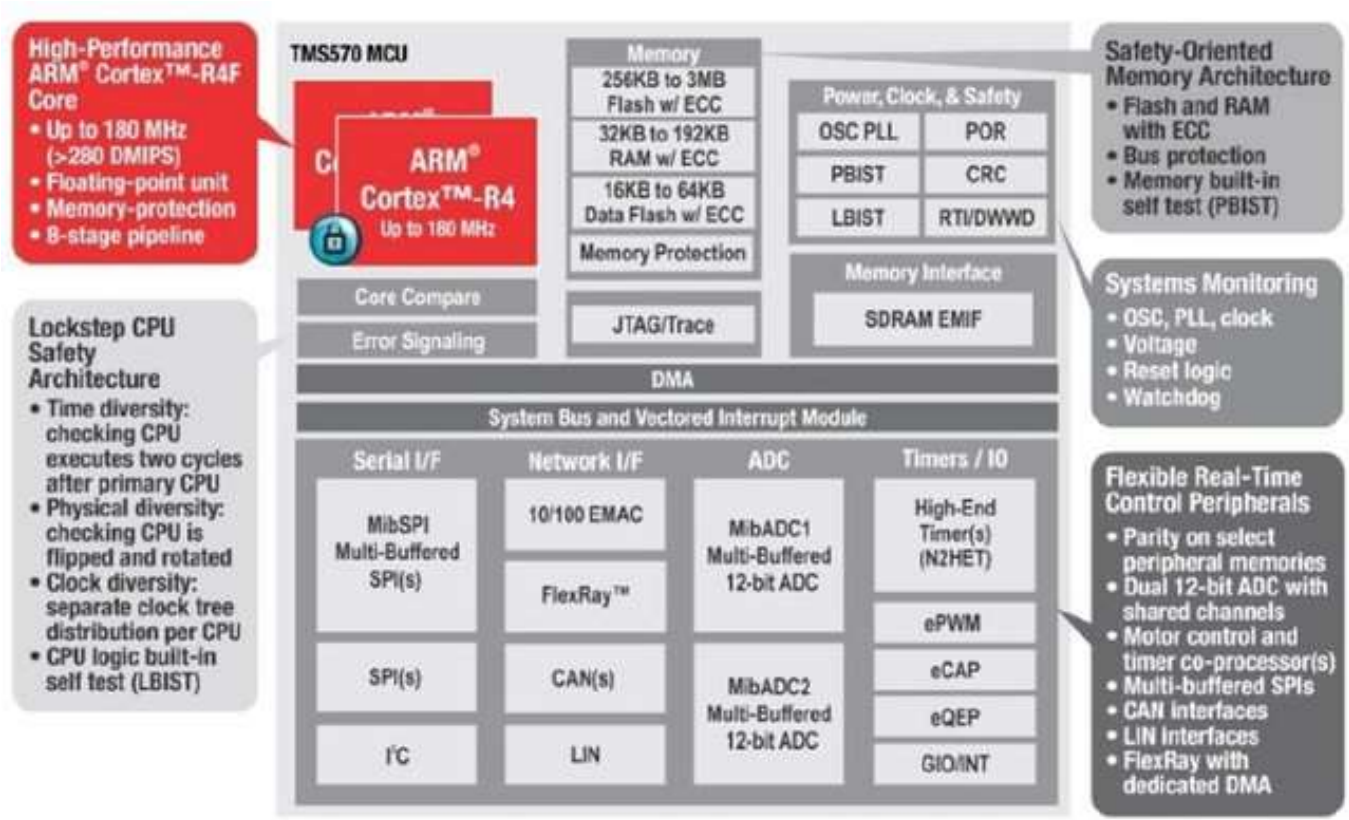
Multicore microcontrollers

Több magú mikrovezélők

- Not the same way multicore as a desktop PC or laptop
  - Desktop PC: multi cores with homogenic cores
  - Operating system allocates jobs to the cores
- For microcontrollers no such multicore OS
- Usual ways of multicore microcontrollers
  - Safety purpose: Dual core lock step microcontrollers
  - General purpose dual core micros with heterogenic cores to dedicated functions

Safety micros

- Dual core lockstep
  - Same processor core
  - They executing the same program
  - The results are compared: protection against hazards
  - Microcontroller has additional safety features: ECC memory …



General purpose microcontrollers

- Usually, heterogeny architecture
  - A M7 and a M4 core, or a M4 core and a M0 core
- Dedicated functions for the cores
- Separate software, separate development process for each cores
- Each Cores attached to the system matrix
- Program and data memory separated to multiple banks
  - Dedicated areas for each cores
  - Shared areas for communication

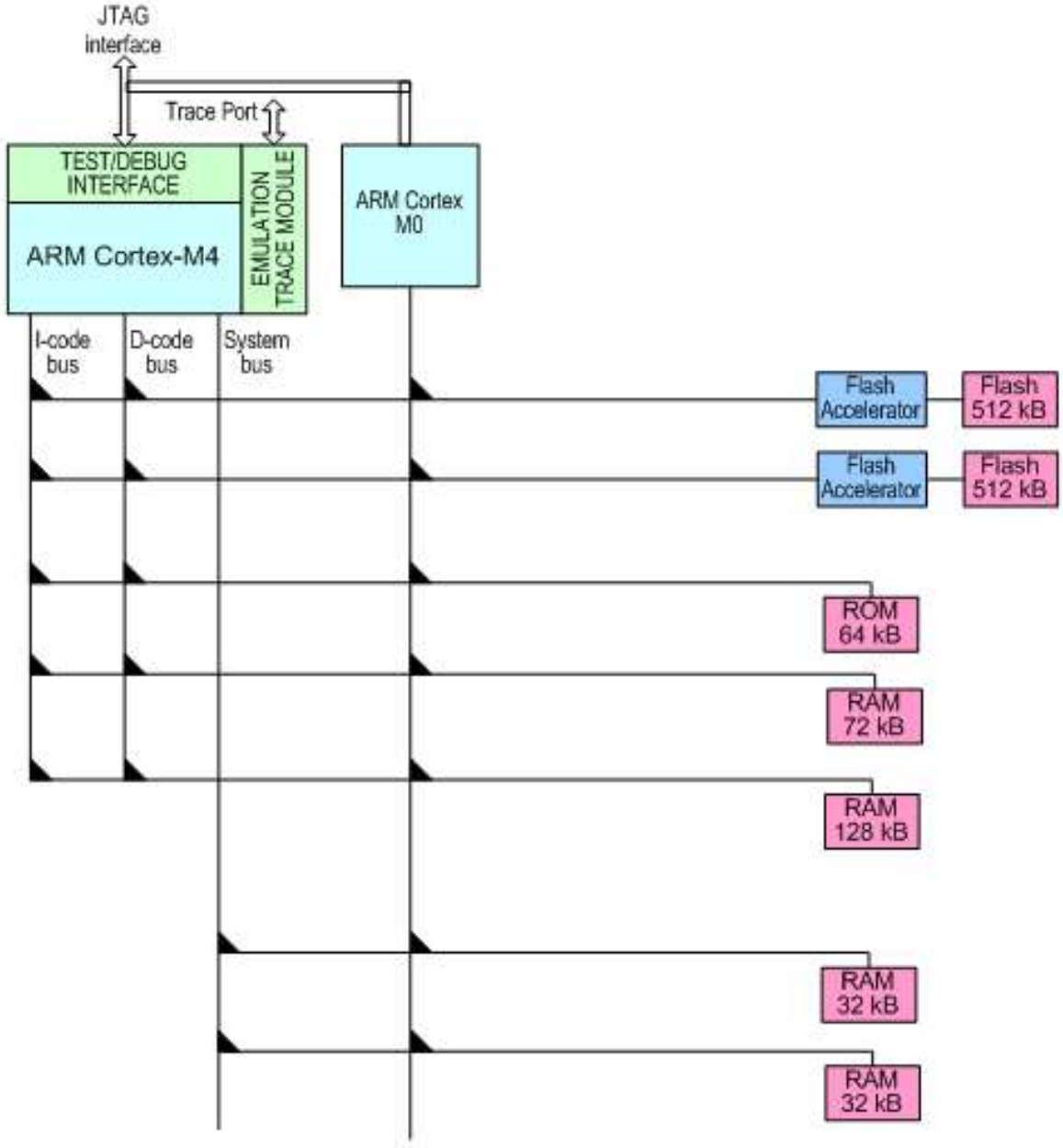
LPC4300 family

- Cortex-M4 based Digital Signal Controller
- Cortex-M0 subsystem for peripheral functions
- max. 1 MByte Flash
  - Organised into two banks Flash
- Max. 200 kbyte SRAM
- High speed USB
- Features
  - 10/100 Ethernet MAC
  - LCD panel controller (max. 1024H × 768V)
  - 2x10-bit ADC and 10-bit DAC at 400 ksps
  - 8 channel DMA controller
  - Motor Control PWM, Quadrature Encoder
  - 4x UARTs, 2x I2C, I2S, CAN 2.0B, 2x SSP/SPI

LPC4300 internal architecture

Memory system of LPC43xx

- Dual Core
  - Both the M4 and M0 core can accesses the Flash
  - RAM can be used to share data
  - The MPU of M4 can protect the regions used by the M0 core



System Control Block

System Control block

- Selecting the clock source of the system
  - External Quartz Crystal, Internal RC oscillator, Real-time quartz
- PLL (Phase Locked Loop)
  - Determination of system clock rate
- Determination of the peripheral clock rates
  - Specifying the relationship between the system clock rate and peripheral bus clock rates
- Controlling the Flash access
  - Flash acceleration, Number of wait cycles
- Controlling the power source of peripherals
  - In modern microcontrollers every peripheral can be switched on or off.
- Determination of pin alternate functions

Reset

The Reset event

- Source of the reset signal
  - Power-on
  - Watchdog
  - Brown – out
  - External pin
  - Software
- The source of the reset is identifiable by reading a register
- What happens after the reset event?

The NVIC vectors

- Reset vector is at 0x00000004
  - The 0x00000000 is the stack pointer to enable the early usage of C language

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
4	MemManage Fault	0	settable	MPU violation or access to illegal locations
5	Bus Fault	1	settable	Fault if AHB interface receives error
6	Usage Fault	2	settable	Exceptions due to program errors
7-10	Reserved	N.A.	N.A.	
11	SVCall	3	settable	System Service call
12	Debug Monitor	4	settable	Break points, watch points, external debug
13	Reserved	N.A.	N.A.	
14	PendSV	5	settable	Pendable request for System Device
15	SYSTICK	6	settable	System Tick Timer
16	Interrupt #0	7	settable	External Interrupt #0
.....	.....	.....	settable	.....
256	Interrupt#240	247	settable	External Interrupt #240

Microcontroller specific

Flash acceleration

Flash memory

- The Flash requires less space than RAM, but slower
- Read access time of Flash is about>
  - 30ns - 50ns (33 –25 MHz)
- This is too slow to run the micro at 60, 72, 120, 180, 200, 300 MHz
- Solutions
  - Run the code from RAM
    - But, the RAM is costly and not power efficient
  - Increase the bus width of the Flash memory
    - 64bit, 128 bit
    - Increase complexity

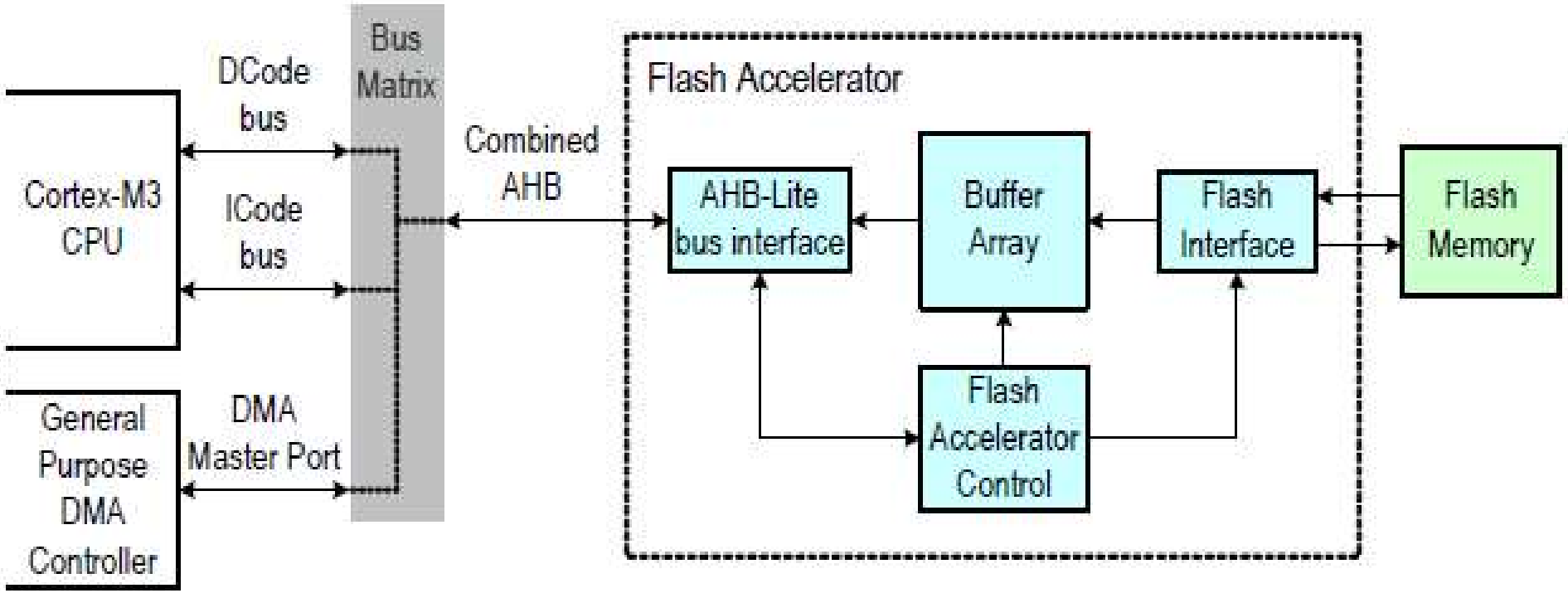
The solution used at STM32F10x at 2009

- 2 pieces of 64-bit prefetch buffer
- Need to program the number of wait cycles

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

Solution used in the LPC1768 at 2010

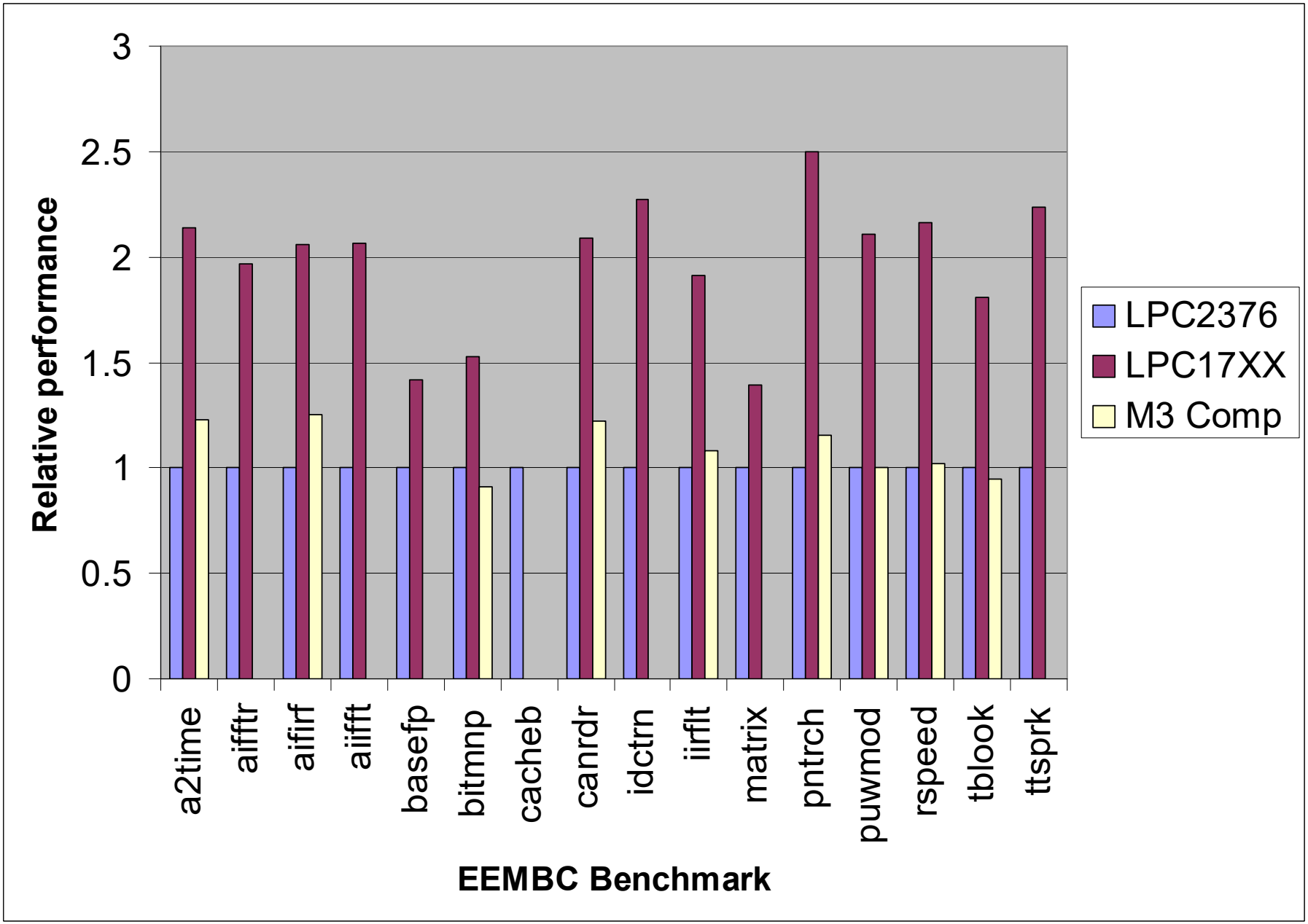
- 8 pieces of 128-bit buffer
- Can fetch constant data



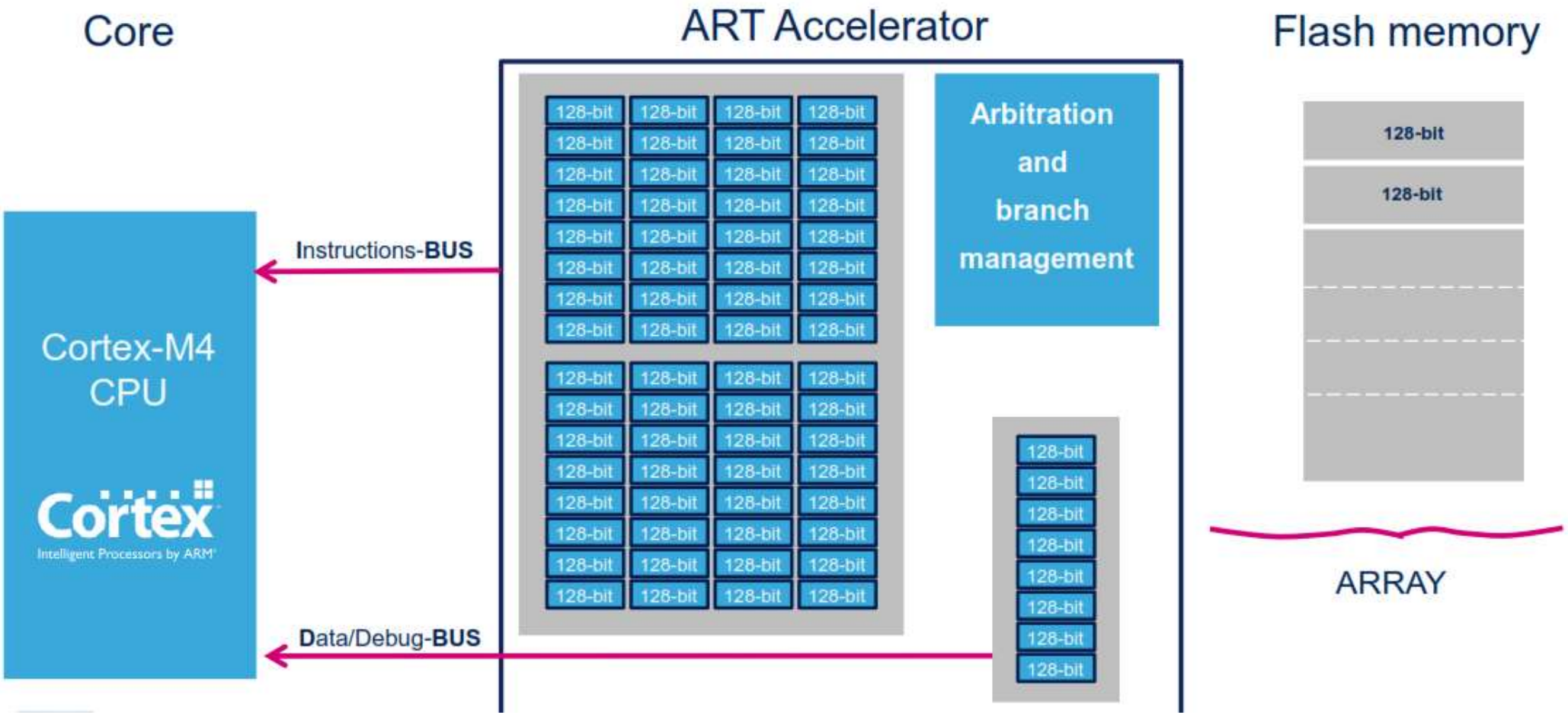
Performance of the Falsh accelerator in the LPC1768

- Comparing to a RAM based execution
  - The executing speed is in a 16% region to the RAM based execution
  - The power consumption is 25% less
- Comparing to the old ARM7 version with 128-bit Flash access
  - 45% increase in performance

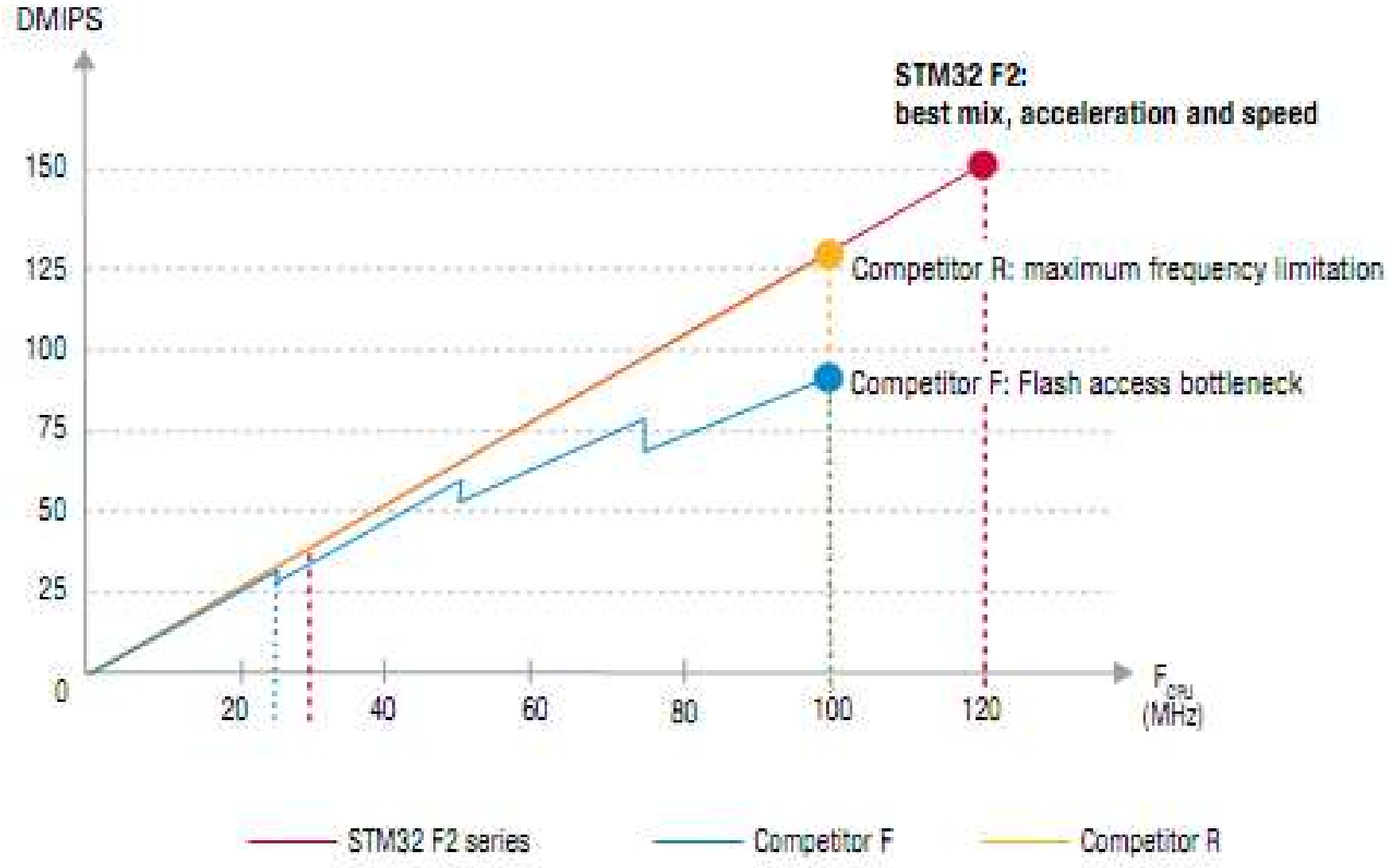
Benchmark results



STM32F2xx/STM32F4xx the latest solution



STM32F2xx/STM32F4xx the latest solution



Clock systems

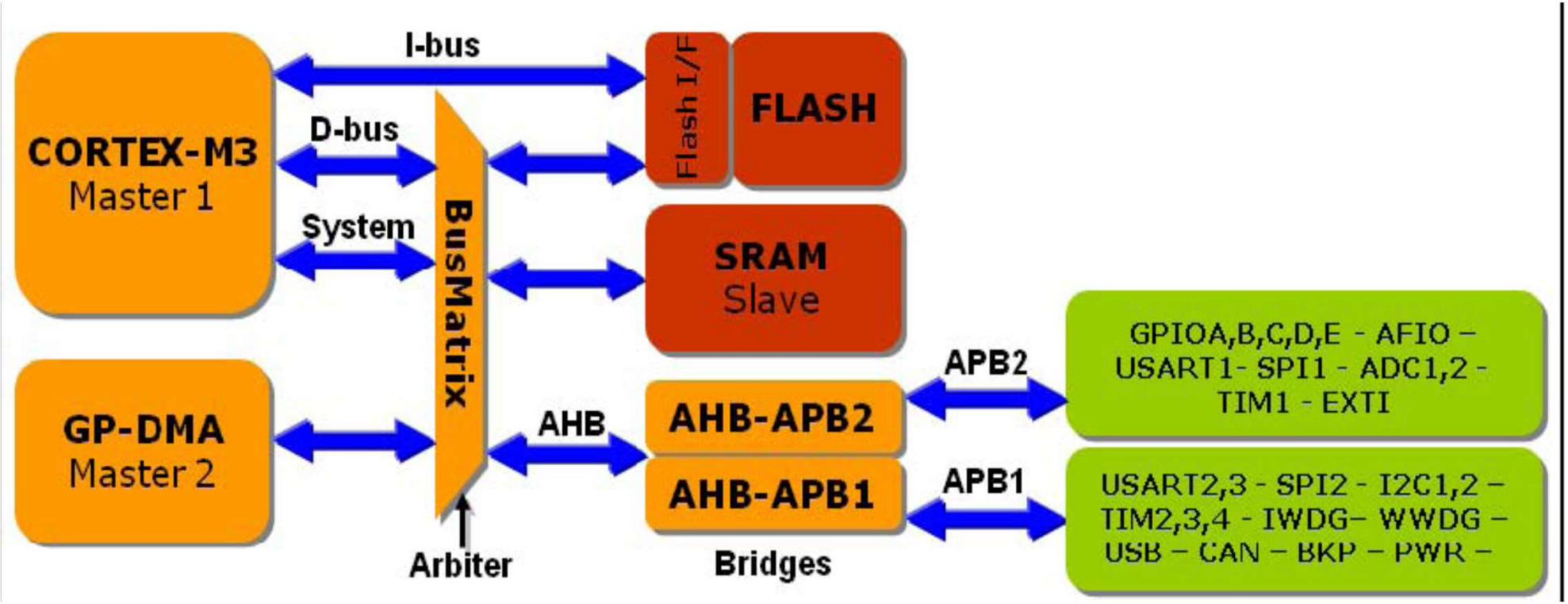
Problems of clock distribution

- Many type of source clock sources
  - Quartz Crystal
    - Precise, stable, but costly
  - RC oscillator
    - Un-Precise, cheap
- Many requirements from the peripheral set
  - Simple base peripherals
  - I/O pins
  - Ethernet
  - USB

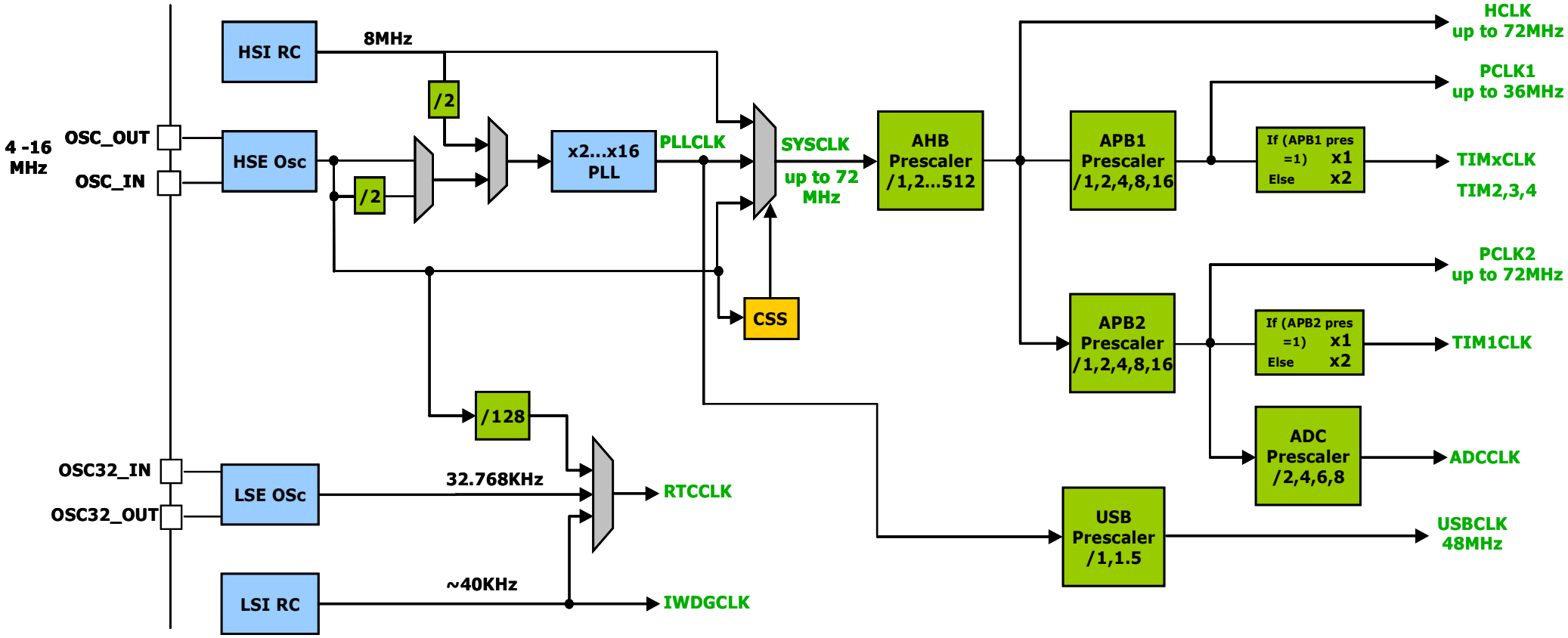
Second generation of Cortex M3

2007: STM32F103 (Max 72 MHz)

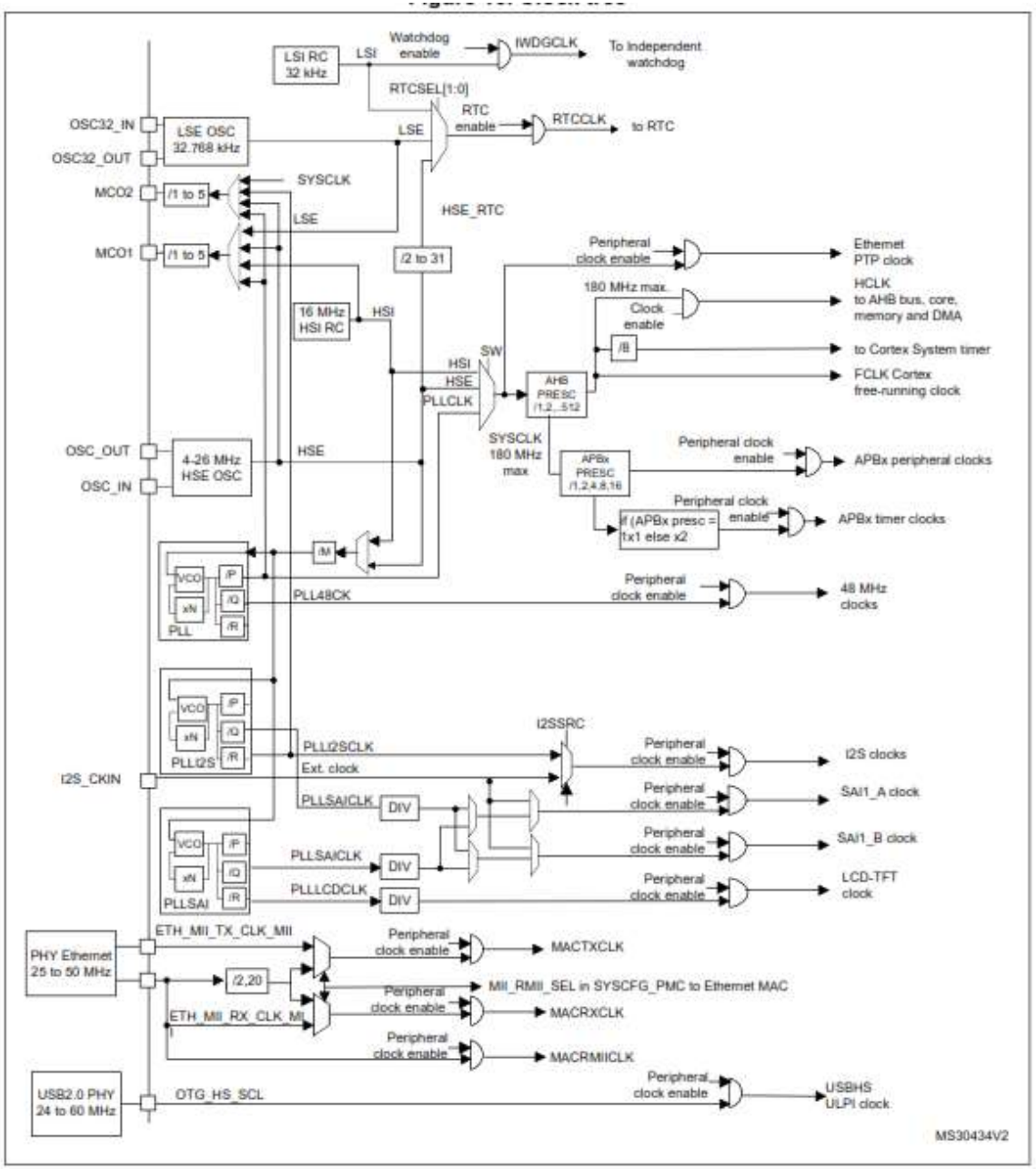
- APB1: max. 72MHz
- APB2: max. 36MHz



Clock tree of the STM32F1xx



Clock tree of the STM32F4xx



Problems caused by the clock tree

- A simple LED switching requires at least 1 clock setting, but for a complex micro 3-4 clock config parameters should be programed

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

Peripheral power also should be checked

- Very microcontroller specific

Table 46. Power Control for Peripherals register (PCONP - address 0x400F C0C4) bit description

Bit	Symbol	Description	Reset value
0	-	Reserved.	NA
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	-	Reserved.	NA
6	PCPWM1	PWM1 power/clock control bit.	1
7	PCI2C0	The I^2C0 interface power/clock control bit.	1
8	PCSPI	The SPI interface power/clock control bit.	1
9	PCRTC	The RTC power/clock control bit.	1
10	PCSSP1	The SSP 1 interface power/clock control bit.	1
11	-	Reserved.	NA
12	PCADC	A/D converter (ADC) power/clock control bit. **Note:** Clear the PDN bit in the AD0CR before clearing this bit, and set this bit before setting PDN.	0
13	PCCAN1	CAN Controller 1 power/clock control bit.	0
14	PCCAN2	CAN Controller 2 power/clock control bit.	0
15	PCGPIO	Power/clock control bit for IOCON, GPIO, and GPIO interrupts.	1
16	PCRIT	Repetitive Interrupt Timer power/clock control bit.	0
17	PCMCPWM	Motor Control PWM	0
18	PCQEI	Quadrature Encoder Interface power/clock control bit.	0
19	PCI2C1	The I^2C1 interface power/clock control bit.	1
20	-	Reserved.	NA
21	PCSSP0	The SSP0 interface power/clock control bit.	1
22	PCTIM2	Timer 2 power/clock control bit.	0

Alternate functions of pins

- Very similar method used by every microcontroller

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

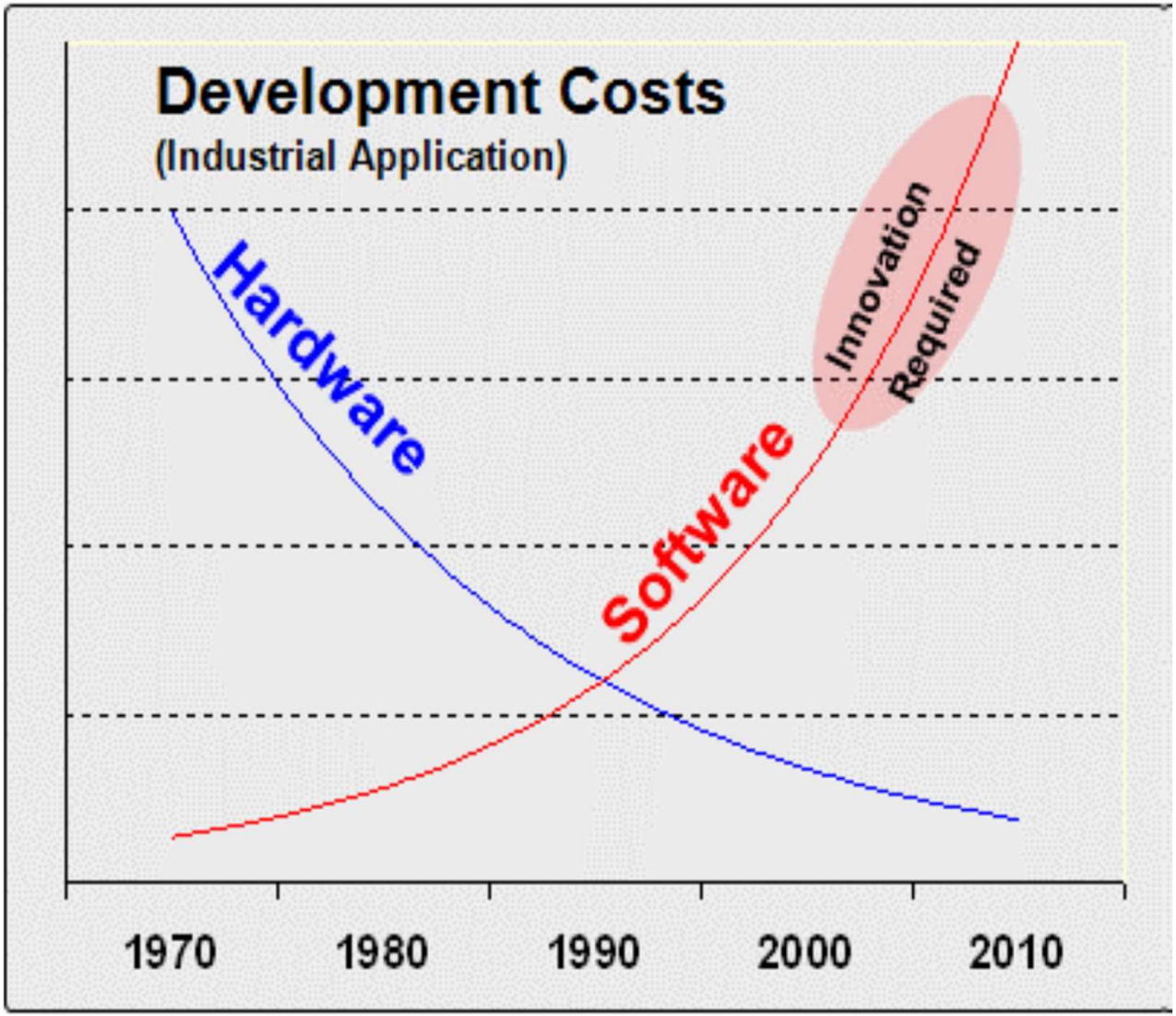
PINSEL0	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.0	GPIO Port 0.0	RD1	TXD3	SDA1	00
3:2	P0.1	GPIO Port 0.1	TD1	RXD3	SCL1	00
5:4	P0.2	GPIO Port 0.2	TXD0	AD0.7	Reserved	00
7:6	P0.3	GPIO Port 0.3	RXD0	AD0.6	Reserved	00
9:8	P0.4[1]	GPIO Port 0.4	I2SRX_CLK	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

CMSIS

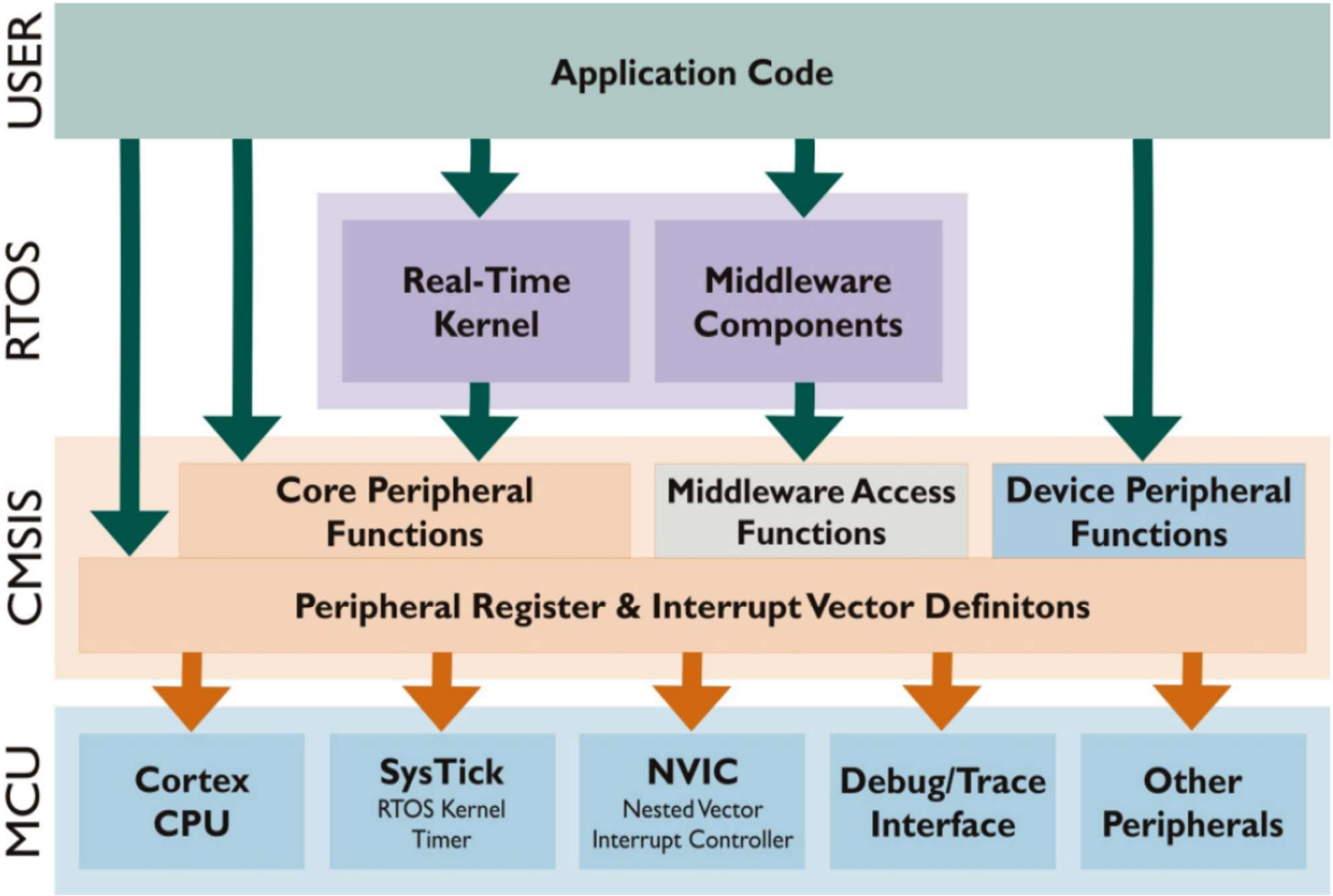
Cortex Microcontroller

Software Interface Standard

Software versus Hardware development costs



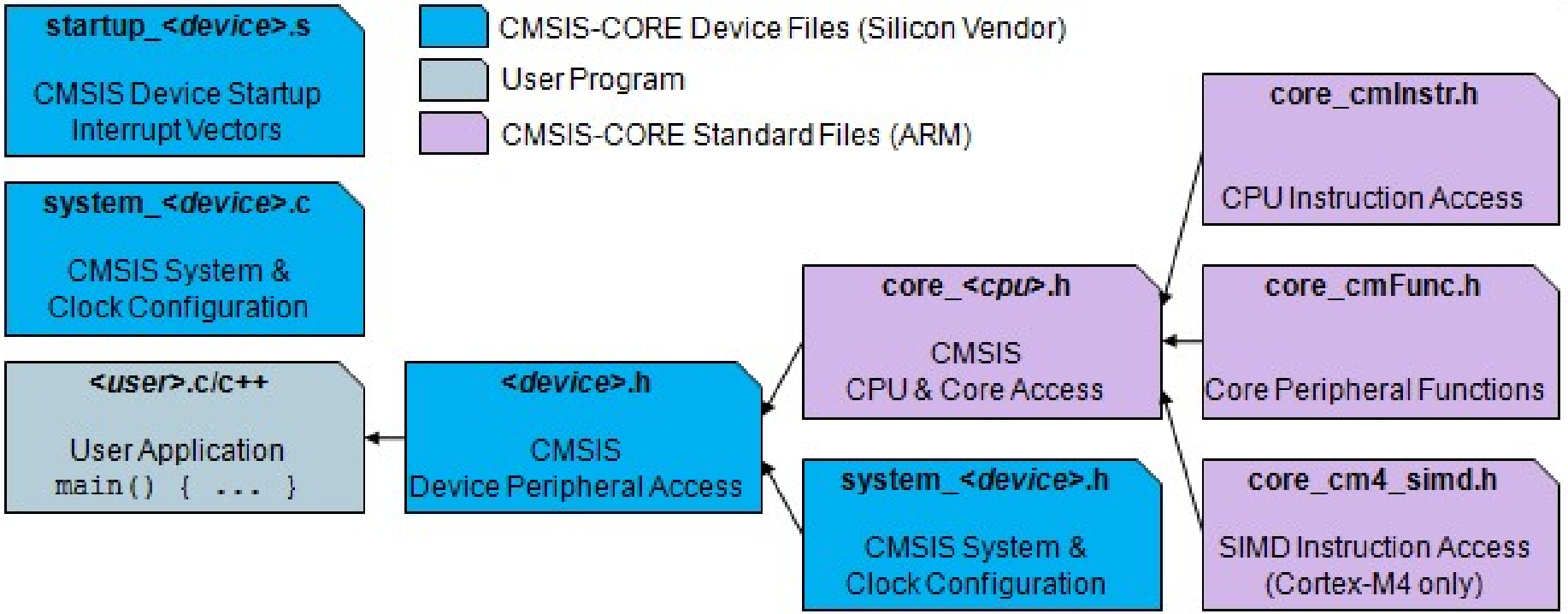
CMSIS architecture (v1.3)



CMSIS Core

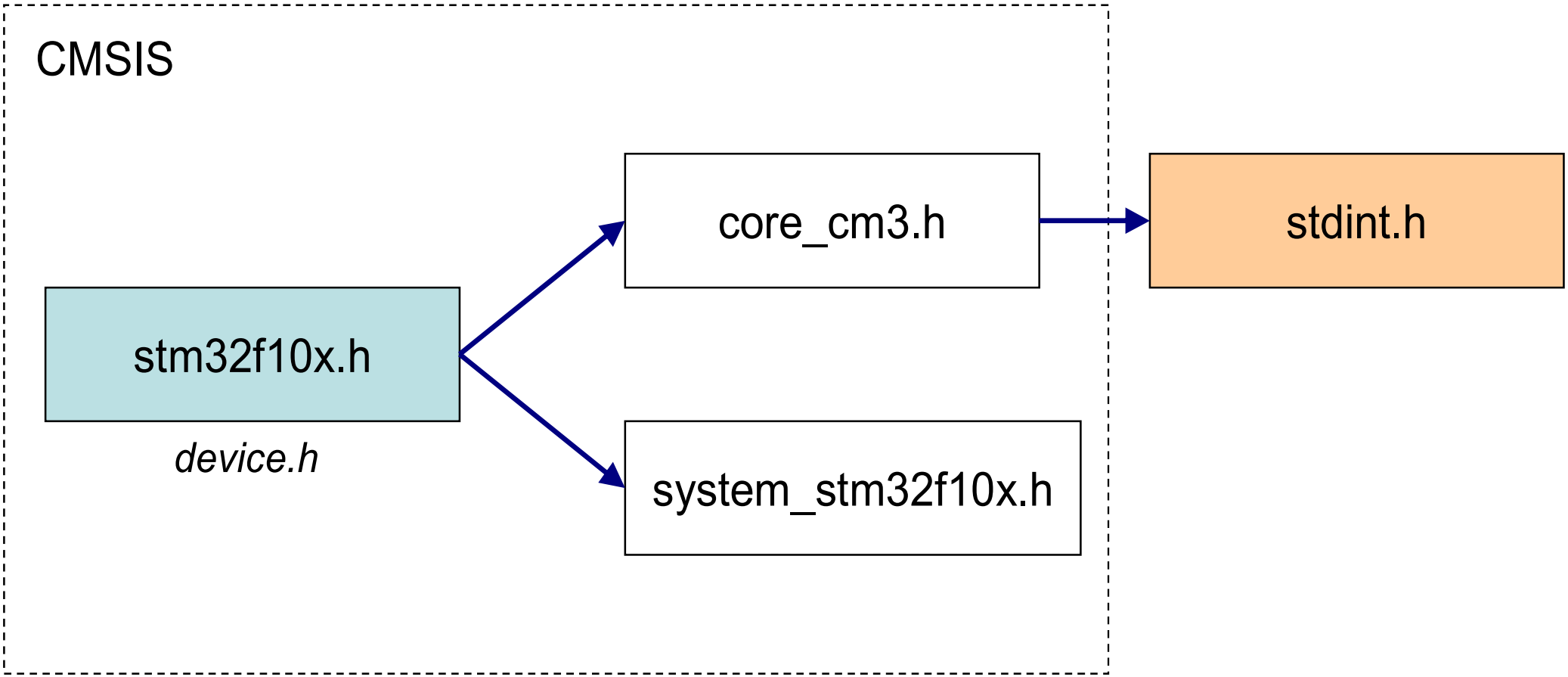
- **Hardware Abstraction Layer (HAL):** Standardized peripheral handling for al Cortex M core variant. Standard for register access and internal peripheral functions like SysTick, NVIC, MPU and FPU.
- **Exception handling:** Standardized names and function interfaces
- **Header file organization:** Naming conventions
- **System start:** Standardized SystemInit() function to cover microcontroller specific clock startups
- **Support for special instructions**
- Global variable for **system clock** frequncy

CMSIS core files



The device.h

- One and only include file for starting the system



The *startup_device* file

- Startup is compiler dependent
- This file contains the Startup Code
- The vector table is defined with ***weak*** pragmas

```
DCD       USART1_IRQHandler,    /* USART1 interrupt vector*/


#pragma weak USART1_IRQHandler = Default_Handler
```

The system_device.c

- Minimum services to start the microcontroller

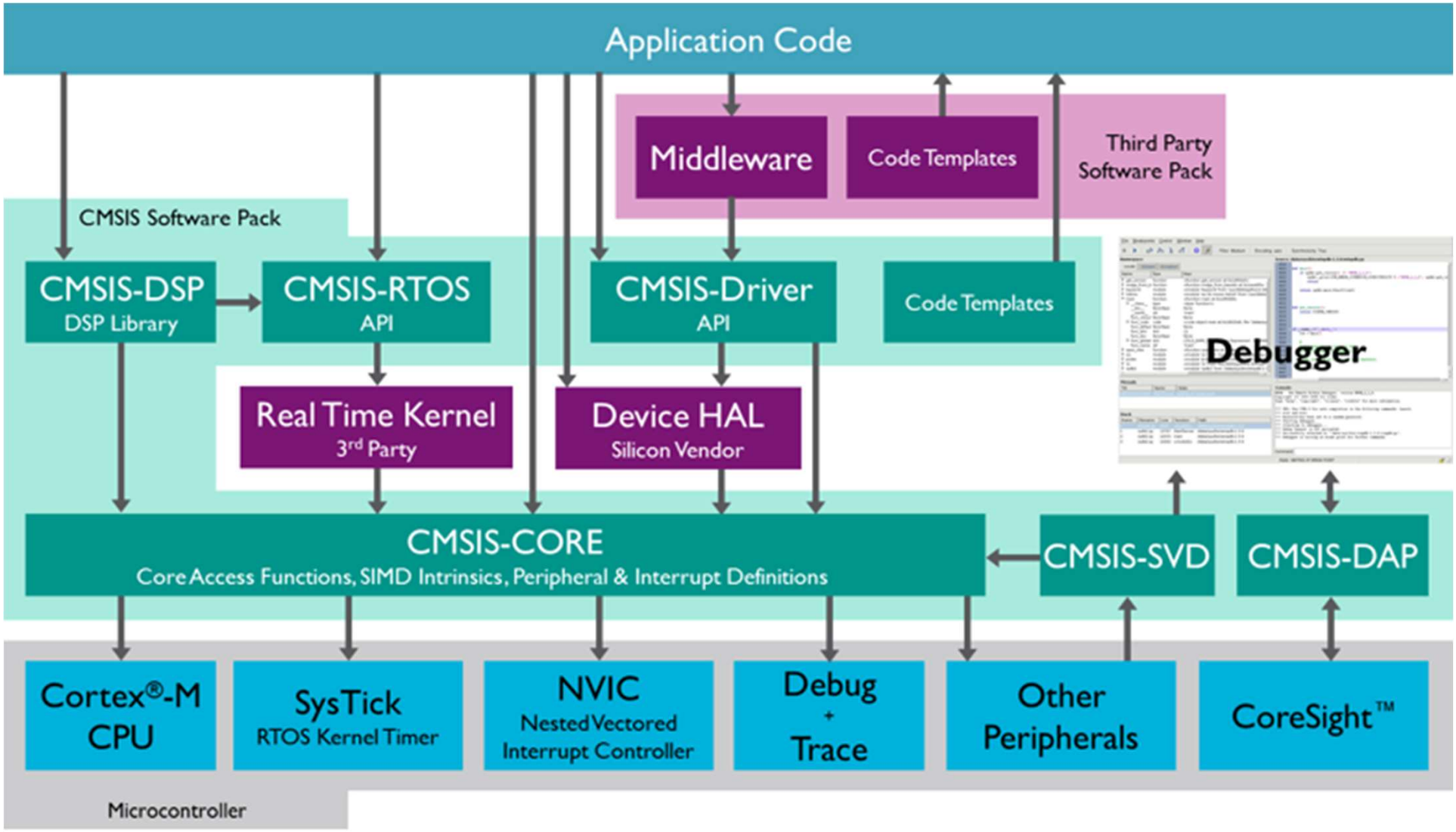
Function	description
void SystemInit (void)	Function to set up the system clocks
void SystemCoreClockUpdate(void)	Upgrading the system clock.

Variable	description
uint32_t SystemCoreClock	The current value of the system clock.

CMSIS coding guildlines

- Based on MISRA 2004
- Data types based on **<stdint.h>**
- All functions of *Core Peripheral Access Layer (**CPAL**) should be reentrant. There is no blocking code*
- All of the interrupt rutins should end with **_IRQHandler**
- Function use CamelCase naming
- Doxygen comments are used

CMSIS architecture improvements (v5)

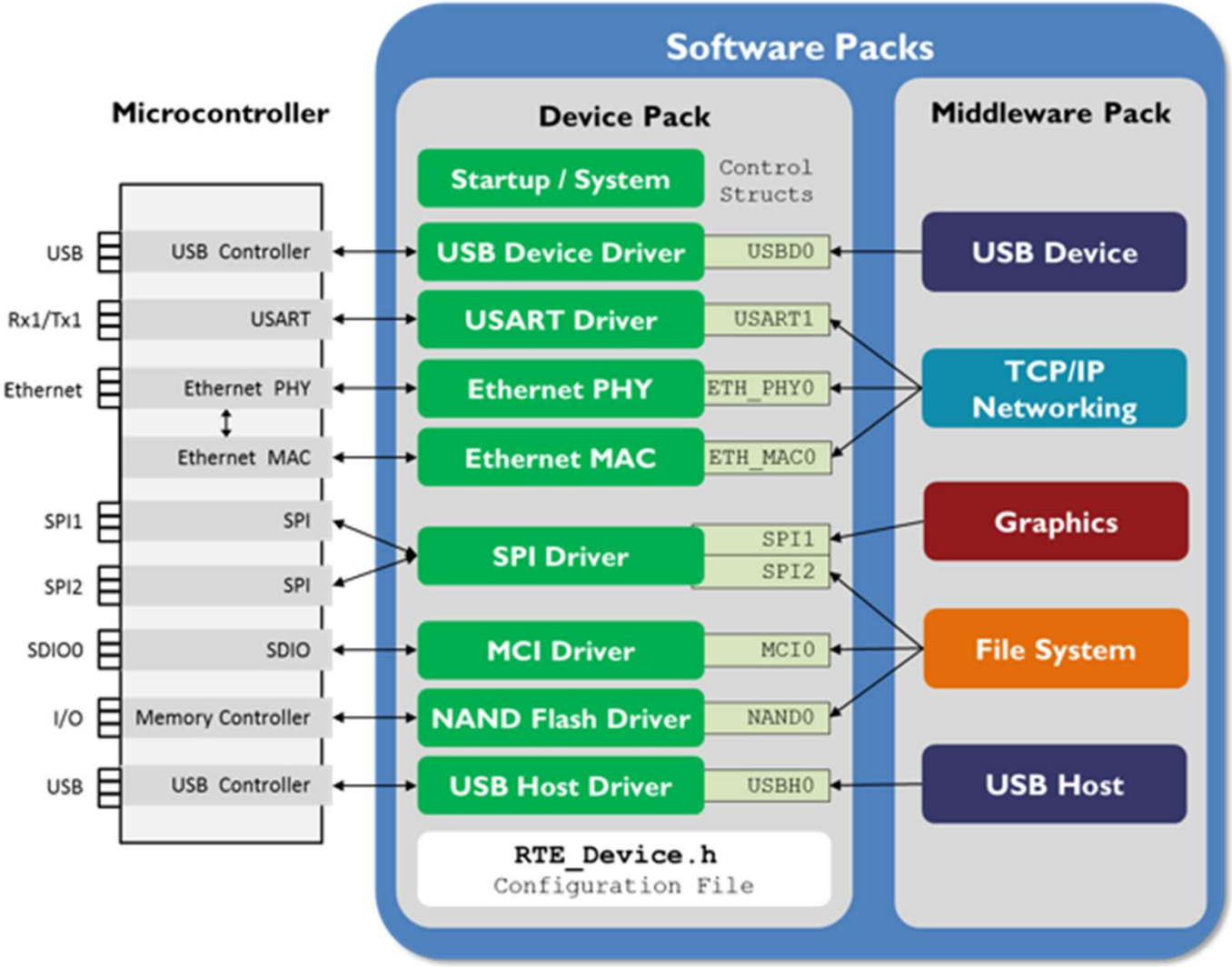


CMSIS DSP library

- Designed for Cortex M4 and M3, including assembly code utilizing the specialties of the instruction set
- Basic math functions
  - Vector multiplication, subtraction, adding
- Fast complex math functions
  - Cosinus, Sinus, Square root
  - Complex number handling
- Filter rutins
  - FIR, IIR

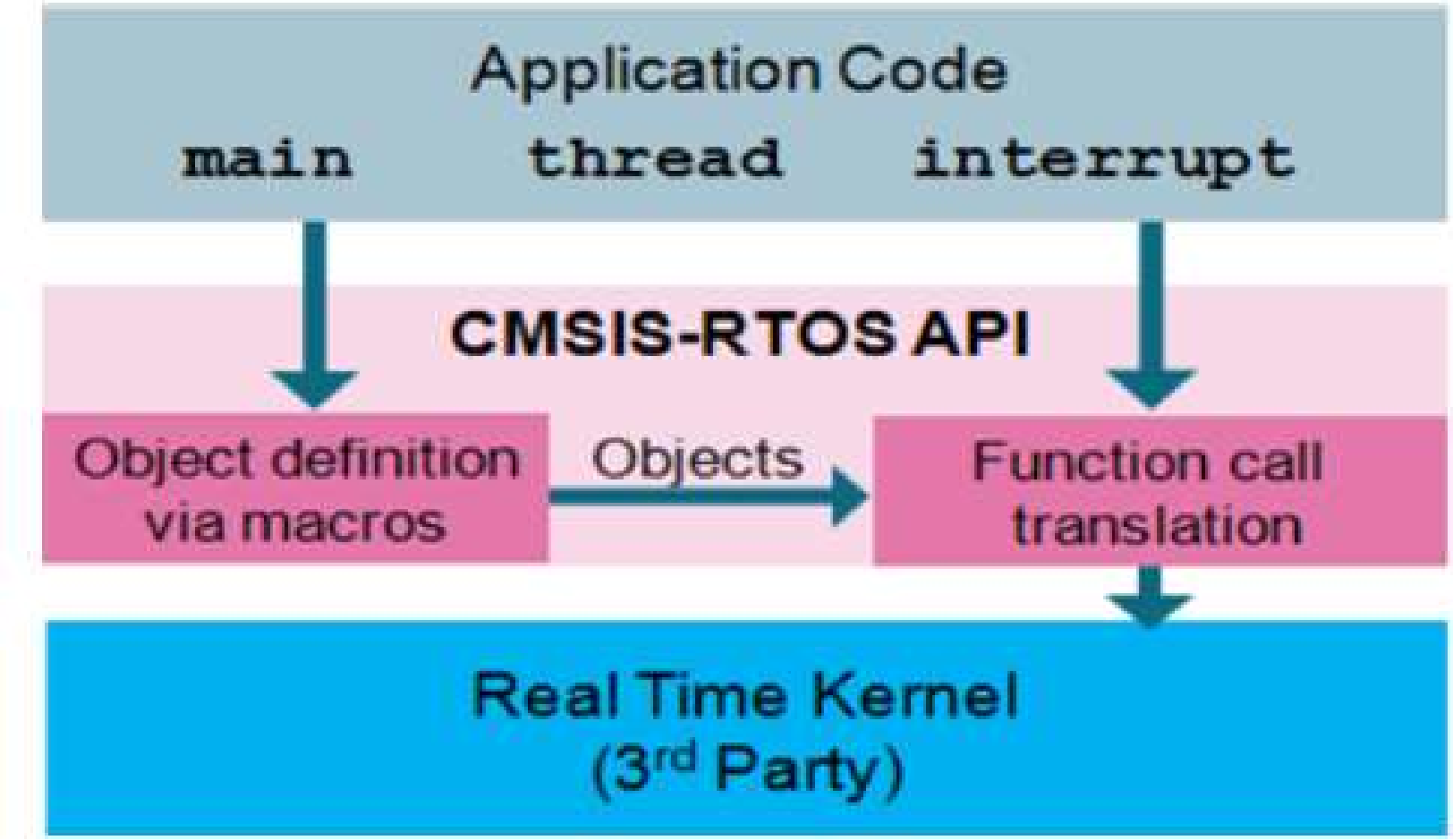
CMSIS Driver API

- Microcontroller vendor independent API



CMSIS RTOS

- Microcontroller independent RTOS abstraction



CMSIS RTOS

- Kernel handling functions

osStatus	**osKernelInitialize** (void) Initialize the RTOS Kernel for creating objects.
osStatus	**osKernelStart** (void) Start the RTOS Kernel.
int32_t	**osKernelRunning** (void) Check if the RTOS kernel is already started.
uint32_t	**osKernelSysTick** (void) Get the RTOS kernel system timer counter.

CMSIS RTOS

- Thread management functions

osThreadId	**osThreadCreate** (const **osThreadDef_t** *thread_def, void *argument) Create a thread and add it to Active Threads and set it to state READY.
osThreadId	**osThreadGetId** (void) Return the thread ID of the current running thread.
osStatus	**osThreadTerminate** (**osThreadId** thread_id) Terminate execution of a thread and remove it from Active Threads.
osStatus	**osThreadSetPriority** (**osThreadId** thread_id, **osPriority** priority) Change priority of an active thread.
osPriority	**osThreadGetPriority** (**osThreadId** thread_id) Get current priority of an active thread.
osStatus	**osThreadYield** (void) Pass control to next thread that is in state **READY**.

CMSIS RTOS

- General purpose timing functions

osStatus **osDelay** (uint32_t millisec)
Wait for Timeout (Time Delay).

osEvent **osWait** (uint32_t millisec)
Wait for Signal, Message, Mail, or Timeout.

- OS timer functionality

osTimerId **osTimerCreate** (const **osTimerDef_t** *timer_def, **os_timer_type** type, void *argument)
Create a timer.

osStatus **osTimerStart** (**osTimerId** timer_id, uint32_t millisec)
Start or restart a timer.

osStatus **osTimerStop** (**osTimerId** timer_id)
Stop the timer.

osStatus **osTimerDelete** (**osTimerId** timer_id)
Delete a timer that was created by **osTimerCreate**.