The Cortex-M3 Core for MCUs

DAMERGI Emir

2010

OUTLINE

- Introduction
- Cortex-M3
- Cortex-M3 Memory
- Cortex-M3 Interrupt Handling
- Cortex-M3 Specificities
- Cortex-M3 Based MCUs: The STM32F10 Family
- STM32 Programming

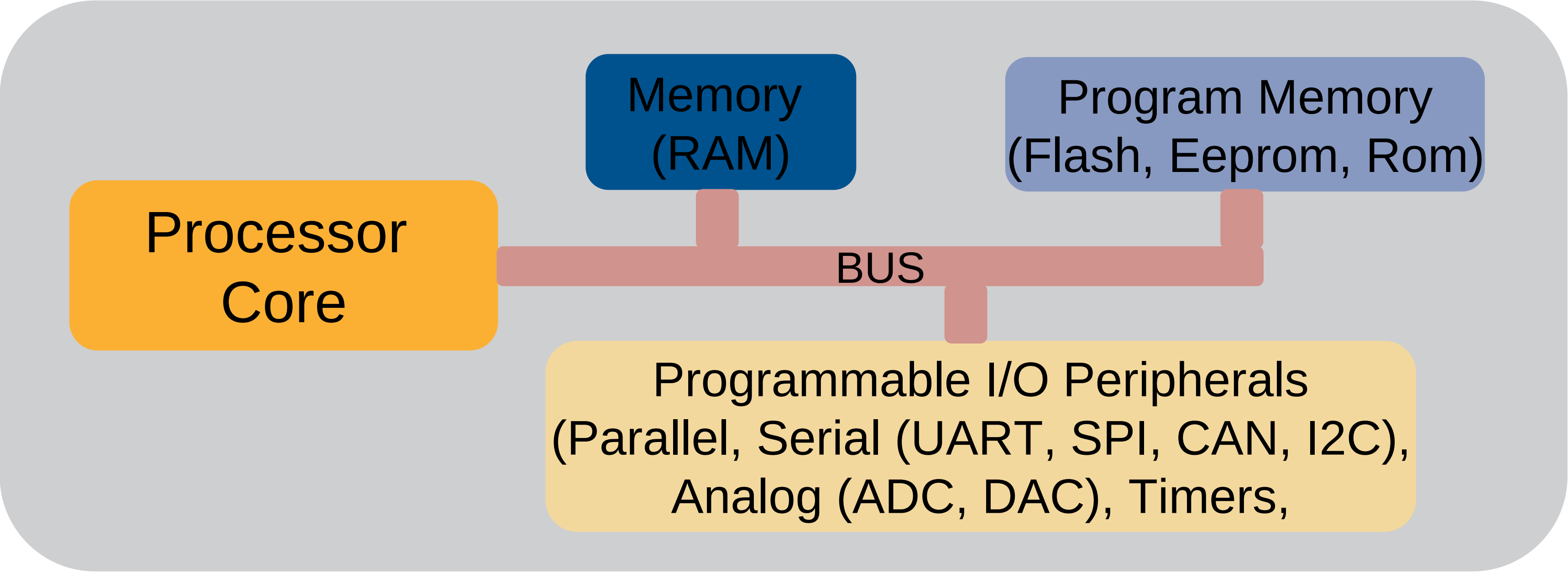
OUTLINE

- Introduction
- Cortex-M3
- Cortex-M3 Memory
- Cortex-M3 Interrupt Handling
- Cortex-M3 Specificities
- Cortex-M3 based MCUs: The STM32F10x Family
- STM32 Programming

Introduction: The Microcontroller

Microcontroller or MCU (MicroController Unit) :

a small computer on a single integrated circuit.

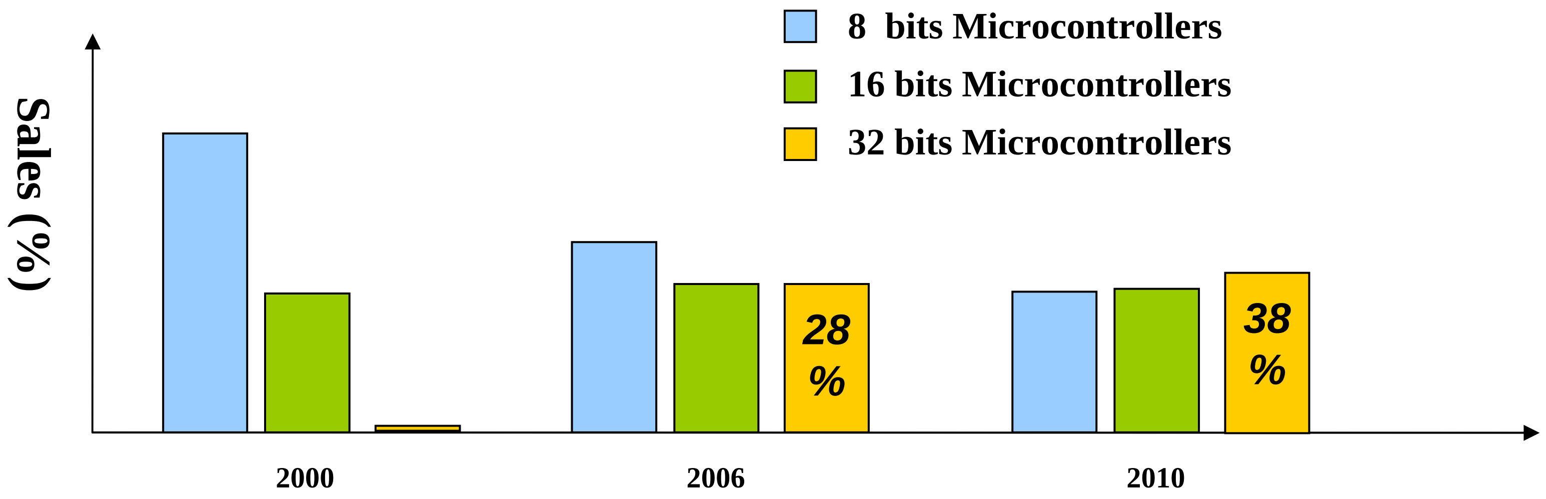

Memory (RAM)
Program Memory (Flash, Eeprom, Rom)
Processor Core
BUS
Programmable I/O Peripherals (Parallel, Serial (UART, SPI, CAN, I2C), Analog (ADC, DAC), Timers,

+ Size (High Integration)
+ Cost (<1$ for large qty)
+ Energy Consumption

→ Large Spectrum of embedded Applications ($26 Billions, 2007)

Introduction: Classification of MCUs

Classified according to the Register size (→Instruction size, Instruction Bus Width): **4** (obsolete), **8**, **16** and **32** bits



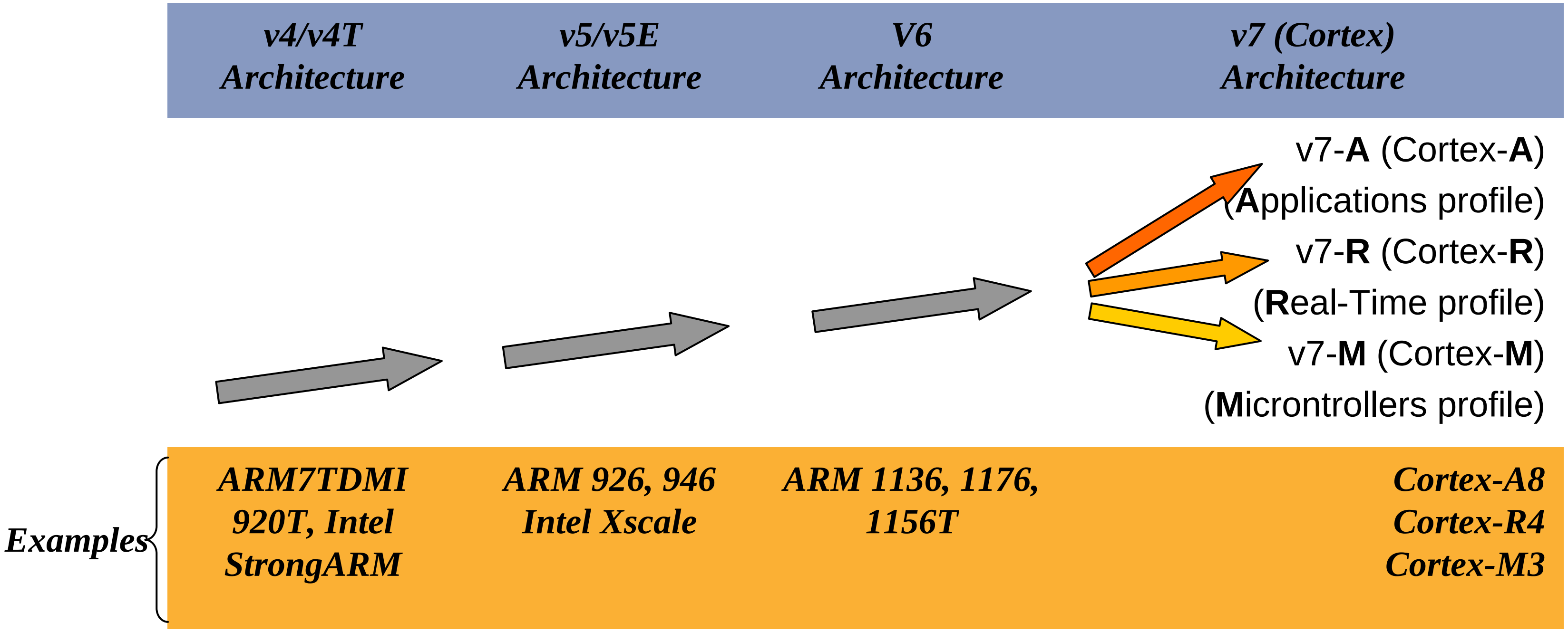
MCU market evolution

(influenced mainly by the cost of 32 Bits MCUs)

Introduction: ARM

Advanced Risc Machines: Created in 1990

Joint-venture: Apple Computer, Acorn Computer Group & VLSI Technology



	v4/v4T Architecture	*v5/v5E Architecture*	*V6 Architecture*	*v7 (Cortex) Architecture*
Examples	*ARM7TDMI 920T, Intel StrongARM*	*ARM 926, 946 Intel Xscale*	*ARM 1136, 1176, 1156T*	*Cortex-A8* *Cortex-R4* *Cortex-M3*

v7-**A** (Cortex-**A**)
(**A**pplications profile)

v7-**R** (Cortex-**R**)
(**R**eal-Time profile)

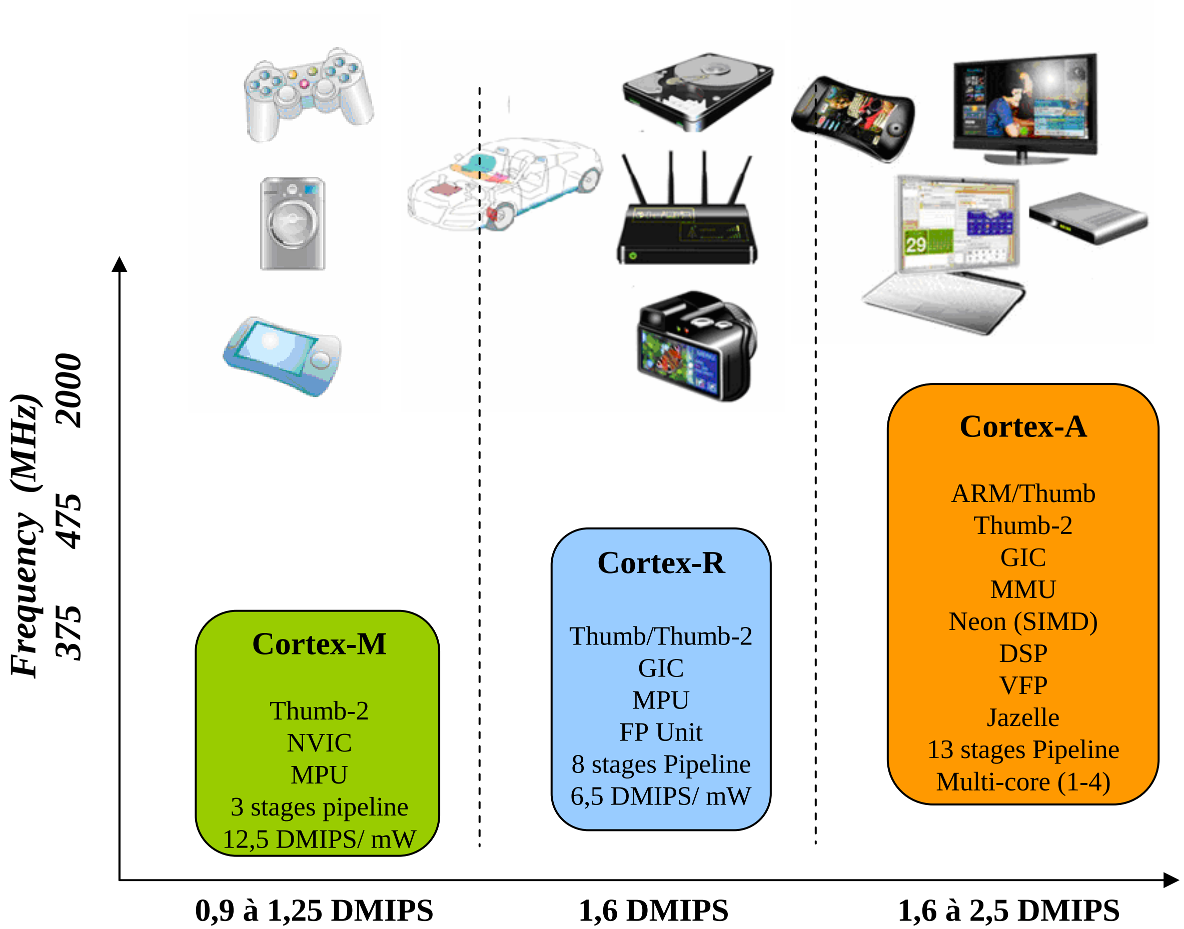
v7-**M** (Cortex-**M**)
(**M**icrontrollers profile)

Introduction: Cortex profiles & applications

Cortex-X N

X :Profile (A,R,M)

N: Performance level (0..9)



Cortex-M

Thumb-2
NVIC
MPU
3 stages pipeline
12,5 DMIPS/ mW

Cortex-R

Thumb/Thumb-2
GIC
MPU
FP Unit
8 stages Pipeline
6,5 DMIPS/ mW

Cortex-A

ARM/Thumb
Thumb-2
GIC
MMU
Neon (SIMD)
DSP
VFP
Jazelle
13 stages Pipeline
Multi-core (1-4)

OUTLINE

- Introduction
- Cortex-M3
  - Core features
  - Privileges & modes
  - Programmer’s Model
- Cortex-M3 Memory
- Cortex-M3 Interrupt Handling
- Cortex-M3 Specificities
- Cortex-M3 based MCUs: The STM32F10x Family
- STM32 Programming

Cortex-M3 Processor

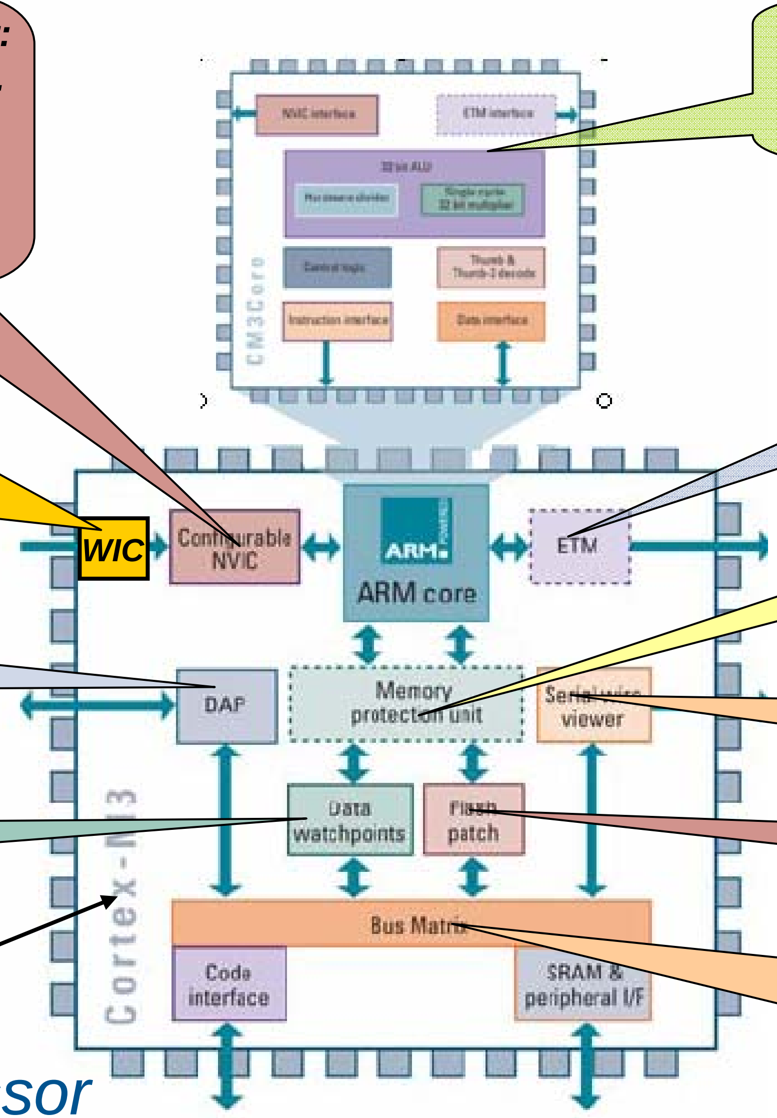
- Hierarchical processor integrating core and advanced system peripherals

Interrupt controller:
-1 to 240 interrupts.
- 256 Priority levels
- NMI
-SysTick
(Porting !!!)

Wakup Int. controller:
Wakeup from Sleep modes throught interrupts & exceptions

Debug Access port

4 Watch points



CM3 Core:
Harvard (Separate Busses)
32 Bits Register & ALUs.

Embedded Trace MacroCell

Optional Memory Protection Unit (8 regions)

Integrated Trace module: Low cost (2 wires)

8 Hardware Breakpoints

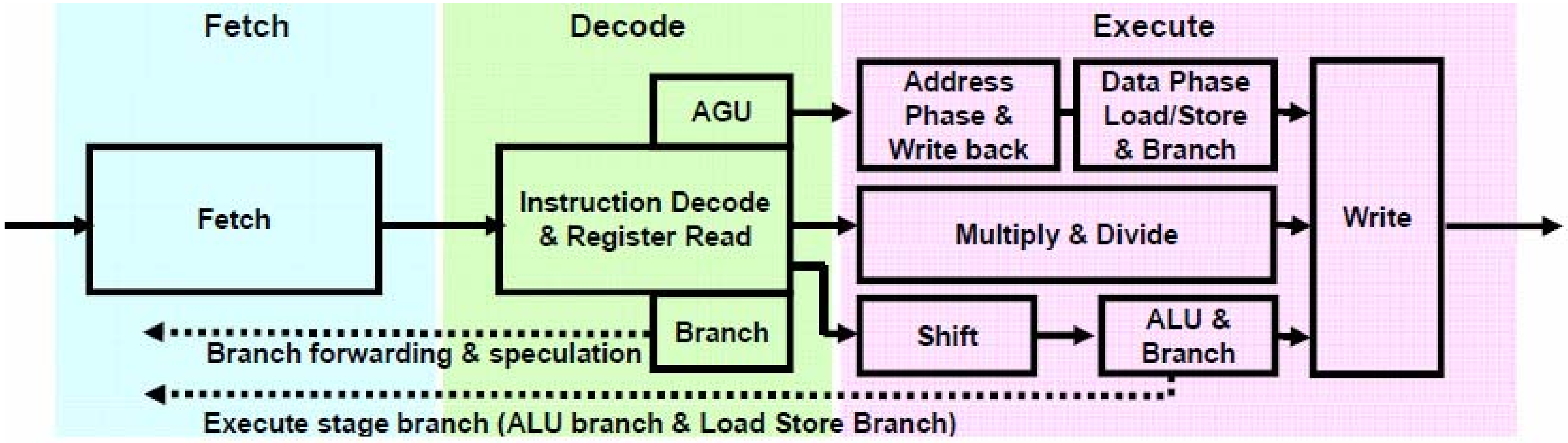
Multi layer Bus Matrix (Parallel transfers between core, memory, & peripherals

Cortex-M3 *Processor*

Cortex-M3 Core Features (1/4)

3-stage pipeline:

Fetch, Decode and & Execute (with static branch prediction)


Fetch
Decode
Execute
AGU
Address Phase & Write back
Data Phase Load/Store & Branch
Fetch
Instruction Decode & Register Read
Multiply & Divide
Write
Branch
Shift
ALU & Branch
Branch forwarding & speculation
Execute stage branch (ALU branch & Load Store Branch)

Simple adressing: linear 4GByte address space

- No data pages
  - No code pages

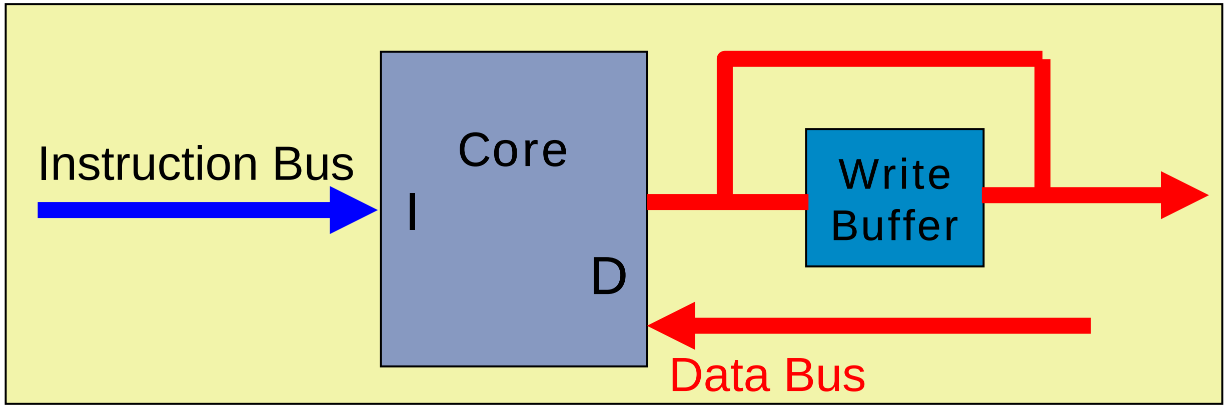
Cortex-M3 Core Features (2/4)

Harvard architecture

- Separate Instruction & Data buses allow parallel instruction fetching & data storage

Write Buffer

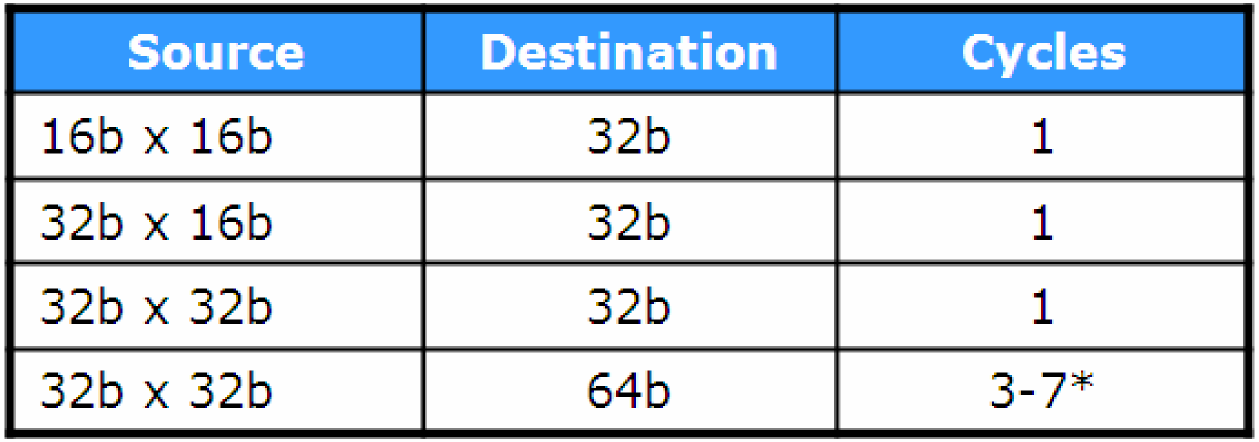
- Used to decouple writes to memory Data is placed in the buffer so the processor can continue execution
- Data is written to memory when possible



Cortex-M3 Core Features (3/4)

ALU with H/W divide and single cycle multiply

- Multiply



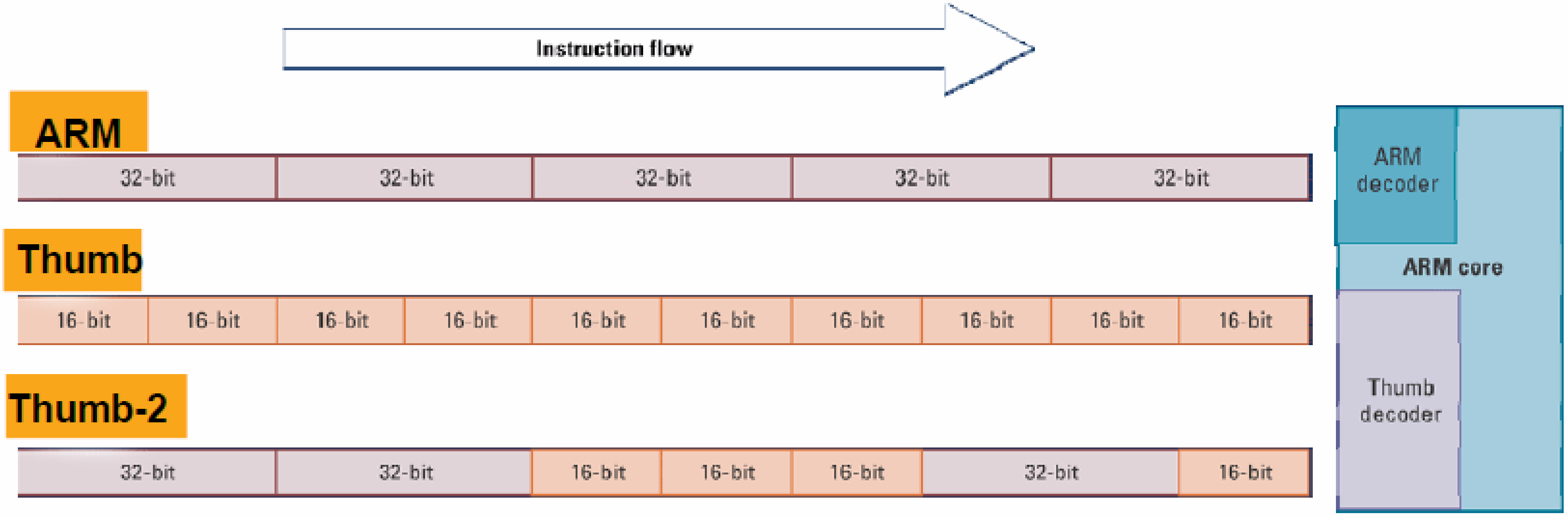
Source	Destination	Cycles
16b x 16b	32b	1
32b x 16b	32b	1
32b x 32b	32b	1
32b x 32b	64b	3-7*

- Divide : between 2 and 12 Cycles

Cortex-M3 Core Features (4/4)

Thumb®-2 and traditional Thumb

- Thumb-2 technology is an enhancement to the ARMv6 architecture
- Thumb-2 technology consists of:
  - New 16-bit Thumb instructions for improved program flow
  - New 32-bit Thumb instructions for improved performance and code size. One ‘new’ 32-bit instruction can replace multiple 16-bit opcodes



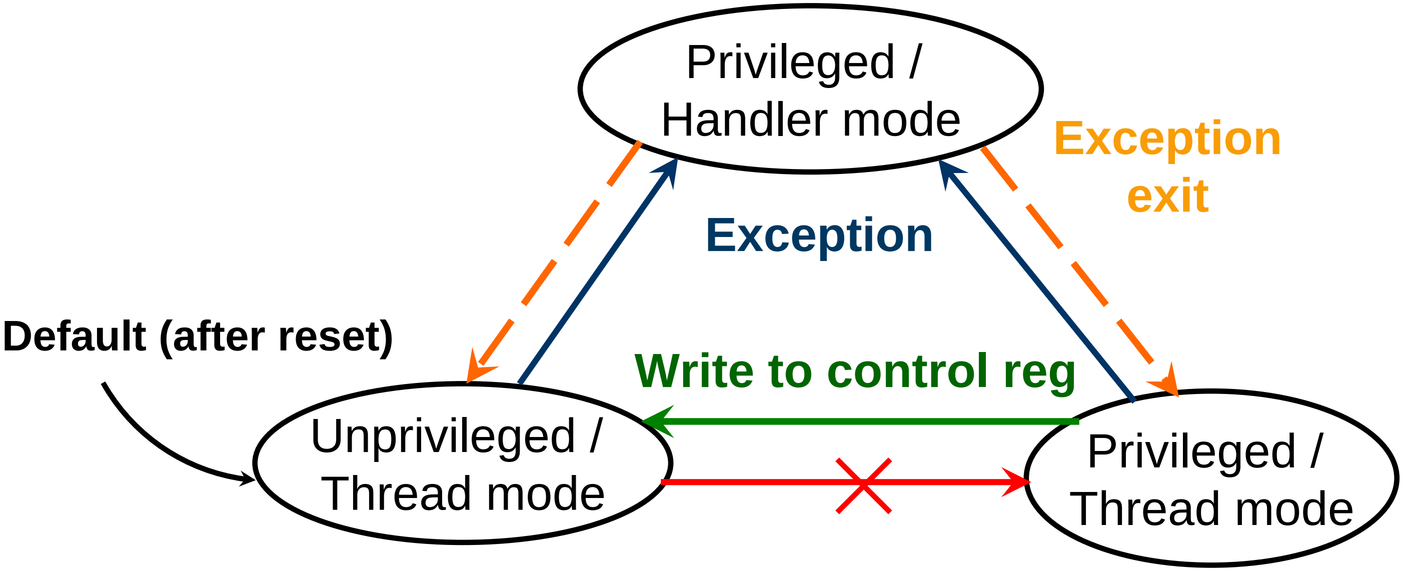
Cortex-M3 Core Privileges & modes (1/5)

		OPERATIONS	
		Privileged operation • Also called supervisor privilege • Active out of reset • Entered whenever an exception or interrupt is taken • Privileged operation allows access to all processor resources	**Unprivileged operation** • Also called user privilege • Limited access to processor resources.
Execution modes	**Handler mode** • An exception is being processed • An exception handler or ISR is executing • Could be an interrupt or a fault • Always privileged execution	**Exception Execution** (Main Stack: MSP)	✕
	Thread mode • No exception is being processed • Normal code is executing • Could be privileged or unprivileged	**Main Program** (Main Stack:MSP)	**Main Program** (Process Stack:PSP)

Cortex-M3 Core Privileges & modes (2/5)

- The privilege level and the mode are defined in the 2-bit **Control register.**

- Programs can’t write to control register in Unprivileged/Thread mode.



•Transition from ***Unprivileged/Thread*** to privileged/Thread mode **not allowed**.

→ Must pass through ***Privileged/Handler*** mode by calling the SVC (System Service Call) which generates an exception + Write to ctrl reg

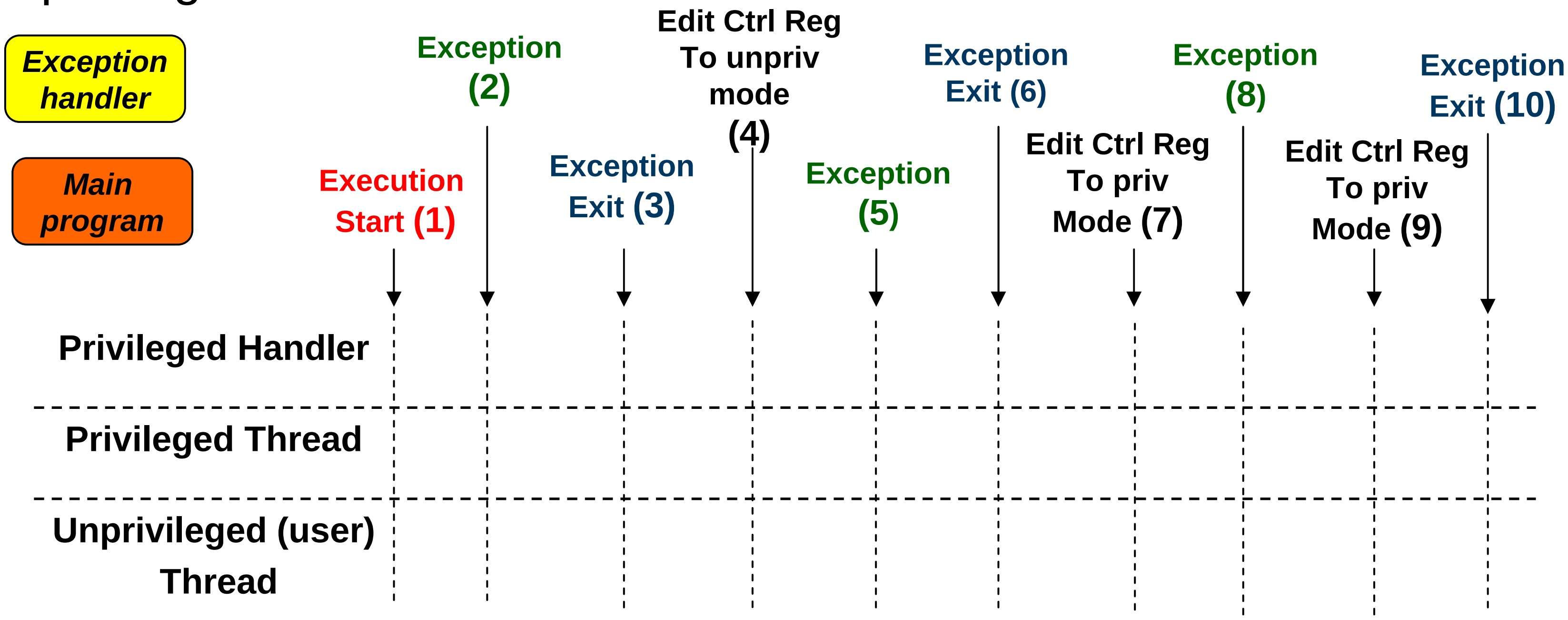
Cortex-M3 Core Privileges & modes (3/5)

Exercise: **Switching of Operation Mode by Programming the Control Register or by Exceptions**

complete the following diagram indicating fo earch event:

- If the processor is executing the main program or the execption handler(ISR).

- The privileg level and execution mode.



Cortex-M3 core: Programmer’s model

Register Set

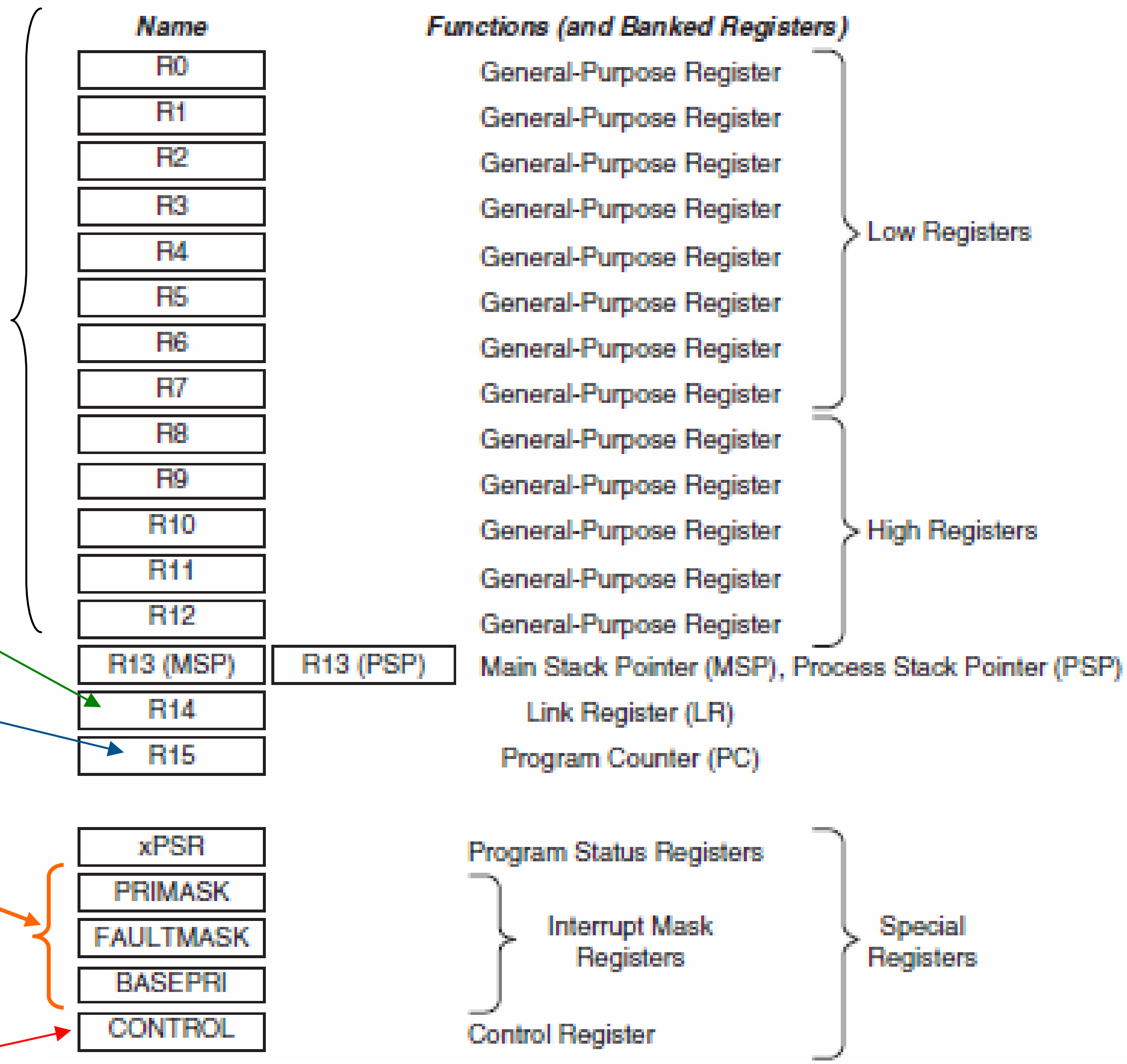
All registers used Similarly for all operations:
- Low registers accessed by all instructions.
- High registers accessed only by thumb2 instructions.

Contains return address after ISR

Contains next instruction address

To mask or enable:
- NMI
- Hard Fault exceptions
- interrupts of specified priority values

Used to define the privilege level and the stack pointer selection



Name		*Functions (and Banked Registers)*	
R0		General-Purpose Register	Low Registers
R1		General-Purpose Register	
R2		General-Purpose Register	
R3		General-Purpose Register	
R4		General-Purpose Register	
R5		General-Purpose Register	
R6		General-Purpose Register	
R7		General-Purpose Register	
R8		General-Purpose Register	High Registers
R9		General-Purpose Register	
R10		General-Purpose Register	
R11		General-Purpose Register	
R12		General-Purpose Register	
R13 (MSP)	R13 (PSP)	Main Stack Pointer (MSP), Process Stack Pointer (PSP)	
R14		Link Register (LR)	
R15		Program Counter (PC)	
xPSR		Program Status Registers	Special Registers
PRIMASK		Interrupt Mask Registers	
FAULTMASK			
BASEPRI			
CONTROL		Control Register	

Cortex-M3 Core: Programmer’s Model

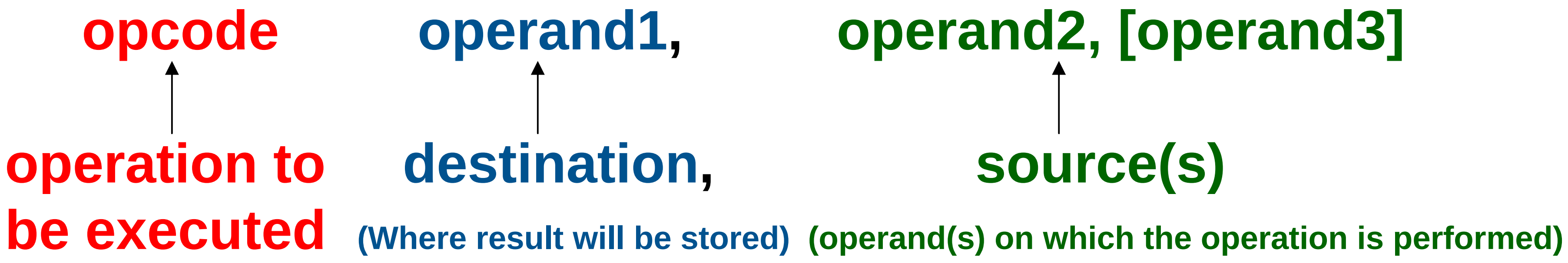
Instructions model

- RISC Processor with Load and Store Architecture:

- Operands must be loaded into CPU registers (R0- R12).

- The Operation must be performed on these registers.

- The used instruction formatting is:



opcode **operand1,** **operand2, [operand3]**

operation to be executed **destination,** **source(s)**

(Where result will be stored) **(operand(s) on which the operation is performed)**

- Example:

add R4, R1, R3 → (R4 = R1 + R3)

Cortex-M3 Core: Programmer's Model

UAL (*Unified Assembler Language)*

•Some of the operations can be handled by either a 16-bit Thumb Instruction or a 32-bit Thumb-2 Instruction.

→ **With UAL , it is possible to specify which Instruction to use by adding suffixes.**

***Example:* To perform the addition R0 = R0 +1**

- ADD.N R0, #1   ;use 16-bit instruction (N = Narrow)
- ADD.W R0, #1   ;use 32-bit instruction (W = Wide)

- If no suffix is given, the 16-bit Thumb code is choosen by default to get a smaller size.

- In most cases, applications will be coded in 'C', and the 'C' compilers will use 16-bit instructions if possible

Cortex-M3 Core: Programmer’s Model

Data Transfer Instructions

- ***Immediate data value to register***

  MOV Dest_Reg, #Value (exple: MOV R6, # 16): R6 ←16

- ***Between Registers***

  MOV Dest_reg, Src_reg (example: MOV R8, R3) : R3 → R8

- ***Between memory and register***

  LDR Rd, [Rn, # offset] ; *Read word from memory location Rn+offset*

(expl: MOV R3, # 0x100

LDR R8,[R3,# 0x010]

STR Rd, [Rn, # offset] ; *Store word to memory location Rn+offset*

Rq: Any register can be used as a pointer to data structures/arrays

Cortex-M3 Core: Programmer's Model

Logic Instructions

Bitwise AND:

```
AND Rd, Rn                  ; Rd = Rd & Rn

AND.W Rd, Rn,#immed   ; Rd = Rn & #immed

AND.W Rd, Rn, Rm          ; Rd = Rn & Rm
```

Bitwise OR :

The same formats with the opcode ORR

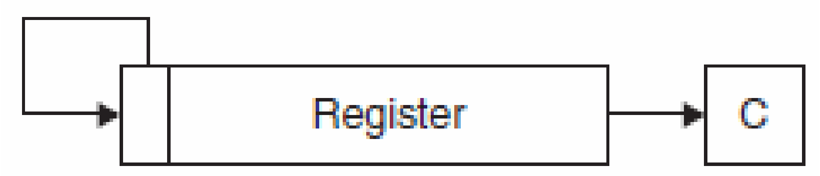
Bitwise Exclusive OR

The same formats with the opcode EOR

Cortex-M3 Core: Programmer’s Model

Shift and rotate Instructions

Arithmetic shift right

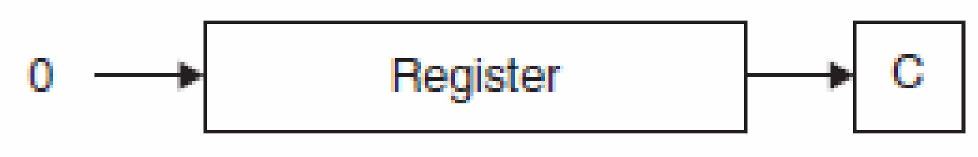


ASR Rd, Rn,#immed ***; Rd = Rn >> immed***

ASR Rd, Rn ***; Rd = Rd >> Rn***

ASR.W Rd, Rn, Rm ***; Rd = Rn >> Rm***

Logical shift Right



LSR Rd, Rn,#immed***; Rd = Rn >> immed***

LSR Rd, Rn ***; Rd = Rd >> Rn***

LSR.W Rd, Rn, Rm ***; Rd = Rn >> Rm***

Cortex-M3 Core: Programmer’s Model

Arithmetic Instructions (1)

Addition:

```
ADD Rd, Rn, Rm      ; Rd = Rn + Rm ADD operation

ADD Rd, Rm          ; Rd = Rd + Rm

ADD Rd, #immed      ; Rd = Rd + #immed

ADDW Rd, Rn,#immed ; Rd = Rn + #immed (immed : 12 bits)
```

Substraction:

```
SUB Rd, Rn, Rm          ; Rd = Rn – Rm

SUB Rd, #immed          ; Rd = Rd – #immed

SUB Rd, Rn,#immed   ; Rd = Rn –#immed
```

Cortex-M3 Core: Programmer’s Model

Arithmetic Instructions (2)

Multiply instructions (32-bit result)

```
MUL Rd, Rm                 ; Rd = Rd * Rm
MUL.W Rd, Rn, Rm           ; Rd = Rn * Rm
```

Multiply instructions for unsigned values (64-bit result)

```
UMULL RdLo, RdHi, Rn, Rm   ; {RdHi,RdLo} = Rn * Rm
UMLAL RdLo, RdHi, Rn, Rm   ;{RdHi,RdLo} += Rn * Rm
```

Use SMULL & SMLAL for 32-bit signed multiply

Cortex-M3 Core: Programmer’s Model

Arithmetic Instructions (3)

Unsigned divide

```
UDIV Rd, Rn, Rm            ; Rd = Rn /Rm
```

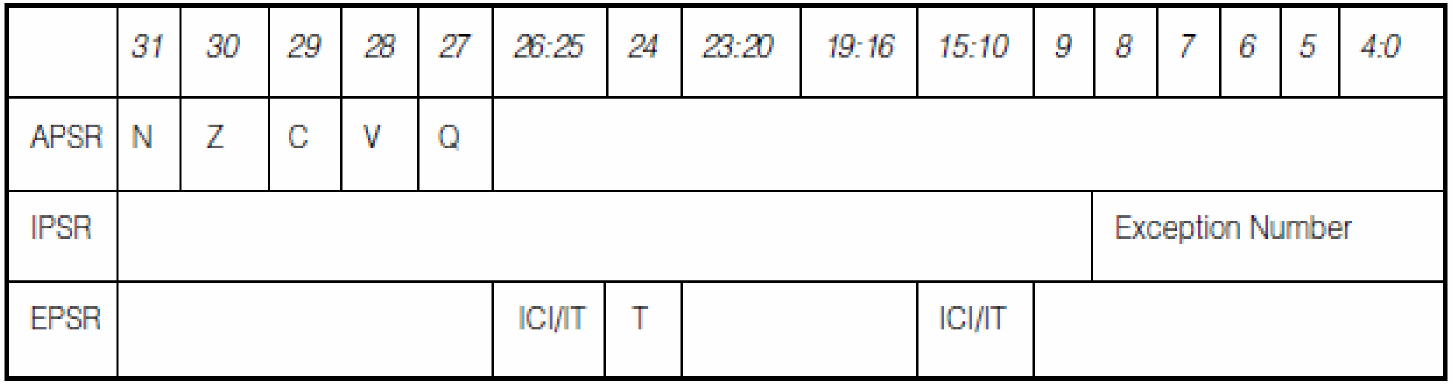
signed divide

```
SDIV Rd, Rn, Rm            ; Rd = Rn /Rm
```

Cortex-M3 Core: Programmer’s Model

Special Registers

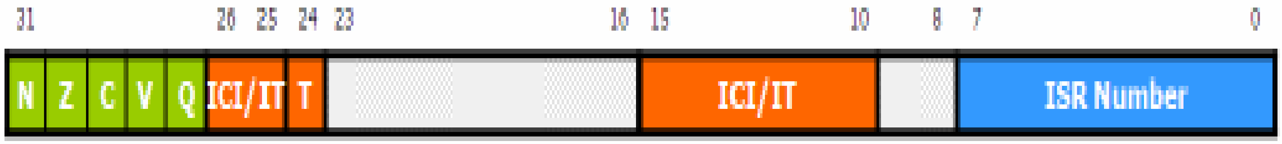
3 unaccessible status Regiters (Read Only)



		31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
Application PSR	APSR	N	Z	C	V	Q											
Interrupt PSR	IPSR												Exception Number				
Execution PSR	EPSR						ICI/IT	T			ICI/IT						

Combined together into one unique read/write register

Combined PSR: xPSR



Cortex-M3 Core: Programmer’s Model

- **xPSR – Program Status Register**

31					26 25	24	23 16	15 10	8	7 0
N	Z	C	V	Q	ICI/IT	T	▨	ICI/IT	▨	ISR Number

- Allows access to APSR, EPSR and IPSR special purpose registers
- xPSR stored on stack during exceptions
- Condition code flags
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU Operation carried out
  - V = ALU Operation overflowed
  - Q = Saturated math overflow
- IT/ICI Bits
  - Contain IF-THEN base condition code and Interrupt Continue information
- ISR Number
  - ISR contains information on which exception was pre-empted

Cortex-M3 Core: Programmer’s Model

Access to xPSR

- **Read Processor status register (xPSR)**

```
MRS R0, PSR          ; Read into R0
```

- **Write value into status register**

```
MSR PSR, R1   ; Write from R1
```

The MRS and MSR instructions are used for access to
All special registers (PSR, Control, PriMASK, FaultMASK, BasePRI)

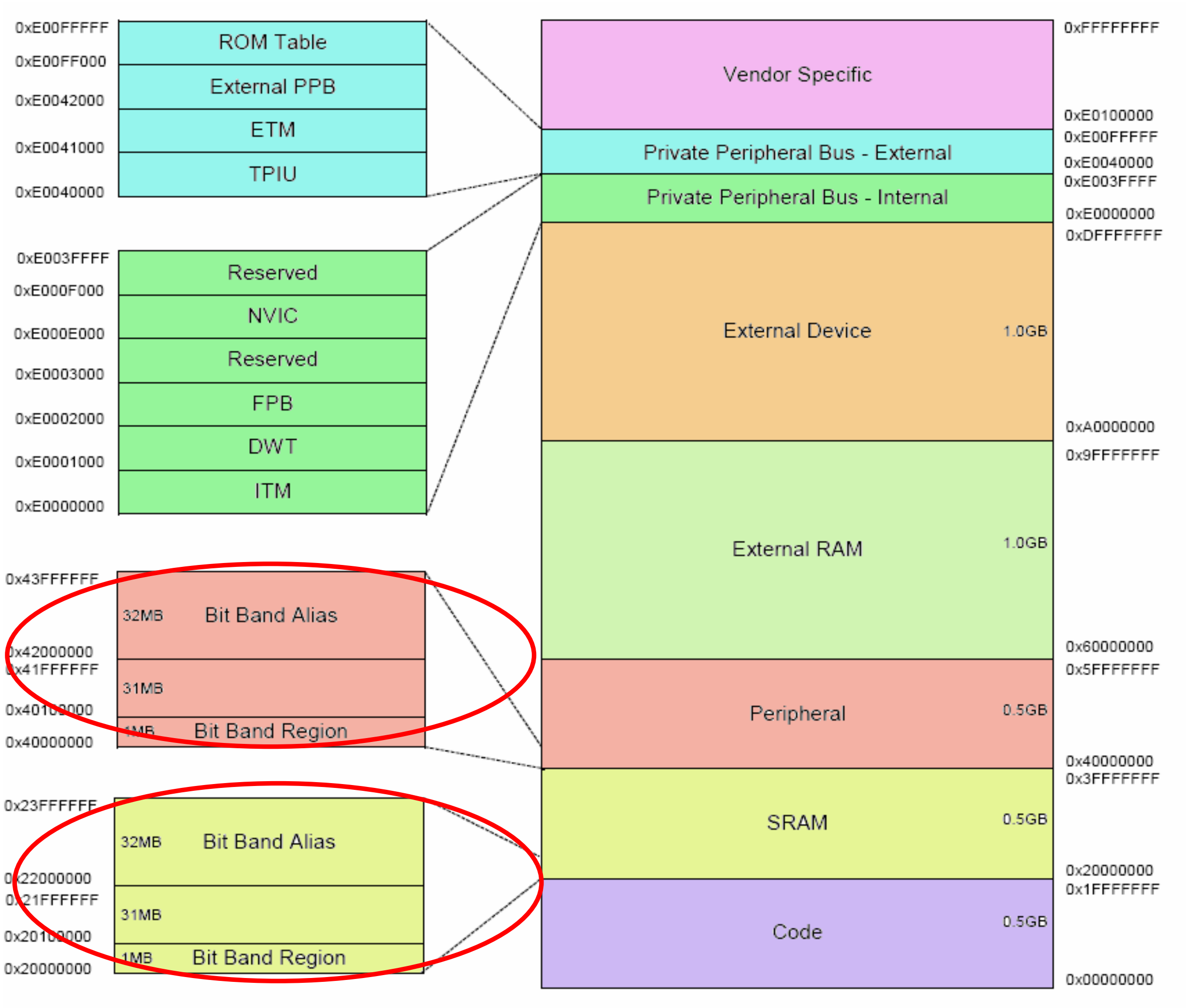
OUTLINE

- Introduction
- Cortex-M3
- Cortex-M3 Memory
  - Memory Map
  - Bit Banding
  - Unaligned Data Access
- Cortex-M3 Interrupt Handling
- Cortex-M3 Specificities
- Cortex-M3 based MCUs: The STM32F10x Family
- STM32 Programming

Cortex-M3: Memory

Memory map

4 GByte linear Memory space



0xE00FFFFF
ROM Table
0xE00FF000
External PPB
0xE0042000
ETM
0xE0041000
TPIU
0xE0040000

0xE003FFFF
Reserved
0xE000F000
NVIC
0xE000E000
Reserved
0xE0003000
FPB
0xE0002000
DWT
0xE0001000
ITM
0xE0000000

0x43FFFFFF
32MB Bit Band Alias
0x42000000
0x41FFFFFF
31MB
0x40100000
1MB Bit Band Region
0x40000000

0x23FFFFFF
32MB Bit Band Alias
0x22000000
0x21FFFFFF
31MB
0x20100000
1MB Bit Band Region
0x20000000

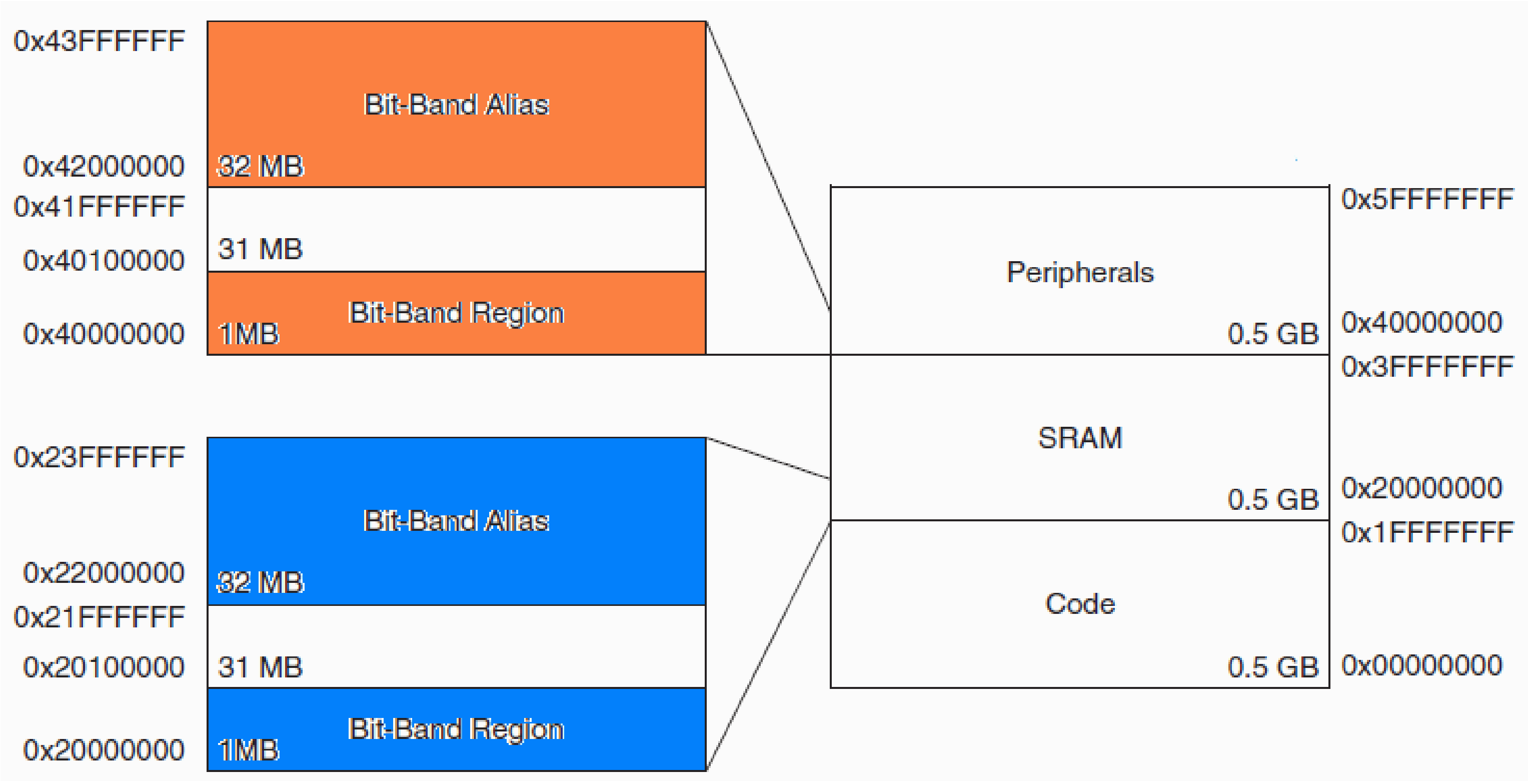
0xFFFFFFFF
Vendor Specific
0xE0100000
0xE00FFFFF
Private Peripheral Bus - External
0xE0040000
0xE003FFFF
Private Peripheral Bus - Internal
0xE0000000
0xDFFFFFFF
External Device 1.0GB
0xA0000000
0x9FFFFFFF
External RAM 1.0GB
0x60000000
0x5FFFFFFF
Peripheral 0.5GB
0x40000000
0x3FFFFFFF
SRAM 0.5GB
0x20000000
0x1FFFFFFF
Code 0.5GB
0x00000000

Cortex-M3: Memory

Bit Banding

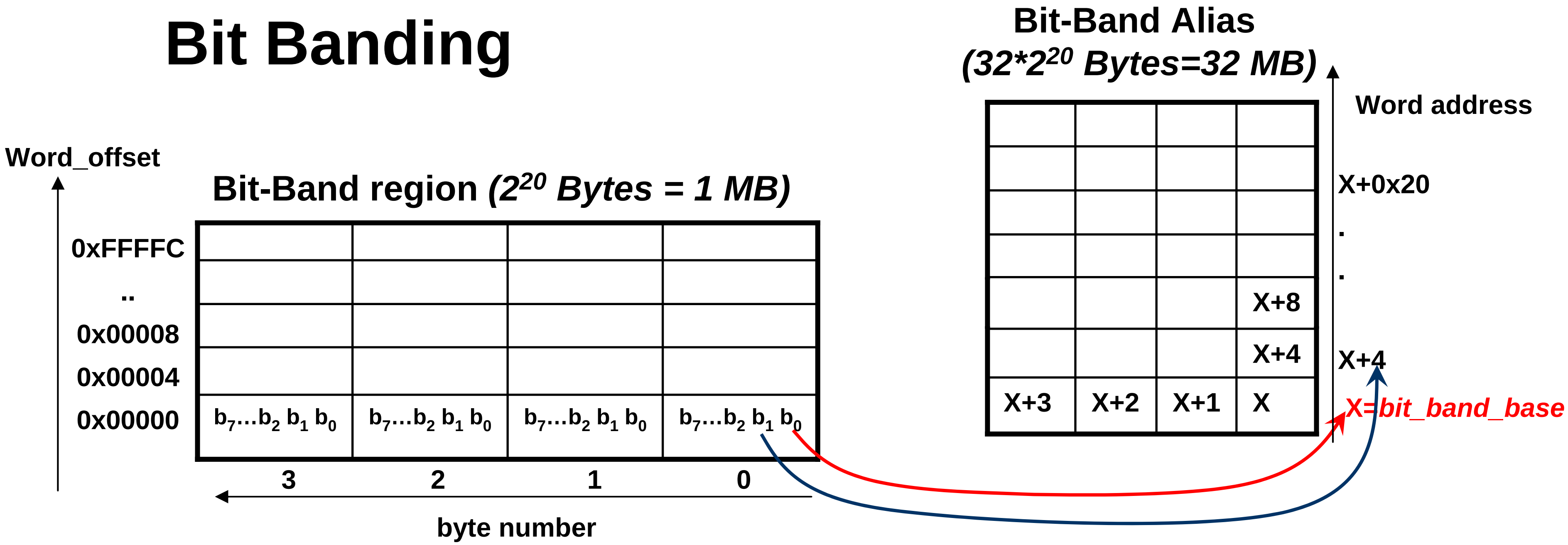
- Each bit of the bit band region (1MB) is mapped to one 32 bit address (32 MB Bit Band Alias = « Virtual zone »).

- Each bit in the Bit Band region can be accessed separately through the corresponding 32 bits register in the alias region.



Cortex-M3: Memory

Bit Banding



byte_offset = Word_offset + byte_number

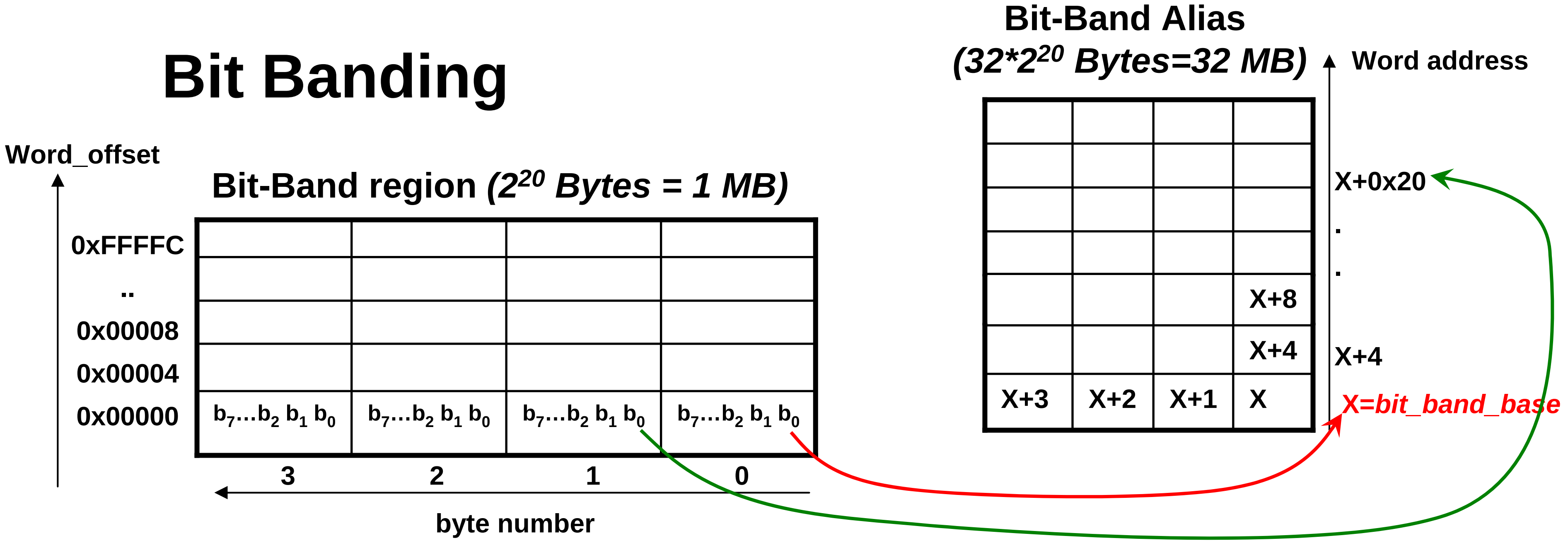
To access the first bit of the band region, we must go through the word address X of the alias region.

To access the second bit of the band region, we must go through the word address X+4 of the alias region. And so on ……

The word address in the Bit-Band Alias increases at a value of 4 when switching to the next bit.

Cortex-M3: Memory

Bit Banding



Bit-Band Alias

($32*2^{20}$ Bytes=32 MB)

Word address

X+0x20

.

.

X+4

X=bit_band_base

X+8

X+4

X+3 X+2 X+1 X

Word_offset

Bit-Band region *(2^{20} Bytes = 1 MB)*

0xFFFFC

..

0x00008

0x00004

0x00000

$b_7…b_2\ b_1\ b_0$ $b_7…b_2\ b_1\ b_0$ $b_7…b_2\ b_1\ b_0$ $b_7…b_2\ b_1\ b_0$

3 2 1 0

byte number

byte_offset = Word_offset + byte_number

To access the first bit of the band region, we must go through the word address X of the alias region.

To access the first bit of the next byte in the band region, we must go through the word address X+0x20 (32) of the alias region. And so on ……

→ The word address in the Bit-Band Alias increases at a value of 0x20 (32) when switching to the next byte.

Cortex-M3: Memory

Bit Banding

- **Bit Banding formula (mapping the bit to the register) is:**

bit_word_addr **=** ***bit_band_base*** **+ (*****byte_offset*** **x 32) + (*****bit_number*** **× 4)**

where:

bit_word_addr: is the address of the word in the alias memory region that maps to the targeted bit.

bit_band_base is the starting address of the alias region (0x22000000 or 0x42000000)

byte_offset is the number of the byte in the bit-band region that contains the targeted bit

bit_number is the bit position of the targeted bit(0-7).

Cortex-M3: Memory

Bit Banding

Example -1:

How to map bit 15 of the byte located at address 0x40000303 in the alias region.

Cortex-M3: Memory

Bit Banding

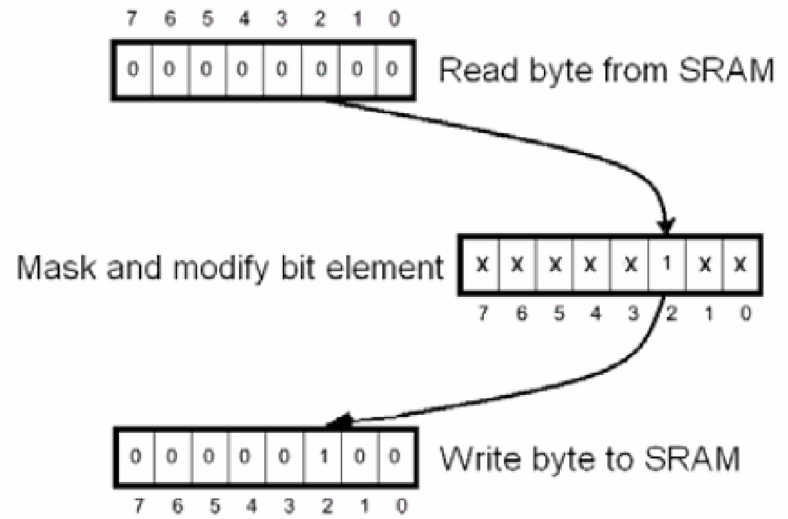
Example -2:

Which bit is accessed when reading the word at address 0x22006008

Cortex-M3: Memory

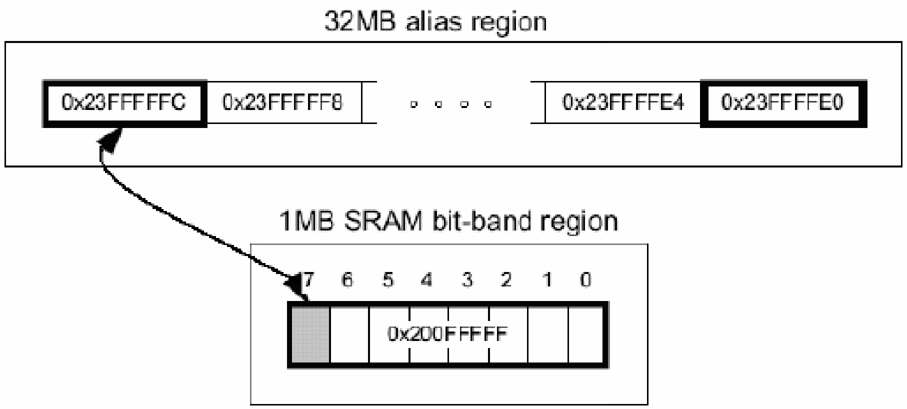
Bit Banding

Atomic Bit access Allows fast and atomic data access



```
LDR R0,=0x200FFFFF ; Setup address
MOV R2, #0x4        ; Setup data
LDR R1, [R0]        ; Read
ORR R1, R2          ; Modify bit
STR R1, [R0]        ; Write back result
```

Traditional bit manipulation method



```
LDR R0,=0x23FFFFFC ; Setup address
MOV R1, #0x1        ; Setup data
STR R1, [R0]        ; Write
```

Direct, single cycle access with bit banding

Cortex-M3: Memory

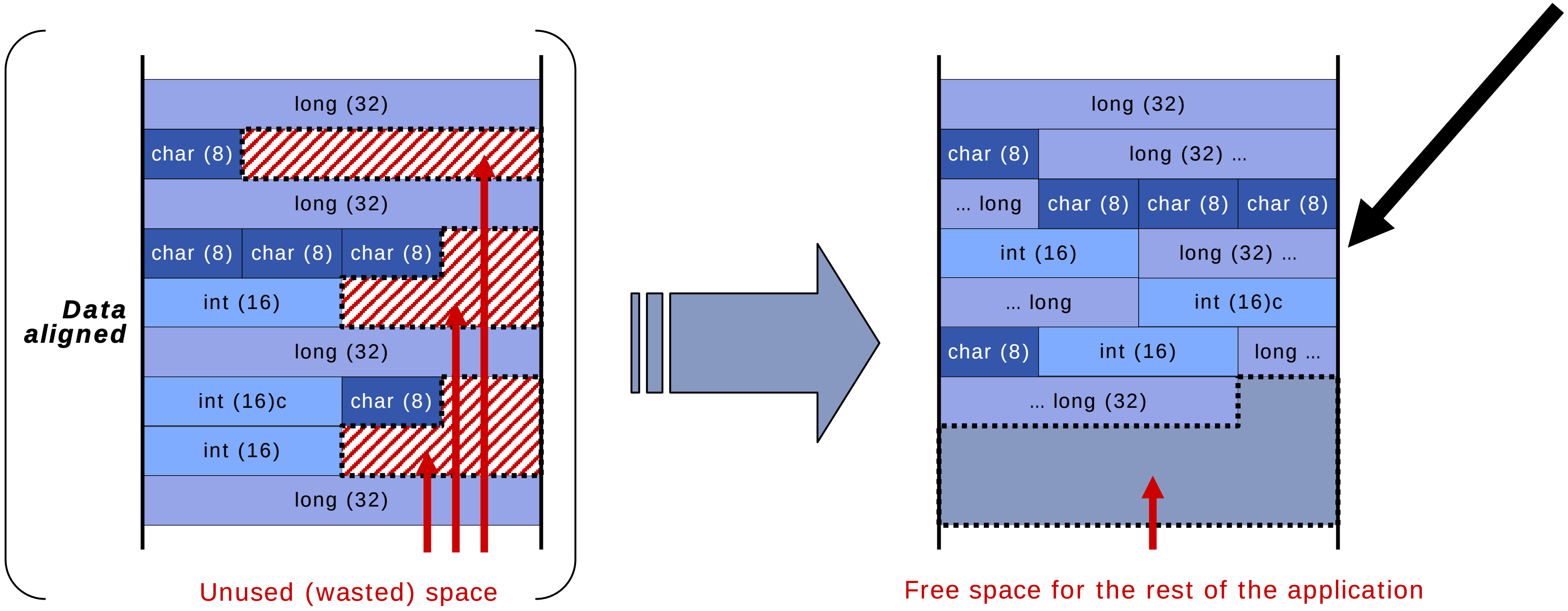
Unaligned Data Access

No support for unaligned data:

The whole Data must reside in the same memory address

Support for unaligned data:

The Data can be split into many memory locations



Unused (wasted) space

Free space for the rest of the application

Reduces SRAM Memory Requirements

Cortex-M3: Memory

Unaligned Data Access

Unaligned transfers are not supported in:

- Stack operations (PUSH/POP).
- Bit-band operations

When an unaligned transfer takes place,it is broken into separate transfers and as a result it takes more clock cycles for a single data access and might not be good for situations in which high performance is required.

Unaligned access:

Better memory use but **Performance degradation**

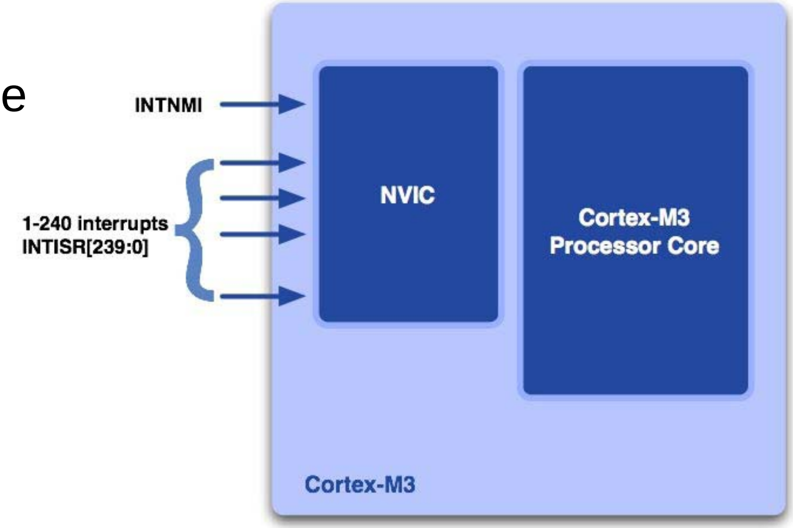
OUTLINE

- Introduction
- Cortex-M3
- Cortex-M3 Memory
- Cortex-M3 Interrupt Handling
  - Exception Vector Table
  - Tail Chaining/ Preemption / Late Arrival
- Cortex-M3 Specificities
- Cortex-M3 based MCUs: The STM32F10x Family
- STM32 Programming

Cortex-M3 : Interrupt Handling (1/9)

Very low latency interrupt processing

- Interruptible LDM(load multipe)/STM(stored multiple) for low interrupt latency
- Automatic processor state save and restore provides low latency ISR entry and exit allows handler to be written entirely in ‘C’



The Nested Vector Interrupt Controller:

- 1-240 Interrupts + NMI

Priority

- A programmable priority level Of 0-31.
- Grouping of priorities values into Group priority and sub-Priority fields

Stack operations

- The processor automatically stacks its state on exception entry and unstacks it on exit with no instruction overhead.

Interupt handling

- Late arrival / Tail chaining/ Preemption
- Priority boosting / inversion

Cortex-M3: Interrupt Handling (2/9)

Exceptions

•**Asynchronous** Exceptions = Interrupts

Generated by hardware peripherals (when enabled):

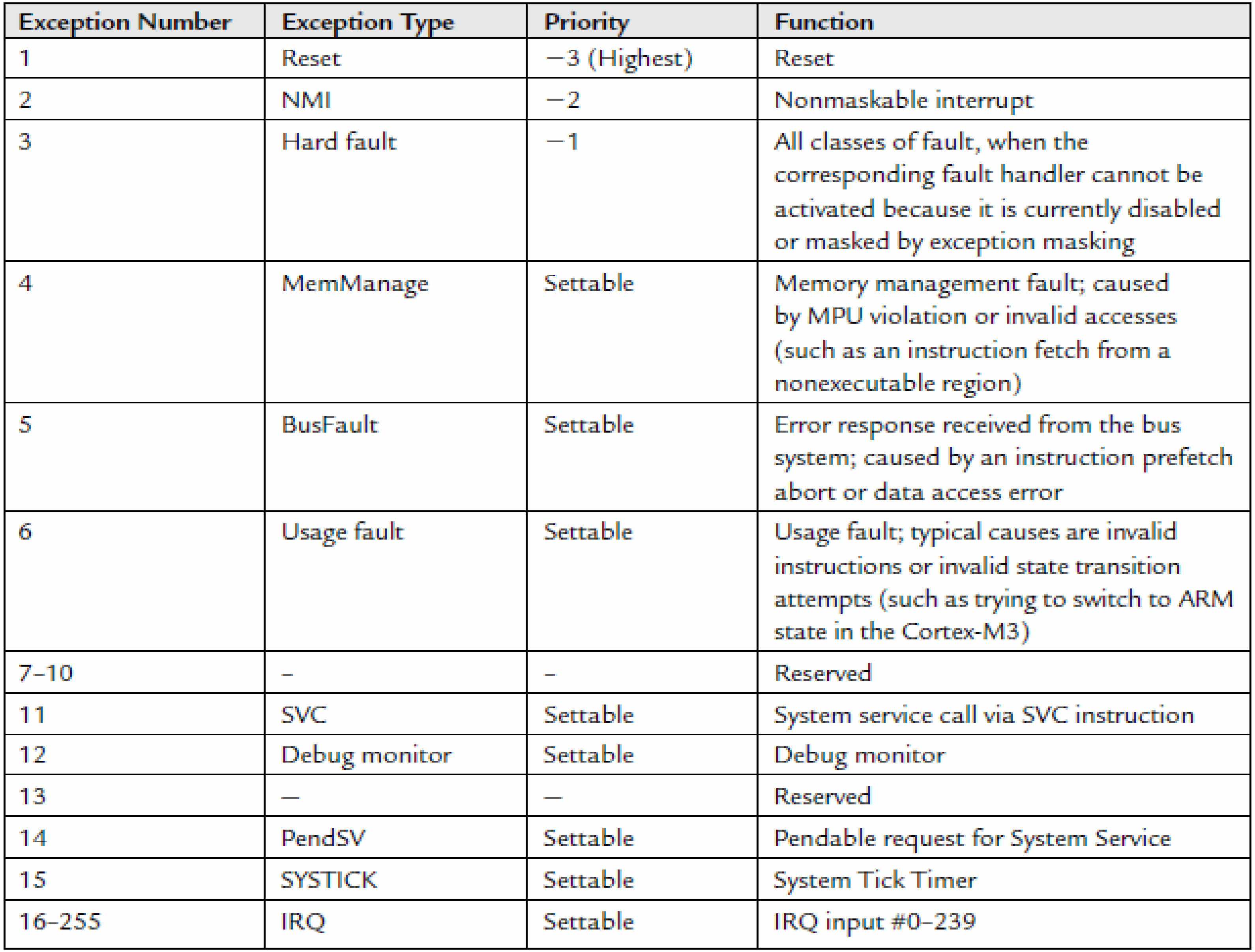
- Signal toggle (P I/O ports).
- Data receive (Serial peripherals)
- A/D conversion finished (DAC)
- …….

•**Synchronous** Exceptions = Exceptions

Generated after instruction execution errors such as:

- unauthorized Memory region access.
- Overflow.
- Divide by 0.
- …..

Cortex-M3: Interrupt Handling (3/9)



Exception Number	Exception Type	Priority	Function
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Nonmaskable interrupt
3	Hard fault	−1	All classes of fault, when the corresponding fault handler cannot be activated because it is currently disabled or masked by exception masking
4	MemManage	Settable	Memory management fault; caused by MPU violation or invalid accesses (such as an instruction fetch from a nonexecutable region)
5	BusFault	Settable	Error response received from the bus system; caused by an instruction prefetch abort or data access error
6	Usage fault	Settable	Usage fault; typical causes are invalid instructions or invalid state transition attempts (such as trying to switch to ARM state in the Cortex-M3)
7–10	–	–	Reserved
11	SVC	Settable	System service call via SVC instruction
12	Debug monitor	Settable	Debug monitor
13	—	—	Reserved
14	PendSV	Settable	Pendable request for System Service
15	SYSTICK	Settable	System Tick Timer
16–255	IRQ	Settable	IRQ input #0–239

Cortex-M3: Interrupt Handling (4/9)

Exceptions Vector Table

To determine the starting address of the exception handler, a **vector table** mechanism is used.

✓ It contains the starting addresses of exception handlers.

✓ The address is fetched via the instruction port allowing register stacking to take place in parallel

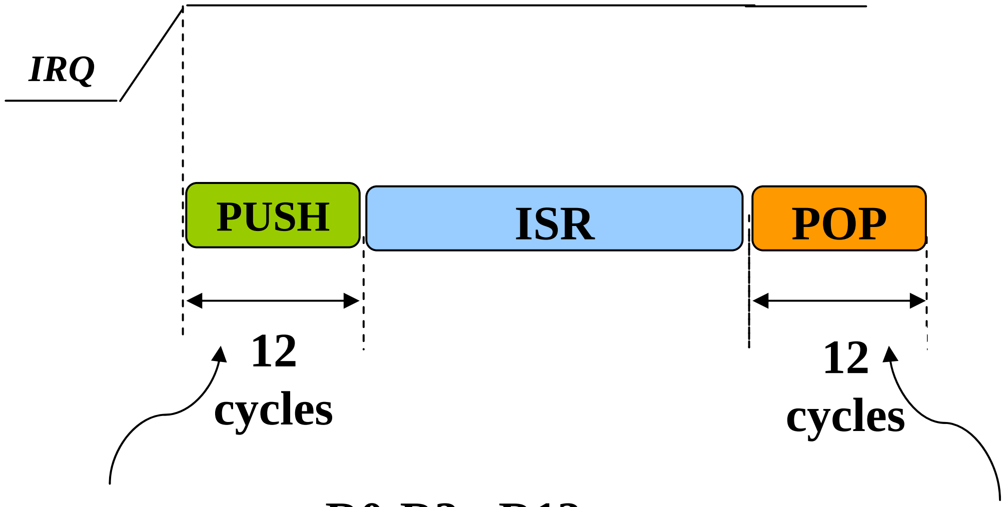
✓ The Vectot table is relocatable.

✓ After reset, the vector table is located at address 0x0.

Exception Type	Address	Offset Exception Vector
18–255	0x48–0x3FF	IRQ #2–239
17	0x44	IRQ #1
16	0x40	IRQ #0
15	0x3C	SYSTICK
14	0x38	PendSV
13	0x34	Reserved
12	0x30	Debug Monitor
11	0x2C	SVC
7–10	0x1C–0x28	Reserved
6	0x18	Usage fault
5	0x14	Bus fault
4	0x10	MemManage fault
3	0x0C	Hard fault
2	0x08	NMI
1	0x04	Reset
0	0x00	Starting value of the MSP

Cortex-M3: Interrupt Handling (5/9)

Interrupt Latency



Save Registers (Hardware)

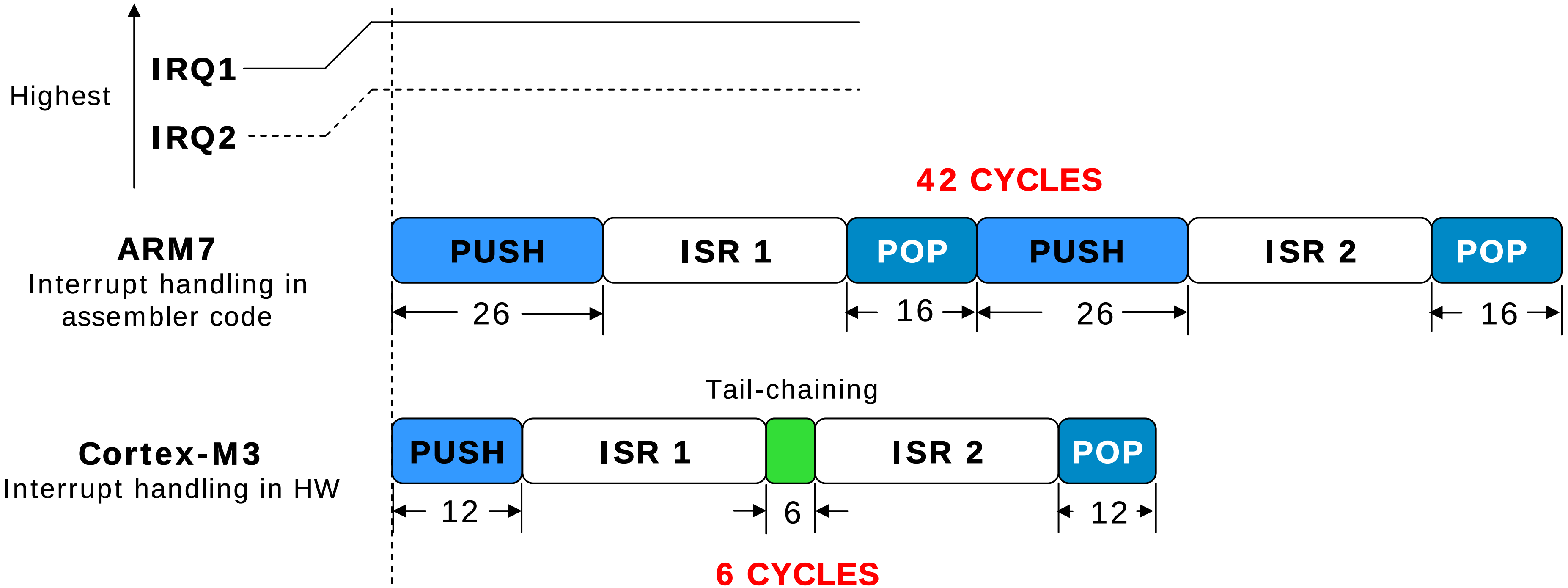
R0-R3 ; R12
R15 (Program Counter)
R14 (Link register)
PSR (Registre d’état)

Restore Registers (Hardware)

For the **ARM7** architecture (Cortex-M predecessor), The PUSH and POP operations were coded in assembler and they last 26 cycles

Cortex-M3: Interrupt Handling (6/9)

Tail Chaining



ARM7

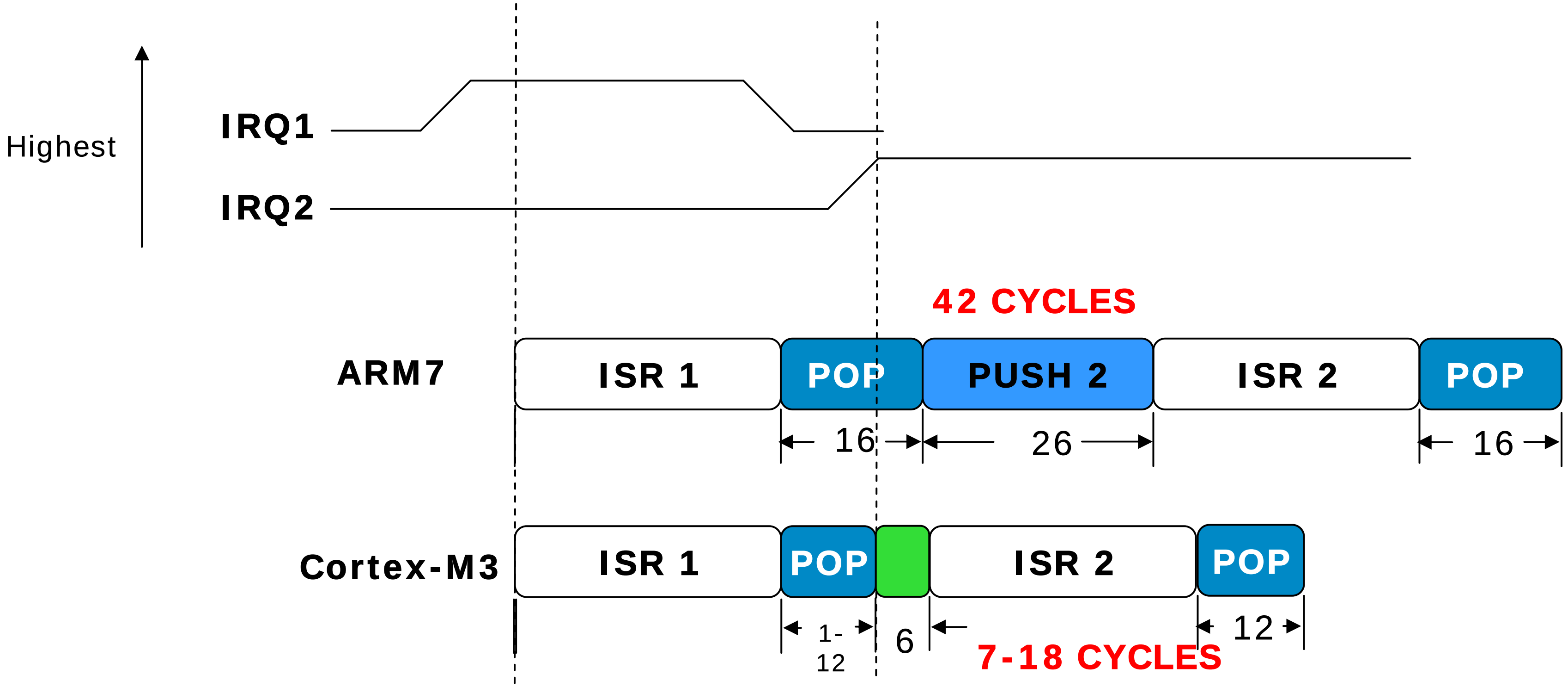
- 26 cycles from IRQ1 to ISR1 entered
  - Up to 42 cycles if LSM
- 42 cycles from ISR1 exit to ISR2 entry
- 16 cycles to return from ISR2

Cortex-M3

- 12 cycles from IRQ1 to ISR1 entered
  - 12 cycles if LSM
- 6 cycles from ISR1 exit to ISR2 entry
- 12 cycles to return from ISR2

Cortex-M3: Interrupt Handling (7/9)

Preemption



ARM7

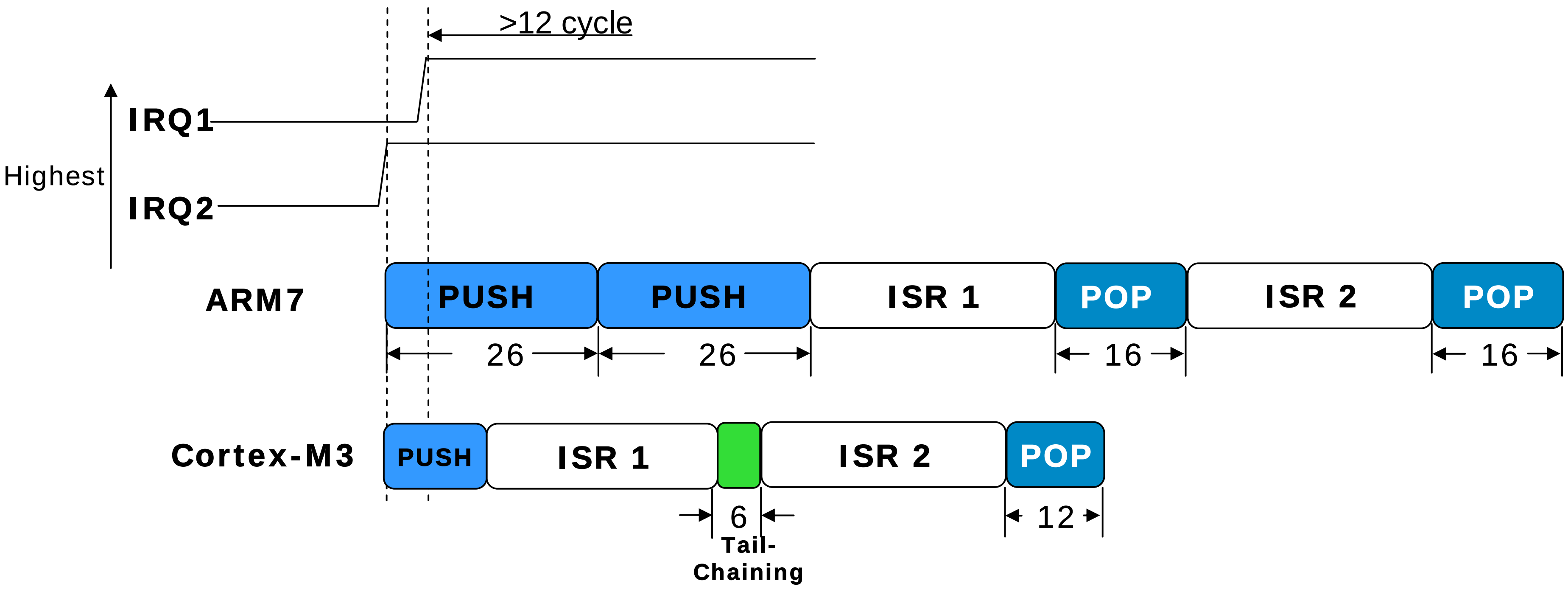
- Load Multiple uninterruptible, and hence the core must complete the POP and the full stack PUSH

Cortex-M3

- POP may be abandoned early if another interrupt arrives
- If POP is interrupted it only takes 6 cycles to enter ISR2 (Equivalent to Tail -chaining)

Cortex-M3: Interrupt Handling (8/9)

Late Arrival



ARM7

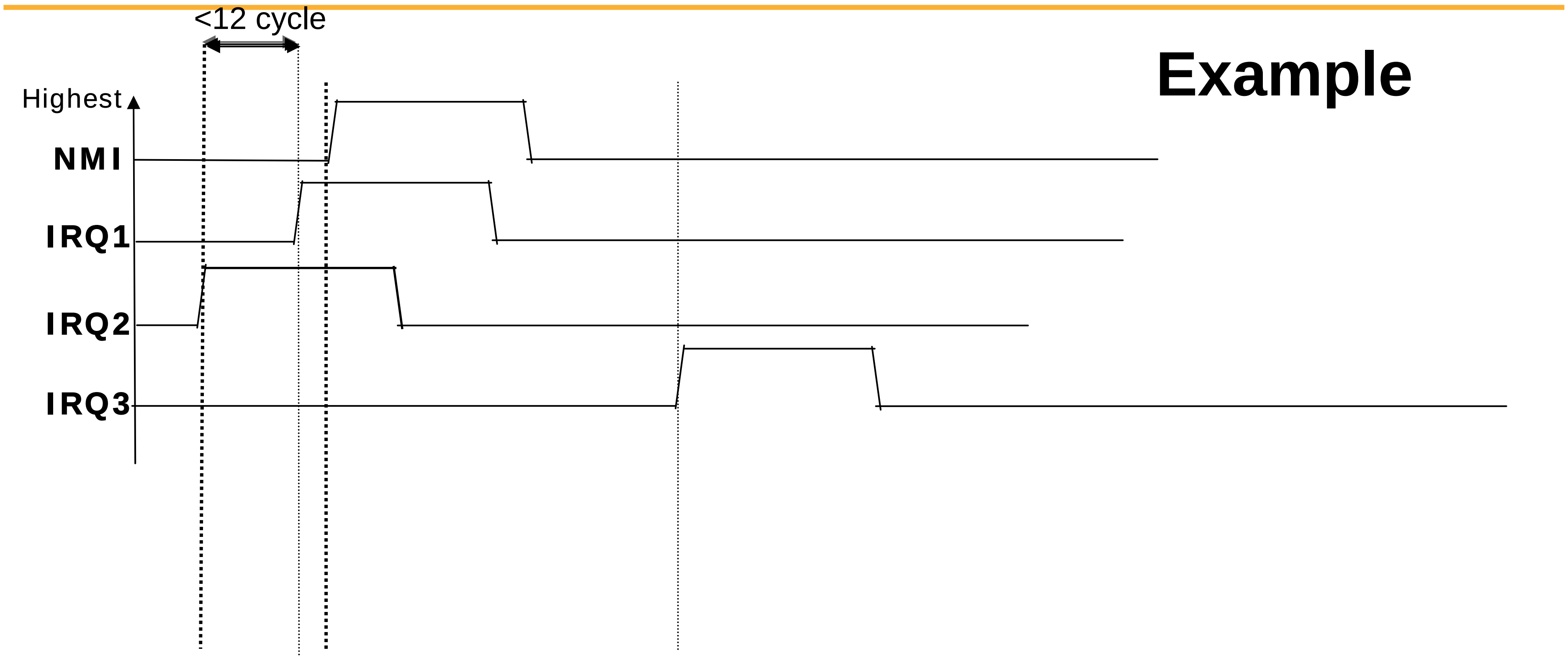
- 26 cycles to ISR2 entered
- Immediately pre-empted by IRQ1 and takes a further 26 cycles to enter ISR 1.
- ISR 1 completes and then takes 16 cycles to return to ISR 2.

Cortex-M3

- Stack push to ISR 2 is interrupted
- Stacking continues but new vector address is fetched in parallel
- 6 cycles from late-arrival to ISR1 entry.
- Tail-chain into ISR 2

Cortex-M3: Interrupt Handling (9/9)

Example



OUTLINE

- Introduction
- Cortex-M3
- Cortex-M3 Memory
- Cortex-M3 Interrupt Handling
- Cortex-M3 Specificities
  - Power management
  - System Timer
  - Debug Capabilities
- Cortex-M3 based MCUs: The STM32F10x Family
- STM32 Programming

Cortex-M3: Power management (1/1)

8bit Microcontroller like power mode management

- SLEEP NOW
  - ♦ **“Wait for Interrupt”** instructions to enter low power mode
    - No more dedicated control register settings sequence
  - ♦ **“Wait for Event”** instructions to enter low power mode
    - No need of Interrupt to wake-up from sleep
    - Rapid resume from sleep
- SLEEP on EXIT
  - ♦ Sleep request done in interrupt routine
  - ♦ Low power mode entered on interrupt return
    - **Very fast wakeup time**
- DEEP SLEEP
  - ♦ **Long duration sleep**
    - From product side: PLL can be stopped or shuts down the power to digital parts of the system
    - Enables low power consumption

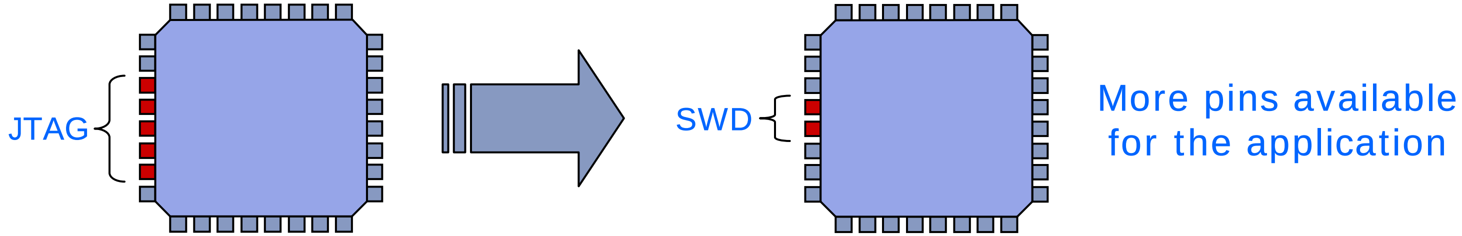
- **Optimized RUN mode CORE power consumption**

Cortex-M3: SysTick ‘System Timer’ (1/1)

- Flexible system timer
- 24-bit self-reloading down counter with end of count interrupt generation
- 2 configurable Clock sources
- Suitable for Real Time OS or other scheduled tasks

Cortex-M3: Debug Capabilities (1/1)

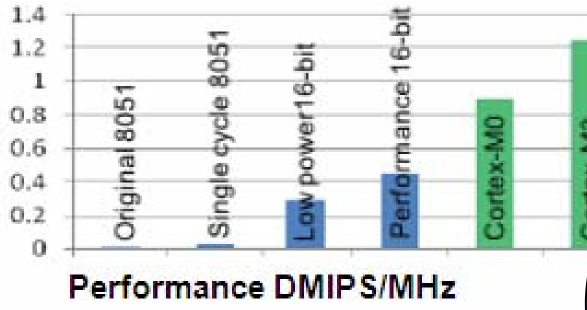
- **Serial Wire Debugging** for optimized device pin-out

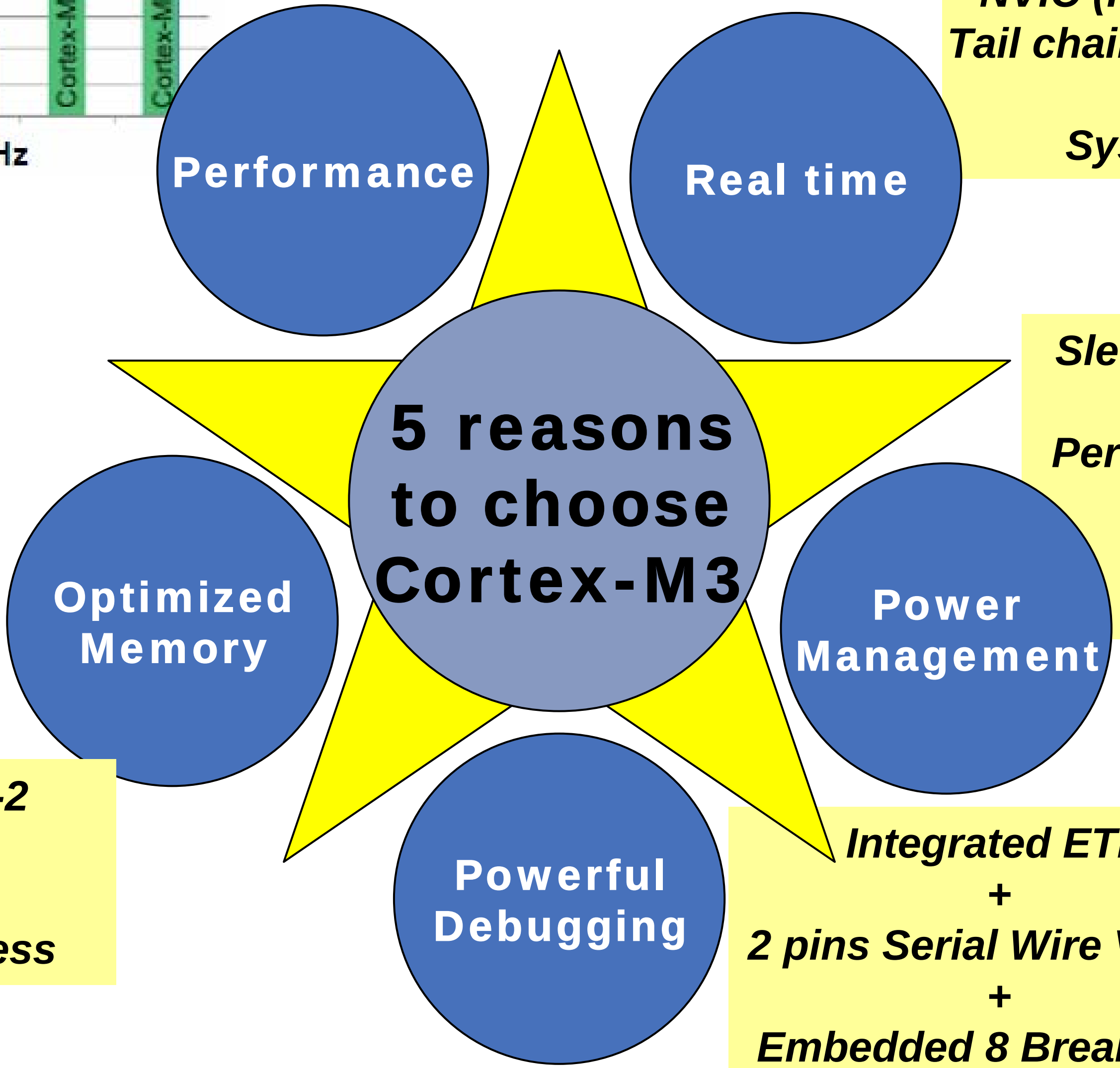


More pins available for the application

- **Embedded break/watch capabilities** for **easy flashed application debugging**
  - ♦ ~~2 hardware breakpoints~~ → 8 hardware breakpoints
  - ♦ 2 hardware watchpoints
- **Serial Wire Viewer** for targeted low bandwidth **data trace**
  - ♦ Using serial wire interface or dedicated bus CKout+D[3..0] for better bandwidth
  - ♦ Triggered by embedded break and watch points
- **ETM capability for better real time debugging**
  - ♦ Instruction trace only
  - ♦ External signal triggering capability
  - ♦ Can be used in parallel with data watchpoint
- **Debugging features still kept whilst the core entered low power mode**

Why Cortex-M3 ?





NVIC (HW Push & Pop, Tail chaining, Late arrival)
+
SysTick Timer

Sleep & Stop modes
+
Peripherals powered Separately
+
12 MIPS/ mW

Integrated ETM
+
2 pins Serial Wire Viewer
+
Embedded 8 Breakpoint & 2 Watchpoints

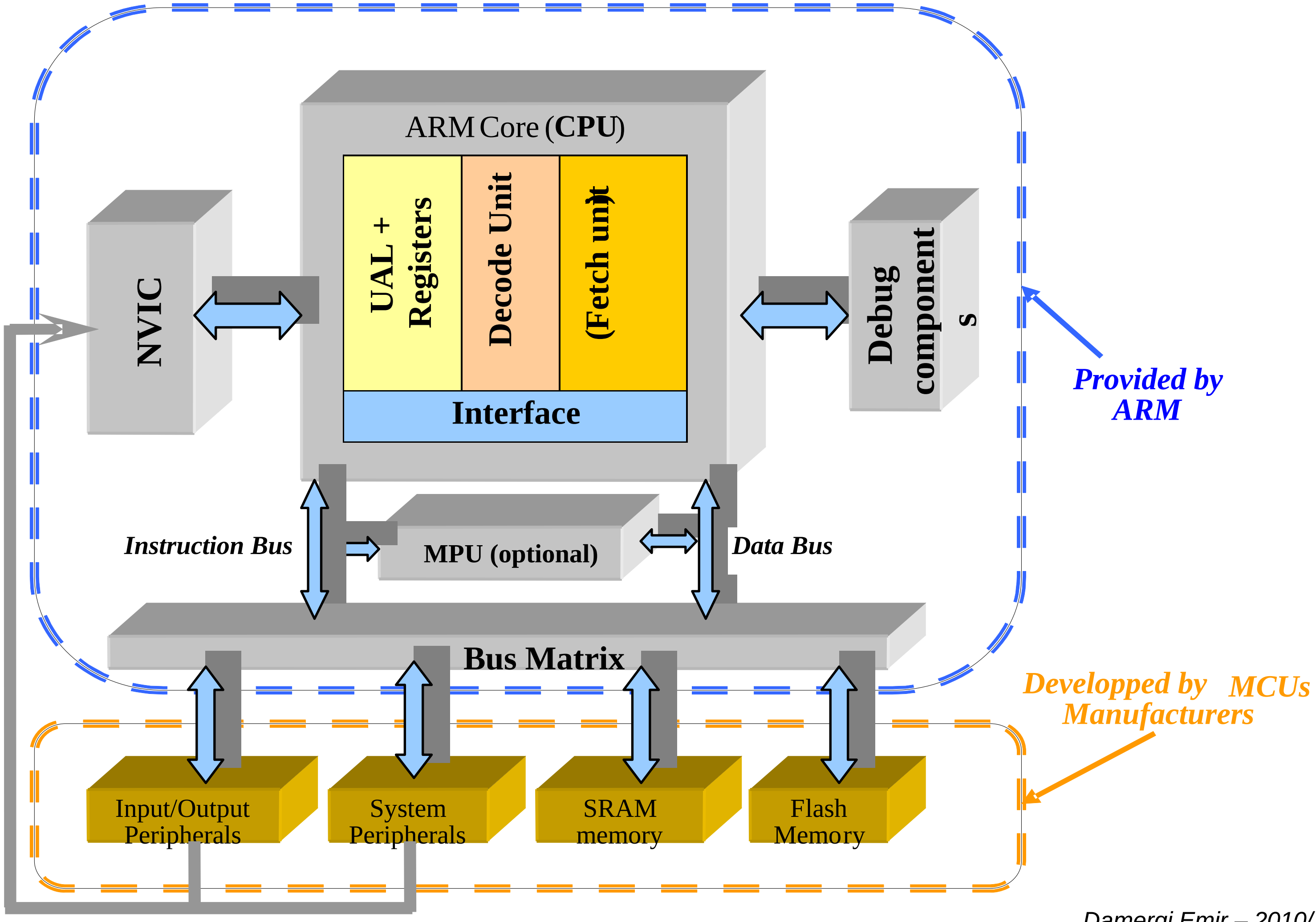
16 bit Thumb-2 instructions
+
Unaligned access

OUTLINE

- Introduction
- Cortex-M3
- Cortex-M3 Memory
- Cortex-M3 Interrupt Handling
- Cortex-M3 Specificities
- Cortex-M3 based MCUs: The STM32F10x Family
  - Architecture
  - Boot Modes
  - Flash features
  - Power & Reset
  - On chip Oscllators & Clock Scheme
- STM32 Programming

Cortex based MCUs

The MCUs Manufacturers integrate the Cortex Processor and add the I/O & System peripherals, SRAM Memory, Flash, etc…



Cortex-M based MCUs Manufacturers

	Cortex-M0	Cortex-M3	Cortex-M4
ST		***STM32***	
Texas Instruments		***Stellaris3x***	
NXP	***LPC11x***	***LPC17x,*** ***LPC3x***	***DSCx***
Atmel		***AT91SAM3x***	
freescale		***LM3S8x***	

STM32: The Cortex-M3 MCU Family

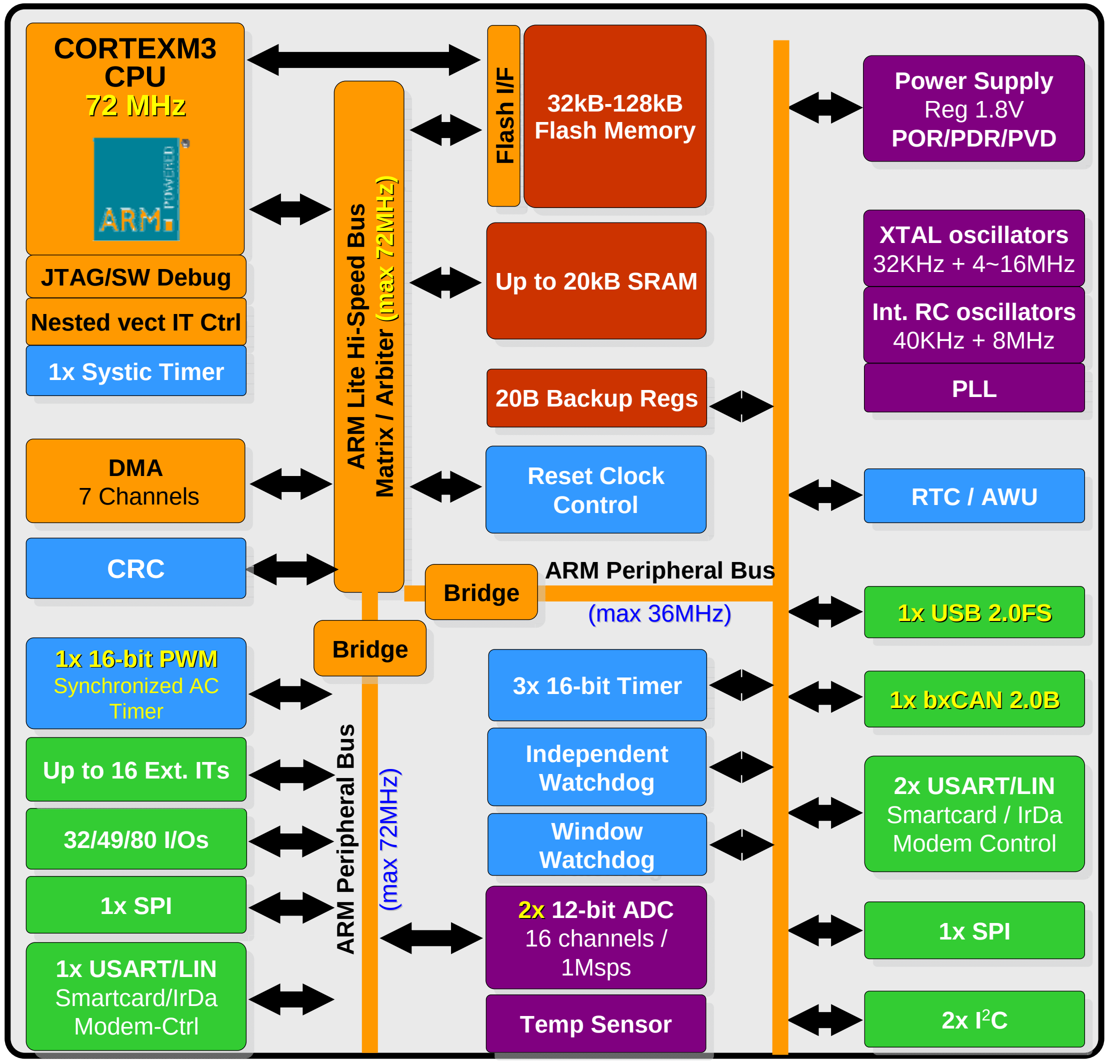
Designation: STM32F***10x y z:***

- y: Number of pins (**T: 36 pins**, C: 48 pins, **R : 64 pins**, V : 100 pins et **Z : 144 pins**)
- z: Flash Memory Size (**4 :16 Kbytes**, *6: 32 Kbytes*, ***8: 64 Kbytes***, *B: 128 Kbytes*, **C: 256 Kbytes**, *D: 384 Kbytes and* **E: 512 Kbytes**) .
- 10x : Product line (see table below)

Product Line	*Reference*	**Common components**	*Freq (MHz)*	*SRAM Size (Kbyte)*	*Specific Peripherals*
Connectivity	105 & 107	• USART (up to 5) • DAC 12 bits (up to 2) • Timers 16 bts (up to 6) • DMA channels (up to 12) • Internal RC oscillators (40 KHz et 8 MHz) • Real Time Clock.	72	Up to 64	USB, CAN, I^{2}S (classe audio) et Ethernet
Performance	103		72	64	USB, SDIO, CAN, I^{2}S et Timer PWM.
USB Access	102		48	16	USB
Access	101		36	48	

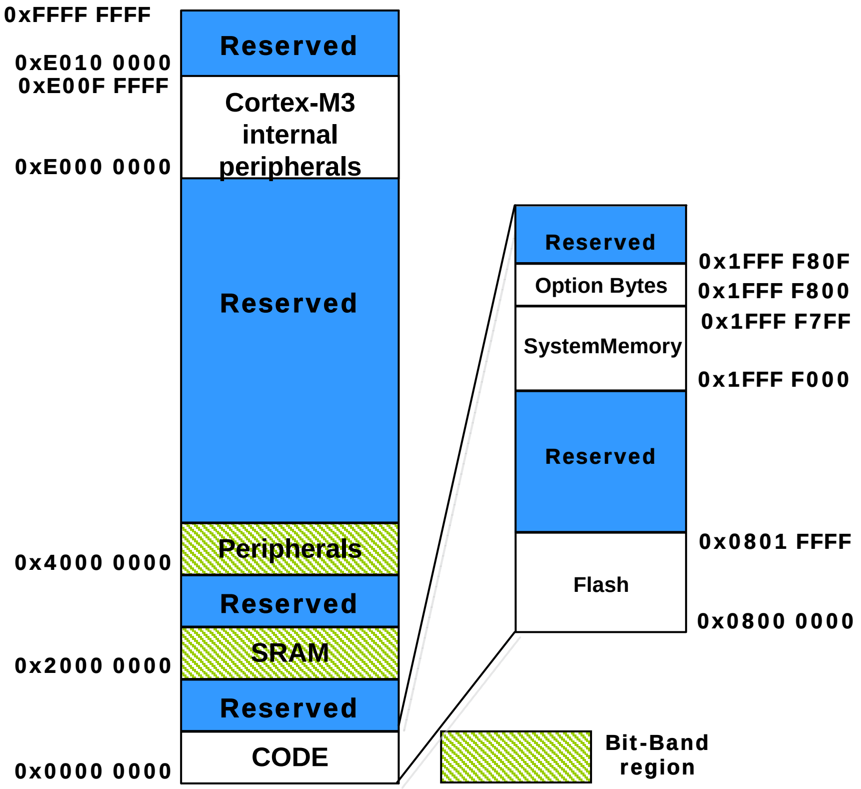
STM32F10x Series Block Diagram

- **ARM 32-bit Cortex-M3 CPU**
- **Nested Vectored Interrupt Controller (NVIC) w/ 43 maskable IT + 16 prog. priority levels**
- **Embedded Memories :**
  - FLASH: up 128 Kbytes
  - SRAM: up 20 Kbytes
- **CRC calculation unit**
- **7 Channels DMA**
- **Power Supply with internal regulator and low power modes :**
  - 2V to 3V6 supply
  - 4 Low Power Modes with Auto Wake-up
- **Integrated Power On Reset (POR)/Power Down Reset (PDR) + Programmable voltage detector (PVD)**
- **Backup domain w/ 20B reg**
- **Up to 72 MHz frequency managed & monitored by the Clock Control**
- **Rich set of peripherals & IOs**
  - Embedded low power RTC with V_{BAT} capability
  - Dual Watchdog Architecture
  - 5 Timers w/ advanced control features (including Cortex SysTick)
  - 9 communications Interfaces
  - Up to 80 I/Os (100 pin package) w/ 16 external interrupts/event
  - Up to 2x12-bits 1Msps ADC w/ up to 16 channels and Embedded temperature sensor w/ +/-1.5° linearity with T°



STM32F10x :Memory Mapping and Boot Modes

- Addressable memory space of 4 GBytes
- RAM : up to 20 kBytes
- FLASH : up to 128 kBytes



0xFFFF FFFF
Reserved
0xE010 0000
0xE00F FFFF
Cortex-M3 internal peripherals
0xE000 0000
Reserved
Peripherals
0x4000 0000
Reserved
SRAM
0x2000 0000
Reserved
CODE
0x0000 0000

Reserved
0x1FFF F80F
Option Bytes
0x1FFF F800
0x1FFF F7FF
SystemMemory
0x1FFF F000
Reserved
0x0801 FFFF
Flash
0x0800 0000

Bit-Band region

- **Boot modes:**

  Depending on the Boot configuration
  - Embedded Flash Memory
  - System Memory
  - Embedded SRAM Memory
  is aliased at @0x00

BOOT Mode Selection Pins		Boot Mode	Aliasing
BOOT1	BOOT0		
x	0	User Flash	User Flash is selected as boot space
0	1	SystemMemory	SystemMemory is selected as boot space
1	1	Embedded SRAM	Embedded SRAM is selected as boot space

SystemMemory: The internal boot **ROM memory** contains the Bootloader (programmed by ST in production) used to re-program the FLASH

- through USART (STM32F101xx, 102xx & 103xx)
- through USART, CAN, USB, etc. (STM32F105xx & 107xx)

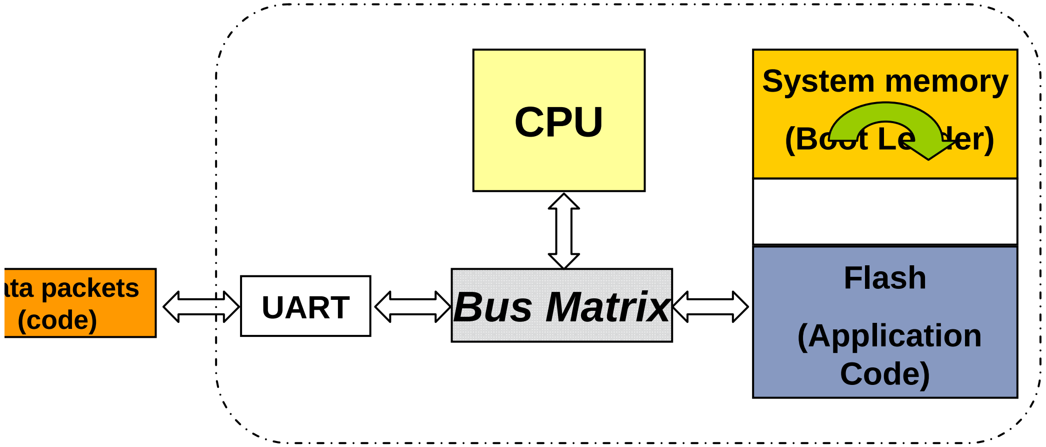
STM32F10x :Boot Modes

- **Boot mode = user Flash**

For finalized debuged and tested applications where update is not necessary (example: Washing machine).

- **Boot mode = System Memory**

For applications where firmware update is performed frequently (example: Satellite receiver, Playsattion, etc…).



The CPU executes the boot loader:

1) Data packets containing code are Received through through serial peripheral (UART).

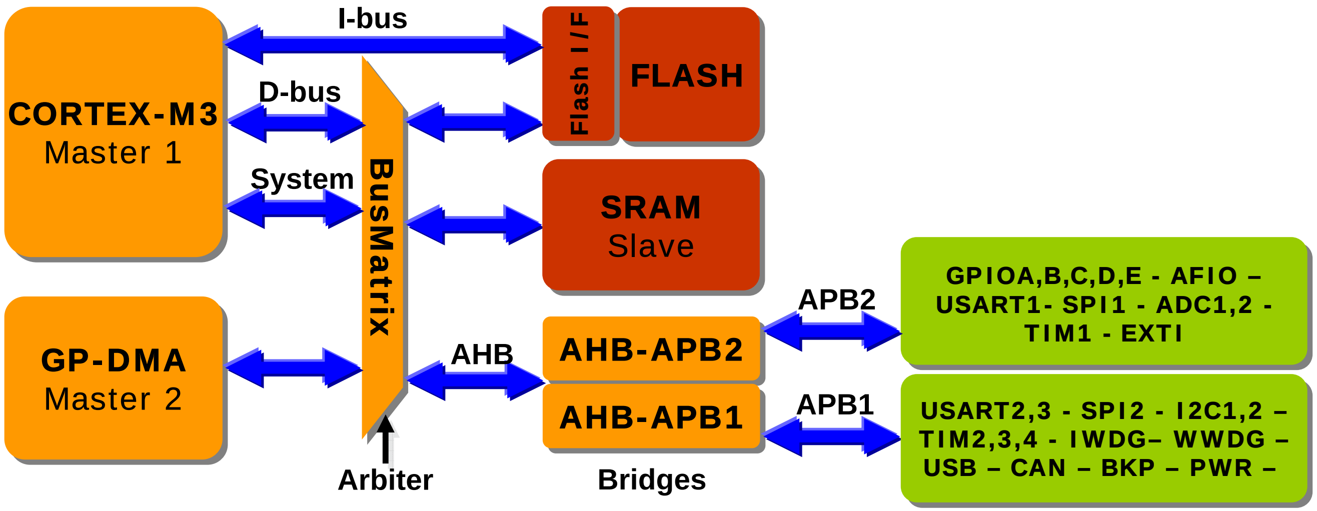
2) The Flash is reprogrammed with the new received code

- **Boot mode = SRAM**

STM32F10x : System Architecture

- **Multiply possibilities of bus accesses to SRAM, Flash, Peripherals, DMA**
  - BusMatrix added to Harvard architecture allows parallel access
- **Efficient DMA and Rapid data flow**
  - Direct path to SRAM through arbiter, guarantees alternating access
  - Harvard architecture + BusMatrix allows Flash execution in parallel with DMA transfer
- **Increase Peripherals Speed for better performance**
  - Dual Advanced Peripheral buses (APB) architecture w/ High Speed APB (APB2) up to 72MHz and Low Speed APB (APB1) up to 36MHz

➔ Allows to optimize use of peripherals (18MHz SPI, 4.5Mbps USART, 72MHz PWM Timer, 18MHz toggling I/Os)


I-bus
CORTEX-M3
Master 1
D-bus
System
BusMatrix
Flash I/F
FLASH
SRAM
Slave
GP-DMA
Master 2
AHB
AHB-APB2
AHB-APB1
APB2
APB1
GPIOA,B,C,D,E - AFIO – USART1- SPI1 - ADC1,2 - TIM1 - EXTI
USART2,3 - SPI2 - I2C1,2 – TIM2,3,4 - IWDG– WWDG – USB – CAN – BKP – PWR –
Arbiter
Bridges

Buses are not overloaded with data movement tasks

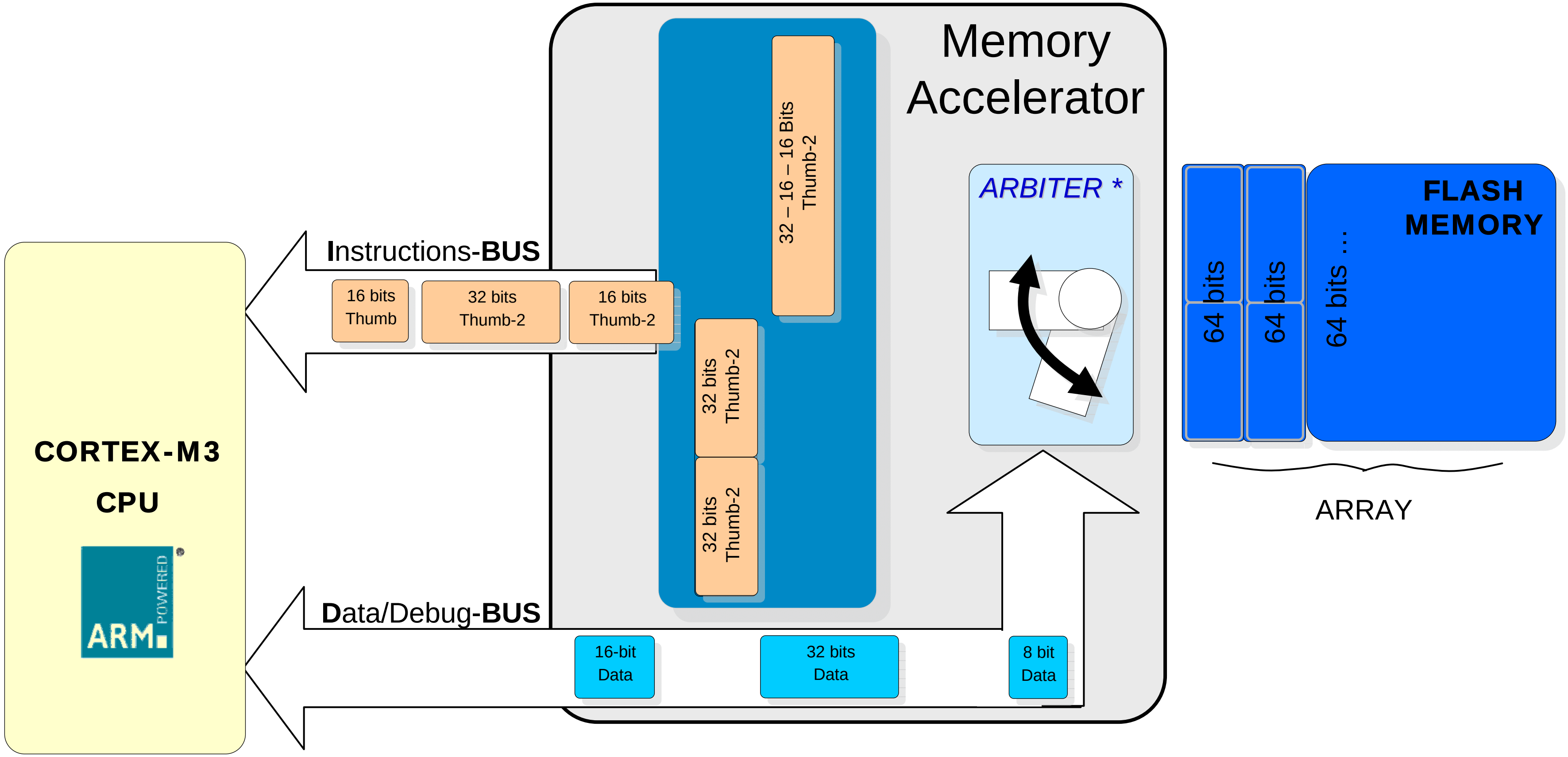
STM32F10x :Flash Features Overview (1/2)

- Flash Features:
  - Up to 128KBytes
  - 1 KByte Page size
  - Endurance: 10k cycles
  - Memory organization:
    - Main memory block
    - Information block
  - Access time: 35ns
  - Halfword (16-bit) program time: 52.5 µs (Typ)
  - Page / Mass Erase Time: 20ms

- Flash interface (FLITF) Features:
  - Read Interface with pre-fetch buffer
  - Option Bytes loader
  - Flash program/Erase operations
  - Types of Protection: Readout Protection/Write Protection

STM32F10x: Flash Memory Accelerator (2/2)

- Mission: Support 72 MHz operation directly from Flash memory
- 64-bits wide Flash with Prefetch (2 × 64bits buffers)



* The data (constant or literals) are provided with the highest priority using the D-Bus.

STM32F10x: Power Supply

- Power Supply Schemes
  - **V_{DD}** = 2.0 to 3.6 V: External Power Supply for I/Os and the internal regulator.
  - **V_{DDA}** = 2.0 to 3.6 V: External Analog Power supplies for ADC, Reset blocks, RCs and PLL.

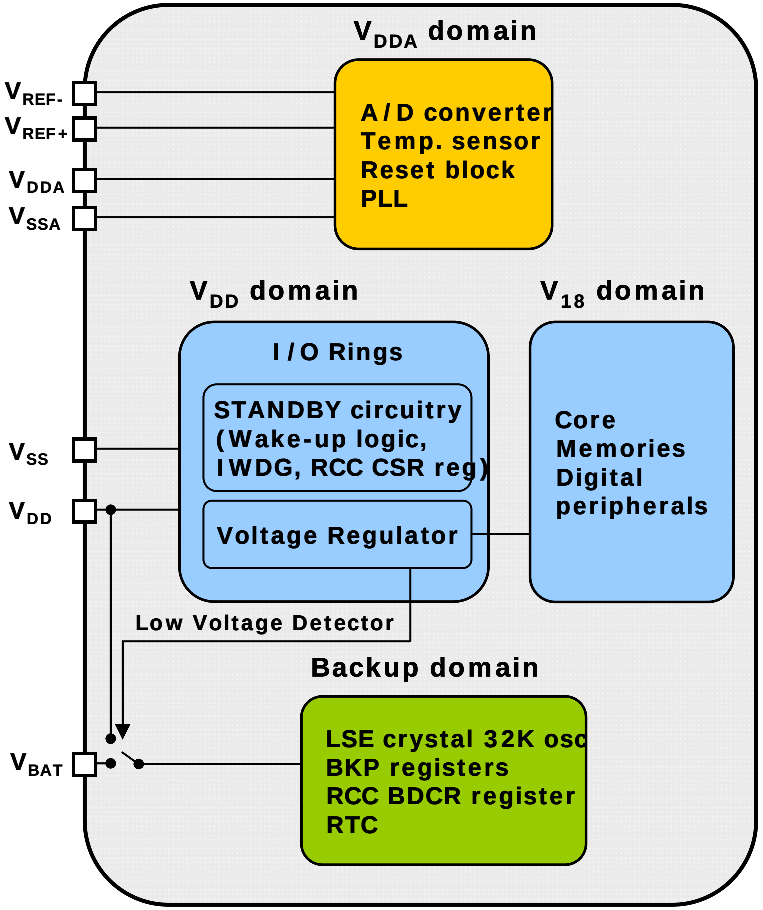
    ➔ ADC working only if $V_{DDA} \geq 2.4$ V
  - **V_{BAT}** = 1.8 to 3.6 V: For Backup domain when **V_{DD}** is not present.
  - Power pins connection:

    - V_{DD} and V_{DDA} must be connected to the same power source

    - V_{SS}, V_{SSA} and V_{REF-} must be tight to ground

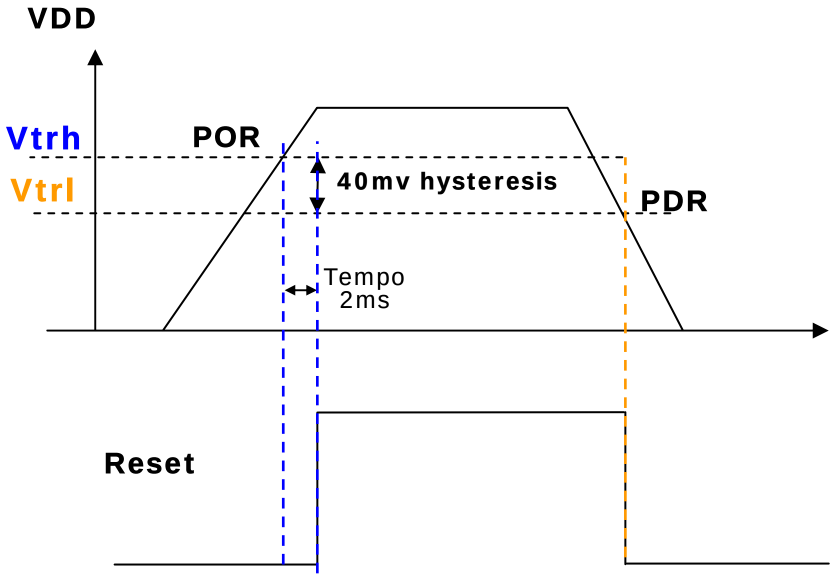
    - 2.4V ≤ V_{REF+} ≤ V_{DDA}

    - V_{REF+} and V_{REF-} available only on 100-pin (144-pin) packages, in other packages they are internally connected respectively to V_{DDA} and V_{SSA}



STM32F10x: Power On Reset (POR)/Power Down Reset (PDR)

- Integrated POR / PDR circuitry guarantees proper product reset when voltage is not in the product guaranteed voltage range (2V to 3.6V)
  - **No need for external reset circuit**
- POR and PDR have a typical hysteresis of 40mV



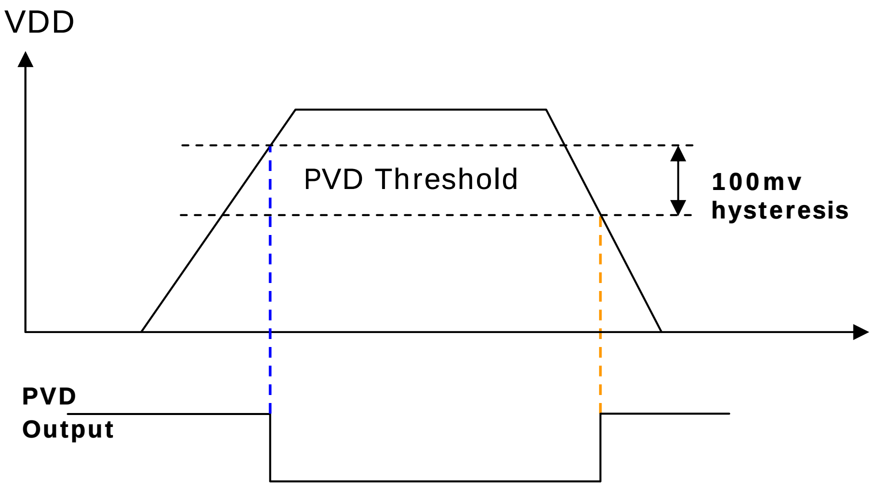
Vtrl min 1.8V / Vtrh max 2V

STM32F10x: Programmable Voltage Detector (PVD)

- Programmable Voltage Detector
  - Enabled by software
  - Monitor the V_{DD} power supply by comparing it to a threshold
  - Threshold configurable from 2.2V to 2.9V by step of 100mV
  - Generate interrupt through EXTI Line16 (if enabled) when

VDD < Threshold and/or VDD > Threshold

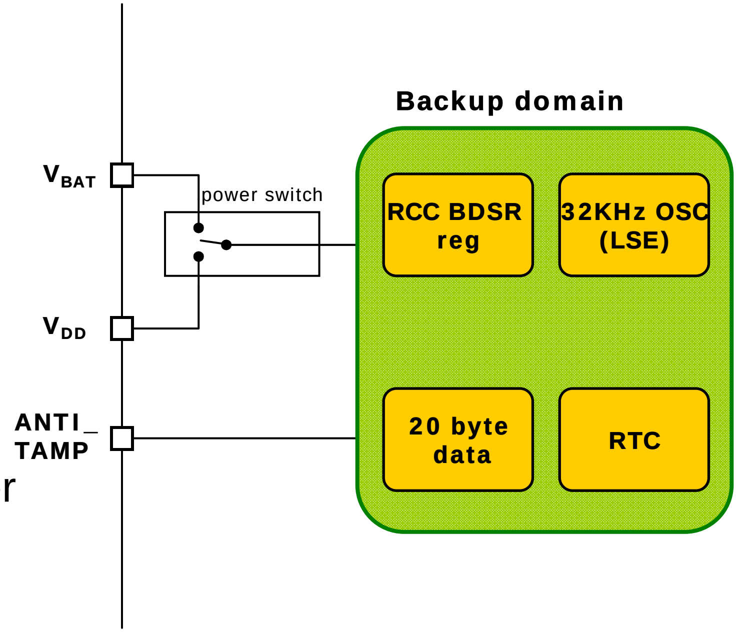
➔ Can be used to generate a warning message and/or put the MCU into a safe state



STM32F10x: Backup Domain

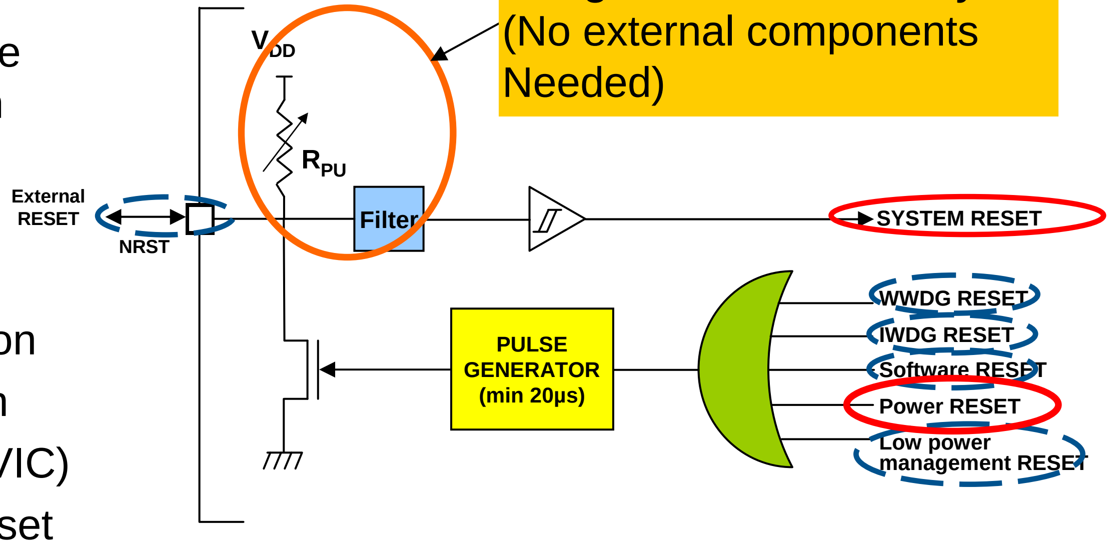
- Backup Domain contains
  - RTC (Counter, Prescaler and Alarm mechanism)
  - Separate 32KHz Osc (LSE) for RTC
  - 20-byte user backup data
  - RCC BDSR register: RTC source clock selection and enable + LSE config

  ➔ Reset only by Backup domain RESET
- V_{BAT} independent voltage supply
  - Automatic switch-over to V_{BAT} when V_{DD} goes lower than PDR level
  - No current sunk on V_{BAT} when V_{DD} present
- Tamper detection: resets all user backup registers
  - Configurable level: low/high
  - Configurable interrupt generation



STM32F10x: RESET Sources

- System RESET
  - Resets all registers except some RCC registers and BKP domain
  - Sources
    - ✓ - Low level on the NRST pin (External Reset)
    - ✓ - WWDG end of count condition
    - ✓ - IWDG end of count condition
    - ✓ - A software reset (through NVIC)
    - ✓ - Low power management Reset



Integrated RC circuitry
(No external components
Needed)

- Power RESET
  - Resets all registers except BKP domain
  - Sources
    - - Power On/Power down Reset (POR/PDR)
    - - When exiting STANDBY mode

- Backup domain RESET
  - Resets all BKP domain
  - Sources
    - - Setting BDRST bit in RCC BDCR register
    - - VDD or VBAT power on, if both supplies have previously been powered off.

STM32F10x: On Chip Oscillators(1/2)

- Multiple clock sources for full flexibility in RUN/Low Power modes
  - **HSE** (High Speed External oscillator): 4MHz to 25MHz main osc which can be multiplied by the PLL to provide a wide range of frequencies

    ➔ Can be bypassed with external clock
  - **HSI** (High Speed **Internal RC**): factory trimmed internal RC oscillator 8MHz **+/- 1%** over 0-70°C temp range
    - Feeds System clock after reset or exit from STOP mode for fast startup (startup time : 2us max)
    - Backup clock in case HSE osc is failing

    Note: When the HSI is used as a PLL clock input, the maximum system clock frequency that can be achieved is 64 MHz.
  - **LSI** (Low Speed **Internal RC**): 40KHz internal RC for IWDG and optionally for the RTC used for Auto Wake-Up (AWU) from STOP/STANDBY mode

STM32F10x: On Chip Oscillators(2/2)

- **LSE** (Low Speed External oscillator): 32.768kHz osc provides a precise time base with very low power consumption (max 1µA). Optionally drives the RTC for Auto Wake-Up (AWU) from STOP/STANDBY mode.

  ➔ Can be bypassed with external clock

STM32F10x: Clock Scheme

- **System Clock (SYSCLK) sources**
  - ✓ HSI
  - ✓ HSE
  - ✓ PLL

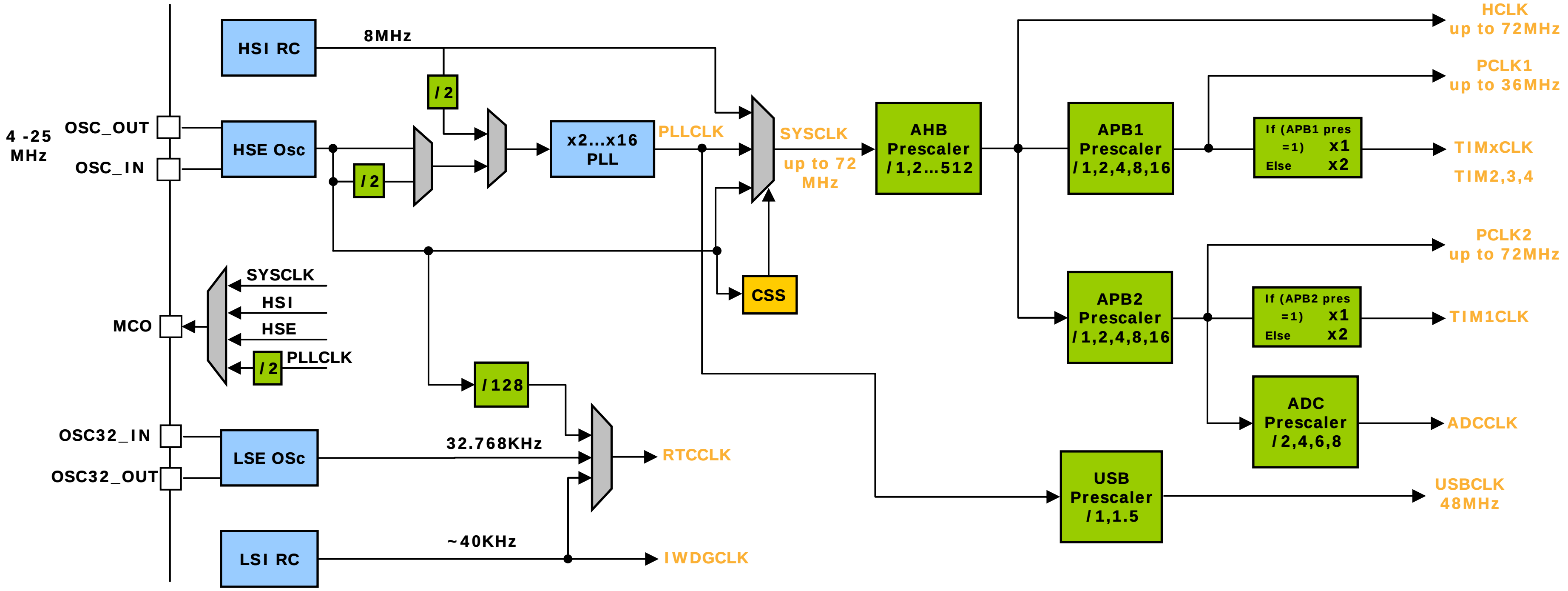
- **RTC Clock (RTCCLK) sources**
  - ✓ LSE
  - ✓ LSI
  - ✓ HSE clock divided by 128

- **USB Clock (USBCLK) provided from the internal PLL**

- **Clock-out capability on the MCO pin (PA.08) / max 50MHz**

- **Configurable dividers provides AHB, APB1/2, ADC and TIM clocks**

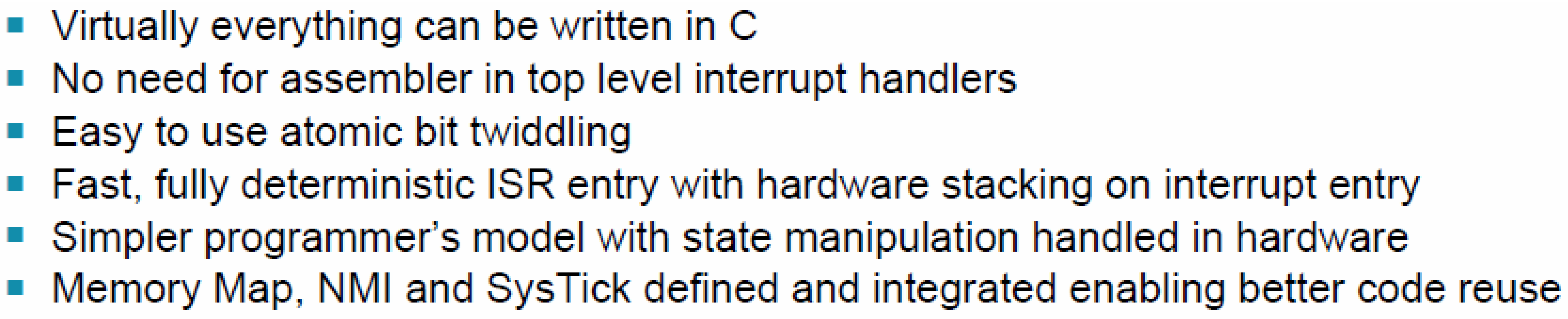
- **Clock Security System (CSS) to backup clock in case of HSE clock failure (HSI feeds the system clock)**
  - Enabled by SW w/ interrupt capability linked to Cortex NMI



OUTLINE

- Introduction
- Cortex-M3
- Cortex-M3 Memory
- Cortex-M3 Interrupt Handling
- Cortex-M3 Specificities
- Cortex-M3 based MCUs: The STM32F10x Family
- STM32 Programming
  - The CMSIS Standard
  - Files deployment & configuration
  - Programming steps (using the ST Library)
  - Interrupt programming
  - Software Tools

STM32 Programming: the software factor

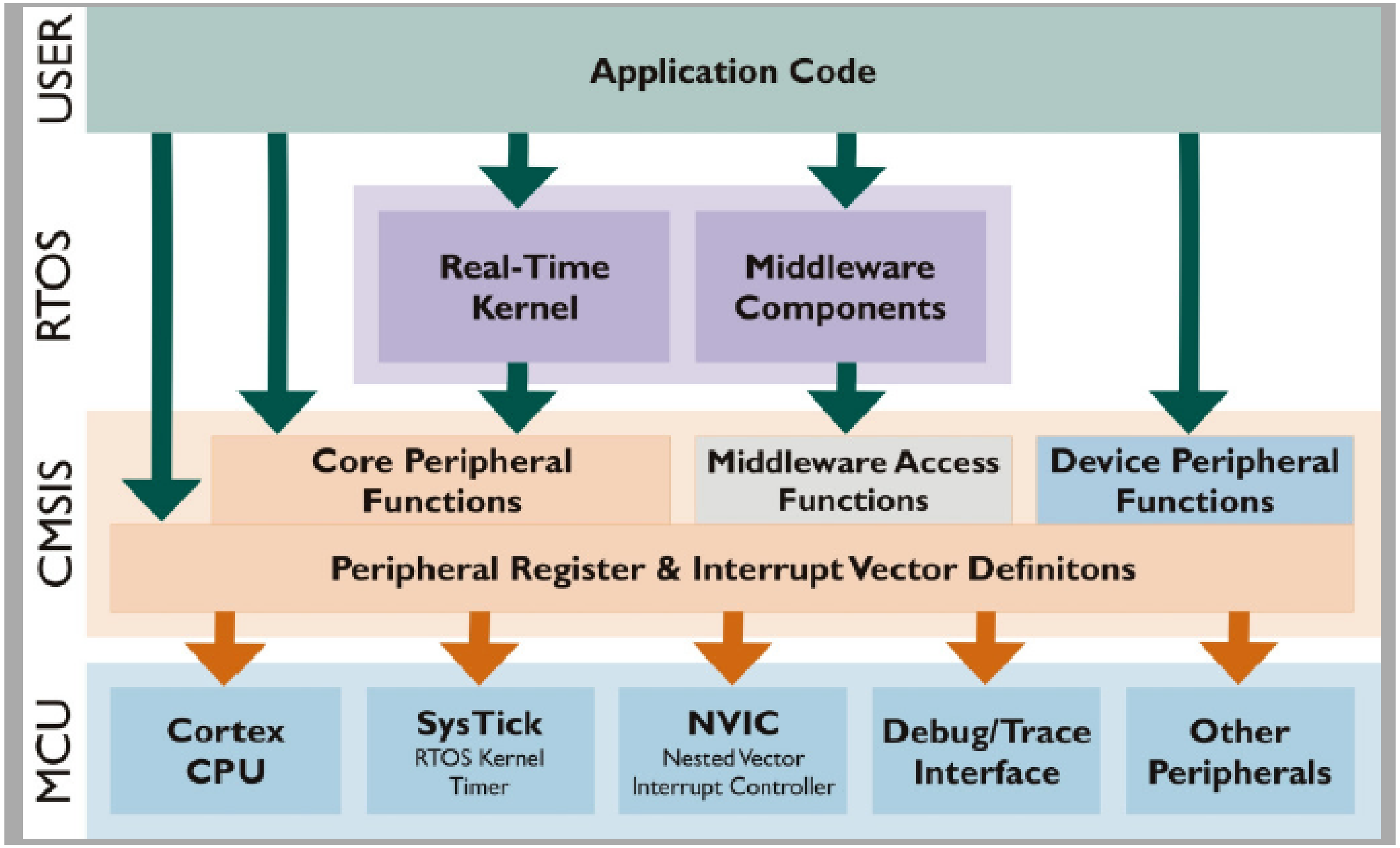


- Virtually everything can be written in C
- No need for assembler in top level interrupt handlers
- Easy to use atomic bit twiddling
- Fast, fully deterministic ISR entry with hardware stacking on interrupt entry
- Simpler programmer's model with state manipulation handled in hardware
- Memory Map, NMI and SysTick defined and integrated enabling better code reuse

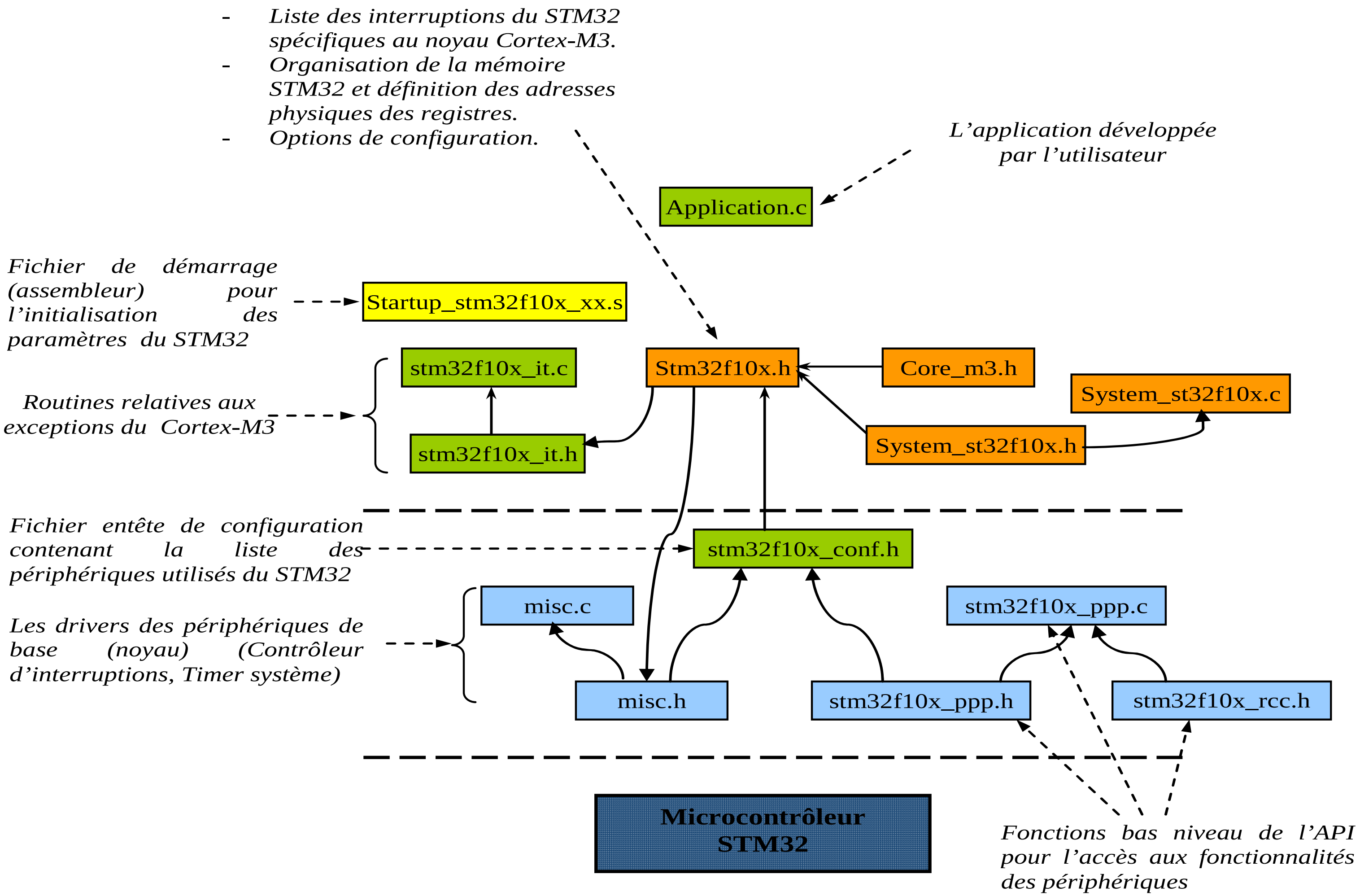
+ CMSIS Standard (Cortex Microcontroller Software Interface Standars

⇨ Eases developpement and porting of applications

The CMSIS programming standard


USER
Application Code
RTOS
Real-Time Kernel
Middleware Components
CMSIS
Core Peripheral Functions
Middleware Access Functions
Device Peripheral Functions
Peripheral Register & Interrupt Vector Definitons
MCU
Cortex CPU
SysTick
RTOS Kernel Timer
NVIC
Nested Vector Interrupt Controller
Debug/Trace Interface
Other Peripherals

The CMSIS programming standard



- *Liste des interruptions du STM32 spécifiques au noyau Cortex-M3.*
- *Organisation de la mémoire STM32 et définition des adresses physiques des registres.*
- *Options de configuration.*

L'application développée par l'utilisateur

Fichier de démarrage (assembleur) pour l'initialisation des paramètres du STM32

Routines relatives aux exceptions du Cortex-M3

Fichier entête de configuration contenant la liste des périphériques utilisés du STM32

Les drivers des périphériques de base (noyau) (Contrôleur d'interruptions, Timer système)

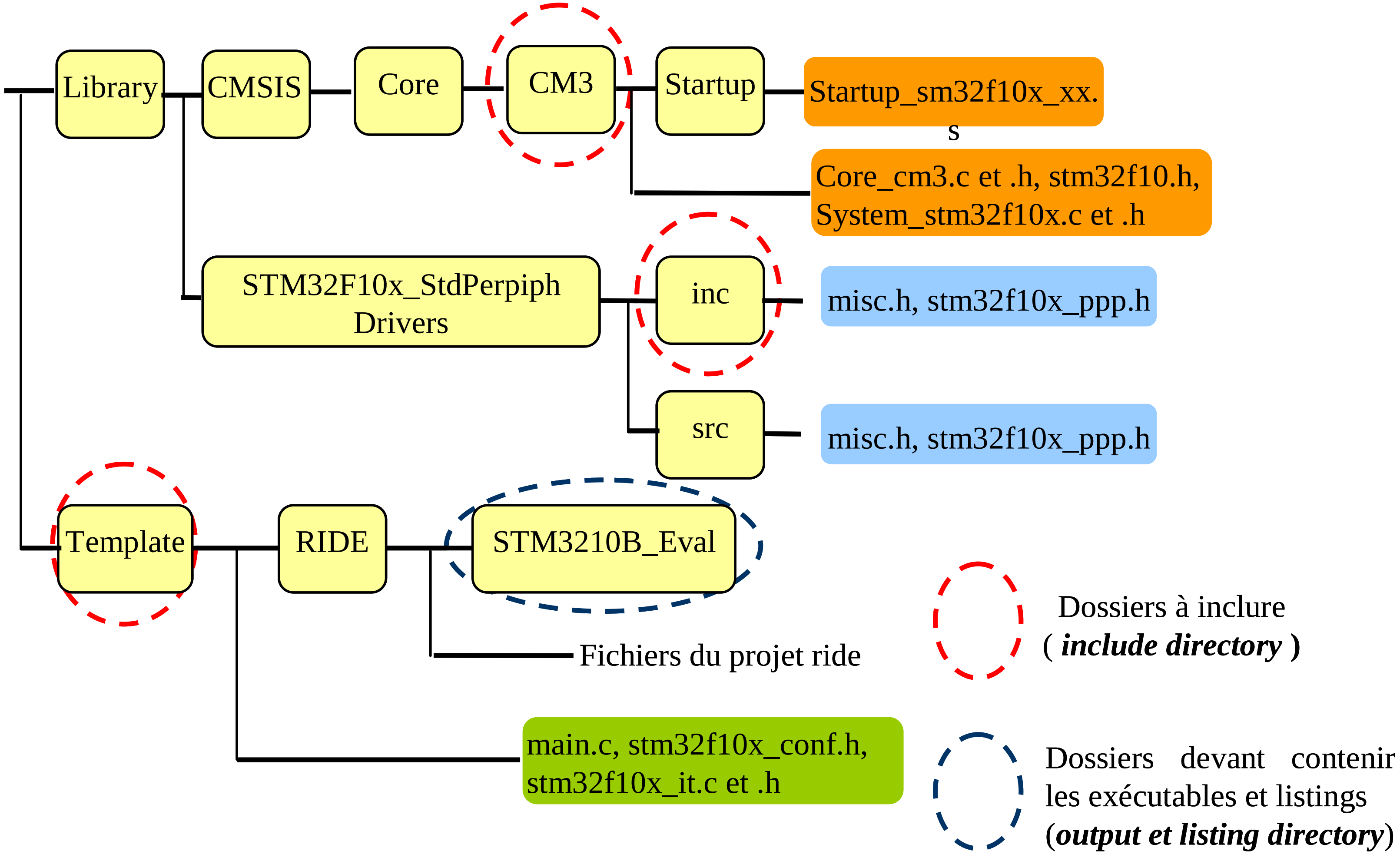
Fonctions bas niveau de l'API pour l'accès aux fonctionnalités des périphériques

🟧 ***Fichiers CMSIS fournis par la société ARM***

🟦 ***Fichiers d'accès aux périphériques fournis par ST-Microelectronics***

🟩 ***Fichiers « template » configurés par l'utilisateur ou code développé entièrement par l'utilisateur.***

Files deployment



Library

CMSIS

Core

CM3

Startup

Startup_sm32f10x_xx.s

Core_cm3.c et .h, stm32f10.h, System_stm32f10x.c et .h

STM32F10x_StdPerpiph Drivers

inc

misc.h, stm32f10x_ppp.h

src

misc.h, stm32f10x_ppp.h

Template

RIDE

STM3210B_Eval

Fichiers du projet ride

main.c, stm32f10x_conf.h, stm32f10x_it.c et .h

Dossiers à inclure (***include directory***)

Dossiers devant contenir les exécutables et listings (***output et listing directory***)

Files to be modified by the user

- **system_stm32f10x.c**

```
#include "stm32f10x.h“
/* #define SYSCLK_FREQ_HSE    HSE_Value */
/* #define SYSCLK_FREQ_24MHz  24000000 */
/* #define SYSCLK_FREQ_36MHz  36000000 */
/* #define SYSCLK_FREQ_48MHz  48000000 */
/* #define SYSCLK_FREQ_56MHz  56000000 */
#define SYSCLK_FREQ_72MHz  72000000
…
```

- **stm32f10x_conf.h**

```
/* Includes ------------------------------------------------------------------*/
/* Uncomment the line below to enable peripheral header file inclusion */
/* #include "stm32f10x_adc.h" */
/* #include "stm32f10x_bkp.h" */
/* #include "stm32f10x_can.h" */
…
…
```

- **stm32f10x.h**

```
/* Uncomment the line below according to the target STM32 device used
        in your  application   */
#if !defined (STM32F10X_LD) && !defined (STM32F10X_MD) && !defined
        (STM32F10X_HD)
 /* #define STM32F10X_LD */   /*!< STM32 Low density devices */
 /* #define STM32F10X_MD */   /*!< STM32 Medium density devices */
  #define STM32F10X_HD   /*!< STM32 High density devices */
#endif
…
…
```

- **main.c**

```
#include "stm32f10x.h“
int main(void)
{
  /* Setup STM32 system (clock, PLL and Flash configuration) */
  SystemInit();
/* main program*/
...
GPIO_WriteBit(GPIOD, GPIO_Pin_1, Bit_SET);
…
}
```

- **stm32f10x_It.h**

```
/* Exported functions ------------------------------------------------------- */
void NMI_Handler(void);
void HardFault_Handler(void);
…
```

- **stm32f10x_It.c“**

```
#include "stm32f10x_it.h"
void HardFault_Handler(void)
{
  /* Go to infinite loop when Hard Fault exception occurs */
  while (1)
  {}
}
…
void EXTI1_IRQHandler(void)
{
GPIO_WriteBit(GPIOD, GPIO_Pin_1, Bit_SET);
}
```

Coding conventions

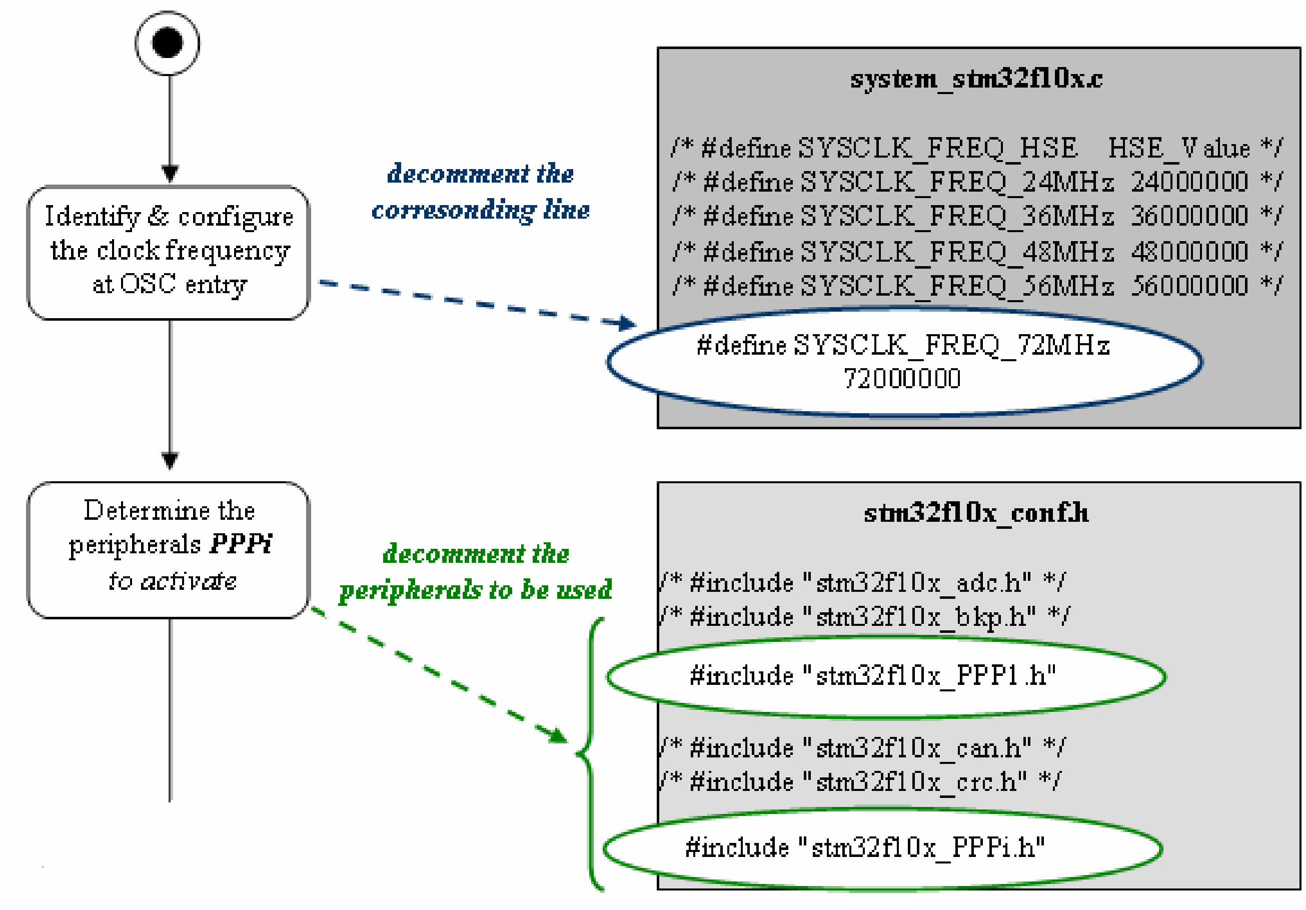
- All firmware is coded in ANSI-C
  - Strict ANSI-C for all library peripheral files
    - Relaxed ANSI-C for projects & Examples files.
    - MISRA C 2004 Compliant
- *PPP* is used to reference any peripheral acronym, e.g. *TIM* for Timer.

- Registers & Structures
  - STM32F10x registers are mapped in the microcontroller address space
    - FW library registers have the same names as in STM32F10x Datasheet & reference manual.
  - All registers hardware accesses are performed through a C structures :
    - Work with only one base address and indirect addressing
    - Improve code re-use : e.g. the same structure to handle and initialize 3 USARTs.

STM32 programming: first steps

1

2


Identify & configure the clock frequency at OSC entry
decomment the corresonding line
Determine the peripherals *PPPi* to activate
decomment the peripherals to be used

system_stm32f10x.c

```
/* #define SYSCLK_FREQ_HSE    HSE_Value */
/* #define SYSCLK_FREQ_24MHz  24000000 */
/* #define SYSCLK_FREQ_36MHz  36000000 */
/* #define SYSCLK_FREQ_48MHz  48000000 */
/* #define SYSCLK_FREQ_56MHz  56000000 */
#define SYSCLK_FREQ_72MHz
72000000
```

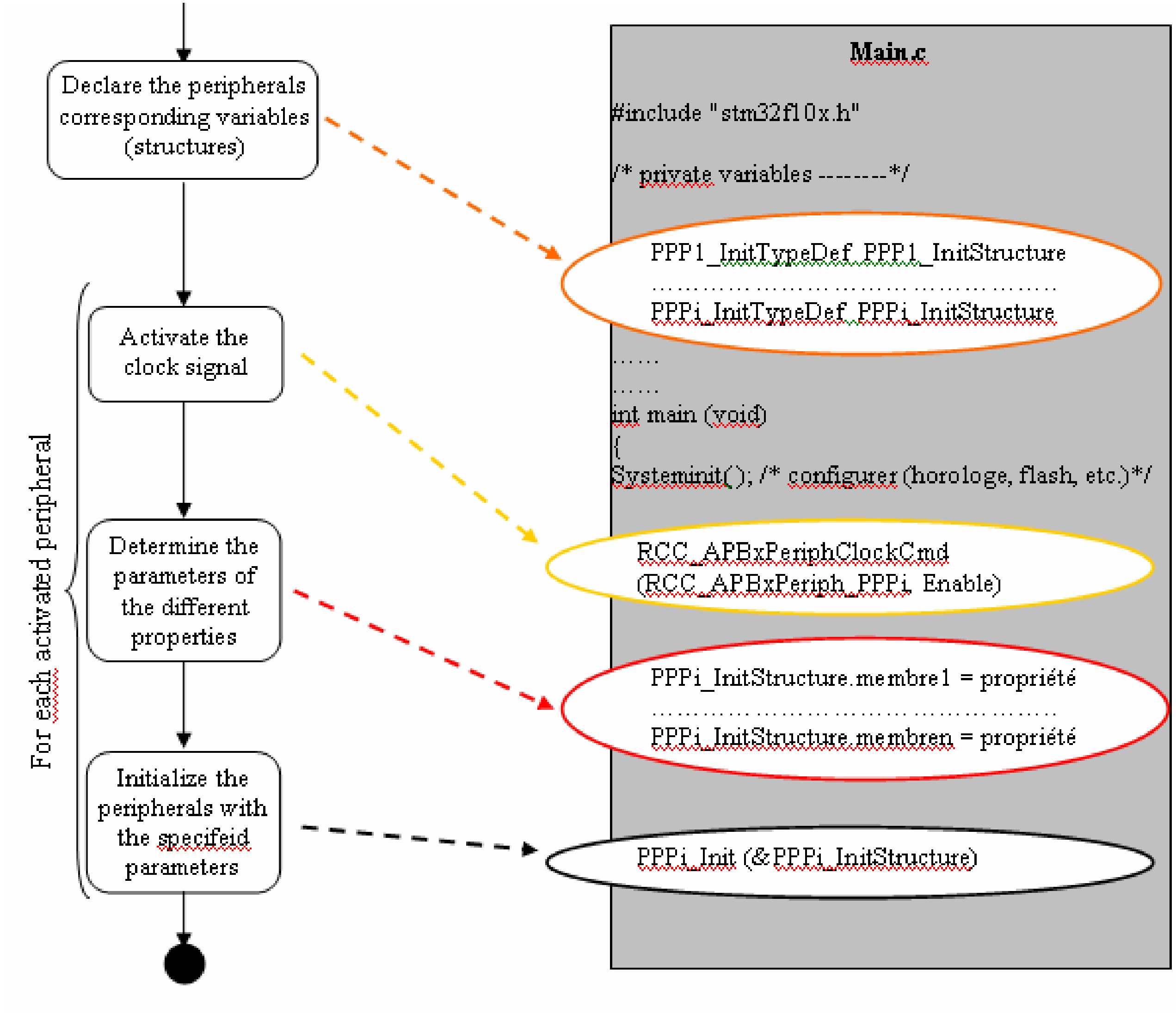
stm32f10x_conf.h

```
/* #include "stm32f10x_adc.h" */
/* #include "stm32f10x_bkp.h" */
#include "stm32f10x_PPP1.h"
/* #include "stm32f10x_can.h" */
/* #include "stm32f10x_crc.h" */
#include "stm32f10x_PPPi.h"
```

STM32 programming: first steps

3

4



Declare the peripherals corresponding variables (structures)
For each activated peripheral
Activate the clock signal
Determine the parameters of the different properties
Initialize the peripherals with the specifeid parameters

Main.c

```
#include "stm32f10x.h"

/* private variables --------*/

PPP1_InitTypeDef PPP1_InitStructure
……………………………………………………
PPPi_InitTypeDef PPPi_InitStructure
......
......
int main (void)
{
Systeminit(); /* configurer (horologe, flash, etc.)*/

RCC_APBxPeriphClockCmd
(RCC_APBxPeriph_PPPi, Enable)

PPPi_InitStructure.membre1 = propriété
……………………………………………………
PPPi_InitStructure.membren = propriété

PPPi_Init (&PPPi_InitStructure)
```

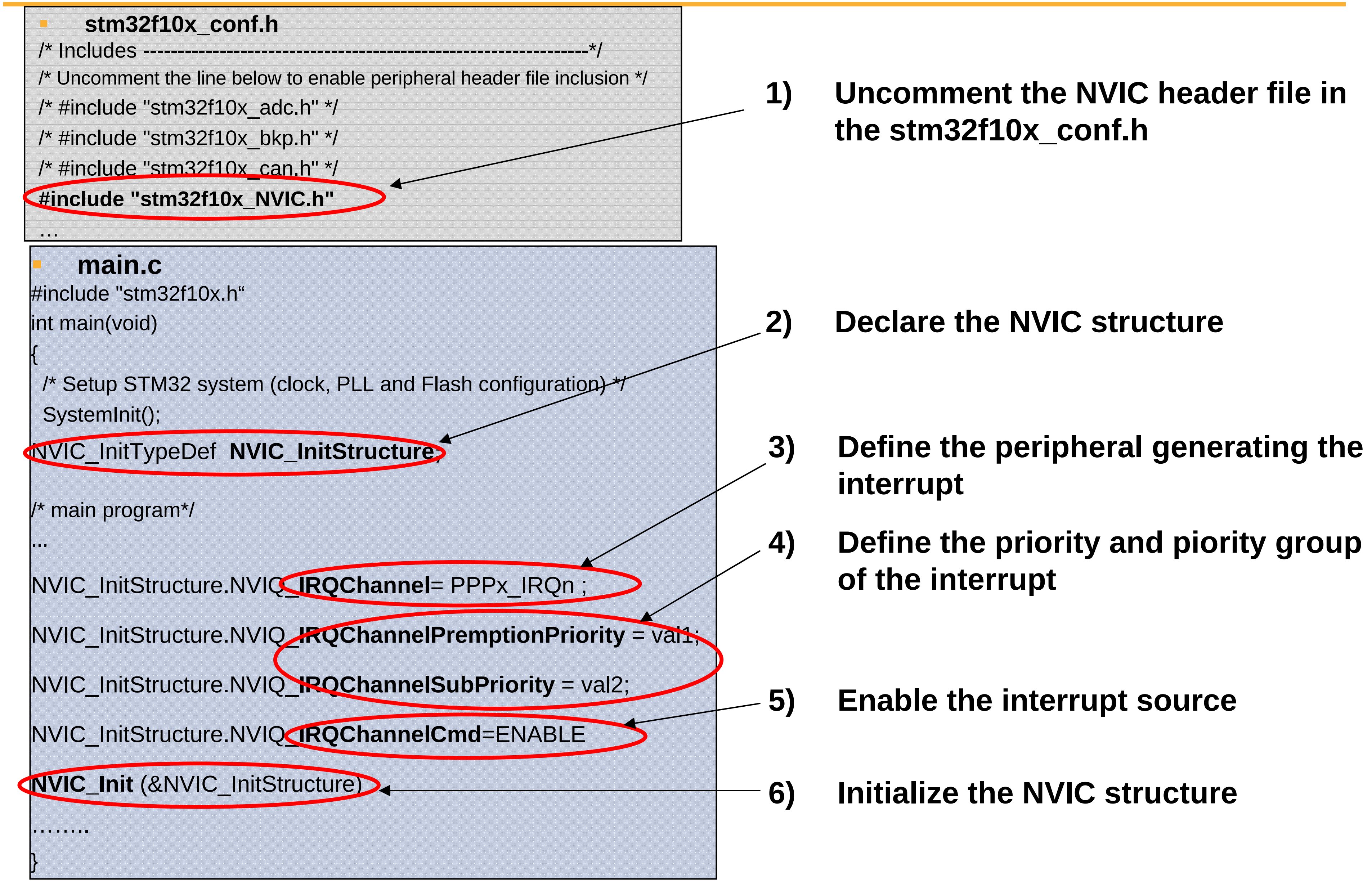
Using the Library (1/2)

- **In the main file , you have to declare a *PPP_InitTypeDef* structure, e.g: PPP_InitTypeDef PPP_InitStructure;**
  - **The PPP_InitStructure is a working variable located in data memory that allows you to initialize one or more instance of PPPs.**
- **You have to fill the *PPP_InitStructure* variable with the allowed values of the structure member.**
  - **Configuration of the whole structure:**
    - **PPP_InitStructure.member1 = val1;**
    - **PPP_InitStructure.member2 = val2;**
    - **...**
    - **PPP_InitStructure.memberN = valN;**
- **Note : The previous initialization could be merged in only one line like the following :**
    - **PPP_InitTypeDef PPP_InitStructure = { val1, val2, …, valn}**
    - **This reduces and optimises code size.**
  - **Configuration of few structure’s members:**
    - **PPP_StructInit(&PPP_InitStructure);**
    - **PPP_InitStructure.memberX = valX;**
    - **PPP_InitStructure.memberY = valY;**

Using the Library (2/2)

- **You have to initialize the PPP peripheral by calling the *PPP_Init(..)* function :**
  - **PPP_Init(PPPx, &PPP_InitStructure);**
- **At this stage the PPP peripheral is initialized and can be enabled by making a call to *PPP_Cmd(..)* function: PPP_Cmd(PPPx, ENABLE);**
- **To access the functionality of the PPP peripheral, the user can use a set of dedicated functions. These functions are specific to the peripheral and for more details refer to STM32F10x Firmware Library User Manual.**
- **Notes :**
- **1) Before configuring a peripheral, you have to enable its clock by calling one of the following functions:**
  - **RCC_AHBPeriphClockCmd(RCC_AHBPeriph_PPPx , ENABLE);**
  - **RCC_APB2PeriphClockCmd(RCC_APB2Periph_PPPx , ENABLE);**
  - **RCC_APB1PeriphClockCmd(RCC_APB1Periph_PPPx , ENABLE);**
- ***2) PPP_DeInit(..)* function can be used to set all PPP's peripheral registers to their reset values:**
  - **PPP_DeInit(PPPx);**
- **3) If after peripheral configuration, the user wants to modify one or more peripheral settings he should proceed as following:**
  - **PPP_InitStucture.memberX = valX;**
  - **PPP_InitStructure.memberY = valY;**
  - **PPP_Init(PPPx, &PPP_InitStructure);**

Programming interrupts



- **stm32f10x_conf.h**

```
/* Includes ------------------------------------------------------------------*/
/* Uncomment the line below to enable peripheral header file inclusion */
/* #include "stm32f10x_adc.h" */
/* #include "stm32f10x_bkp.h" */
/* #include "stm32f10x_can.h" */
#include "stm32f10x_NVIC.h"
…
```

- **main.c**

```
#include "stm32f10x.h“
int main(void)
{
  /* Setup STM32 system (clock, PLL and Flash configuration) */
  SystemInit();
NVIC_InitTypeDef  NVIC_InitStructure;

/* main program*/
...
NVIC_InitStructure.NVIQ_IRQChannel= PPPx_IRQn ;
NVIC_InitStructure.NVIQ_IRQChannelPremptionPriority = val1;
NVIC_InitStructure.NVIQ_IRQChannelSubPriority = val2;
NVIC_InitStructure.NVIQ_IRQChannelCmd=ENABLE
NVIC_Init (&NVIC_InitStructure)
……..
}
```

1) **Uncomment the NVIC header file in the stm32f10x_conf.h**

2) **Declare the NVIC structure**

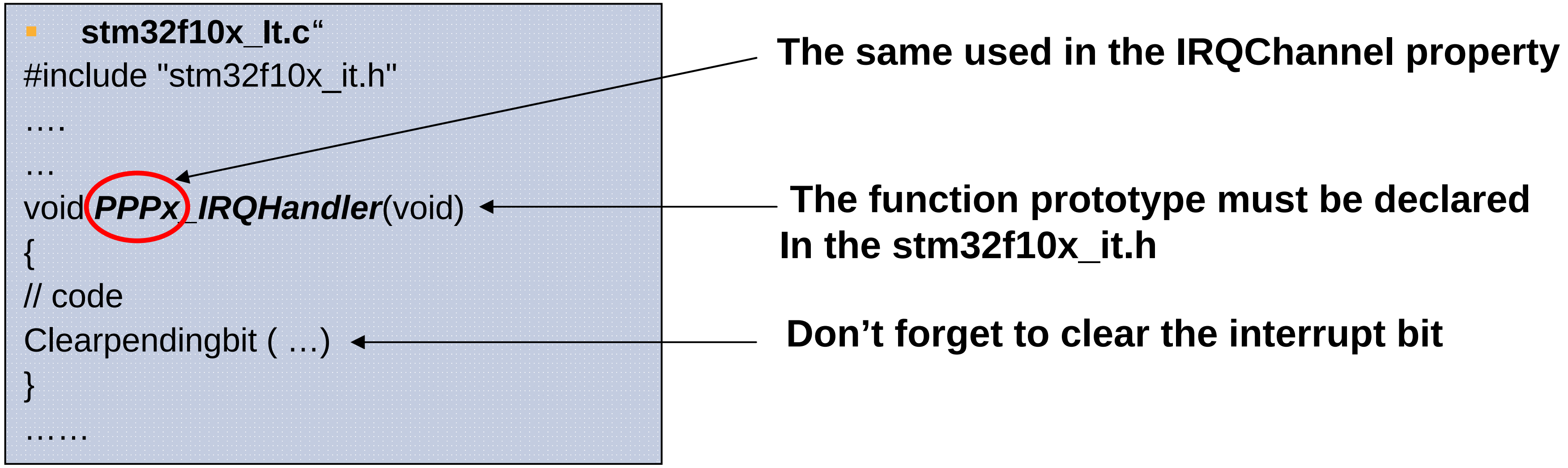
3) **Define the peripheral generating the interrupt**

4) **Define the priority and piority group of the interrupt**

5) **Enable the interrupt source**

6) **Initialize the NVIC structure**

Programming interrupts (2/2)



- **stm32f10x_It.c“**

```
#include "stm32f10x_it.h"
….
…
void PPPx_IRQHandler(void)
{
// code
Clearpendingbit ( …)
}
……
```

The same used in the IRQChannel property

**The function prototype must be declared
In the stm32f10x_it.h**

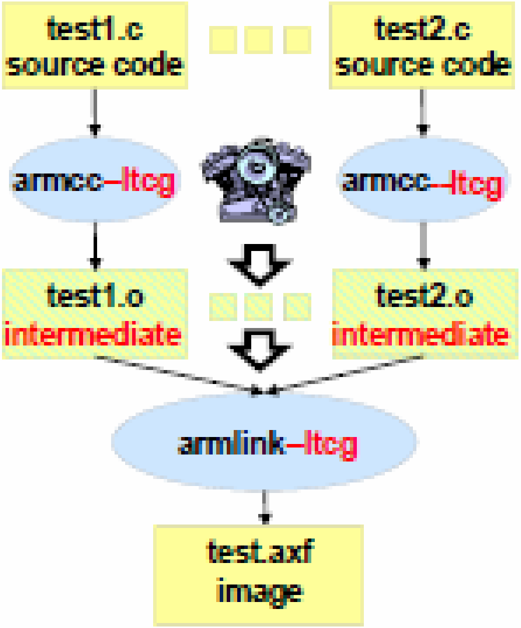
Don’t forget to clear the interrupt bit

Software Developpment tools

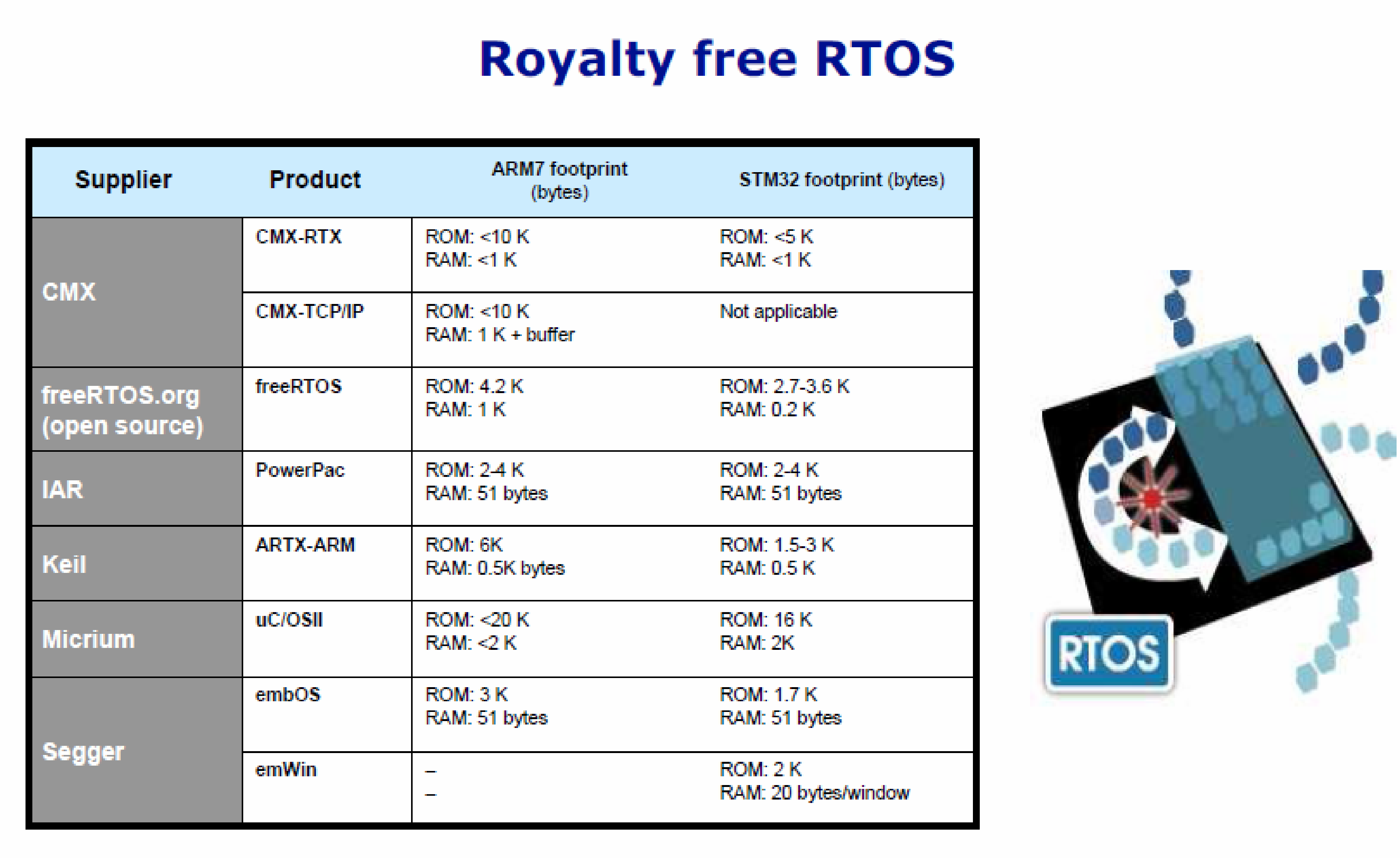
- Raisonnance Ride7:
  - Based on gnu gcc.
  - Free

- Keil µVision
  - Based on ARM compilation tools
  - Includes optimized Library
  - Reduced code size.

- IAR Embedded Workbench



STM32 programming: Available RTOS



Royalty free RTOS

Supplier	Product	ARM7 footprint (bytes)	STM32 footprint (bytes)
CMX	CMX-RTX	ROM: <10 K RAM: <1 K	ROM: <5 K RAM: <1 K
	CMX-TCP/IP	ROM: <10 K RAM: 1 K + buffer	Not applicable
freeRTOS.org (open source)	freeRTOS	ROM: 4.2 K RAM: 1 K	ROM: 2.7-3.6 K RAM: 0.2 K
IAR	PowerPac	ROM: 2-4 K RAM: 51 bytes	ROM: 2-4 K RAM: 51 bytes
Keil	ARTX-ARM	ROM: 6K RAM: 0.5K bytes	ROM: 1.5-3 K RAM: 0.5 K
Micrium	uC/OSII	ROM: <20 K RAM: <2 K	ROM: 16 K RAM: 2K
Segger	embOS	ROM: 3 K RAM: 51 bytes	ROM: 1.7 K RAM: 51 bytes
	emWin	– –	ROM: 2 K RAM: 20 bytes/window

Slide Title – Arial Bold, 32pt

- Content first level – Arial, 28pt
  - Content second level – Arial, 24pt
    - Content third level – Arial, 20pt