



# STM32 Updates for technical development community

Microcontrollers Division

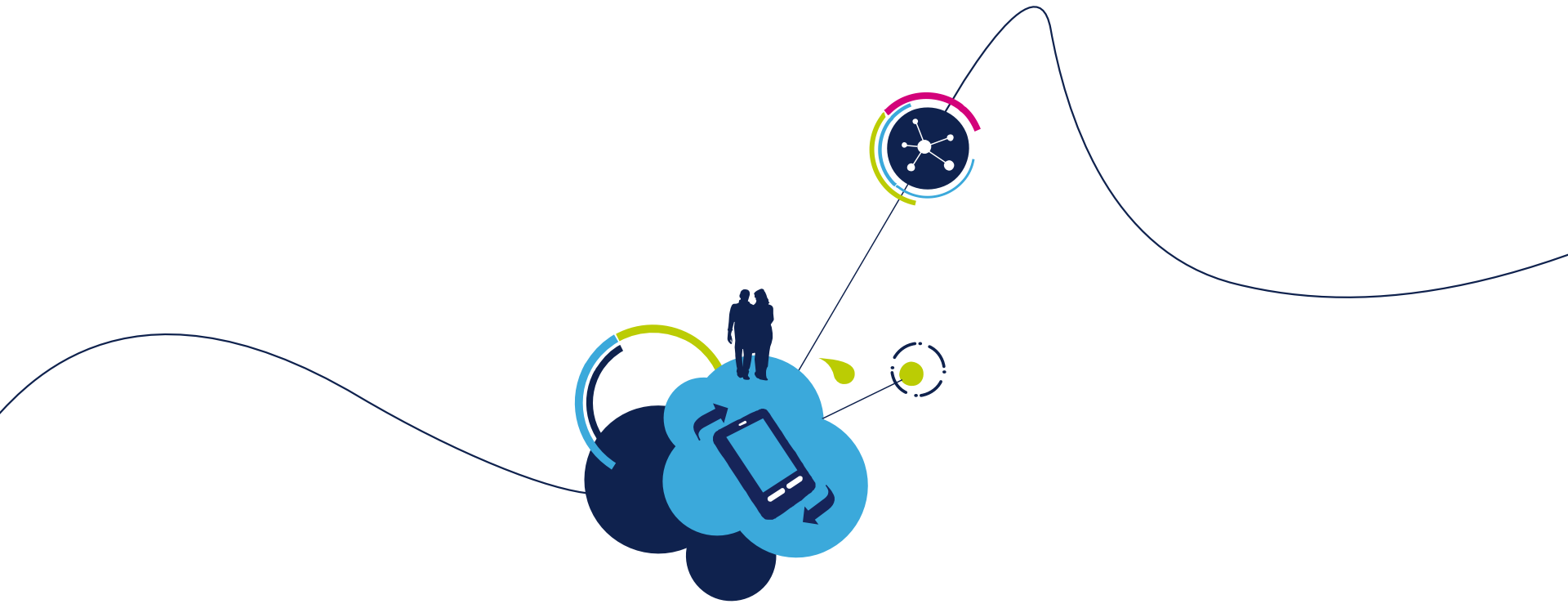
2012 – 2<sup>nd</sup> Issue (July)

- Documents Major Updates

- STM8
- STM32F1/L1
- STM32F2/F4

- Technical presentations

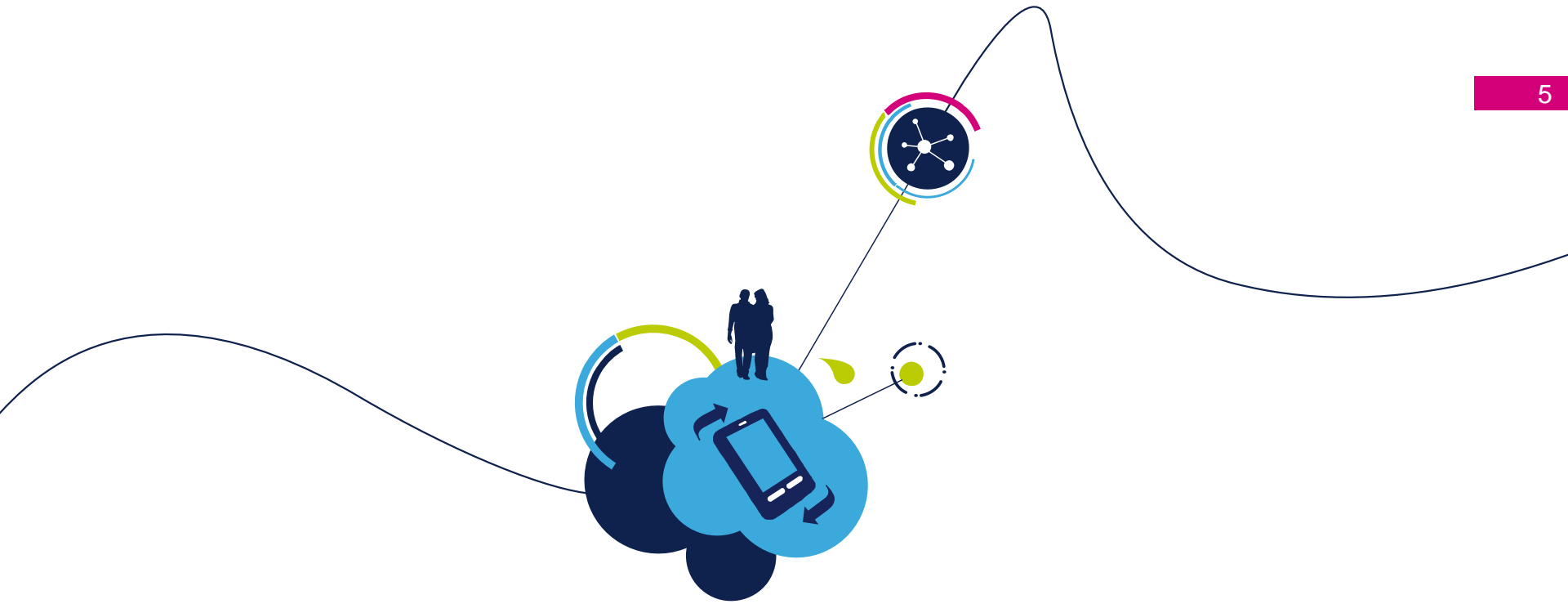
- F0 family updates Introduction
- Cortex-M0 CPU
- STM32F05x system parts
- STM32F05x peripherals



# Documents Major Updates

# Documents Major Updates Overview

- STM8S/A errata Sheet Rev3 **(On Web)**
- STM32F205x and STM32F207x datasheet Rev9 **(On Web)**
- STM32F215x and STM32F217x datasheet Rev7 **(On Web)**
- STM32F40xx and STM32F41xx datasheet Rev3 **(On Web)**
- STM32F103xFG and STM32F101xFG datasheet Rev4 **(On Web Q3)**
- STM32F20x and STM32F21x Errata sheet - Rev3 **(On Web Q3)**  
STM32F40x STM32F41x Errata sheet - Rev3 **(On Web Q3)**



# New STM8A/S beCAN limitation

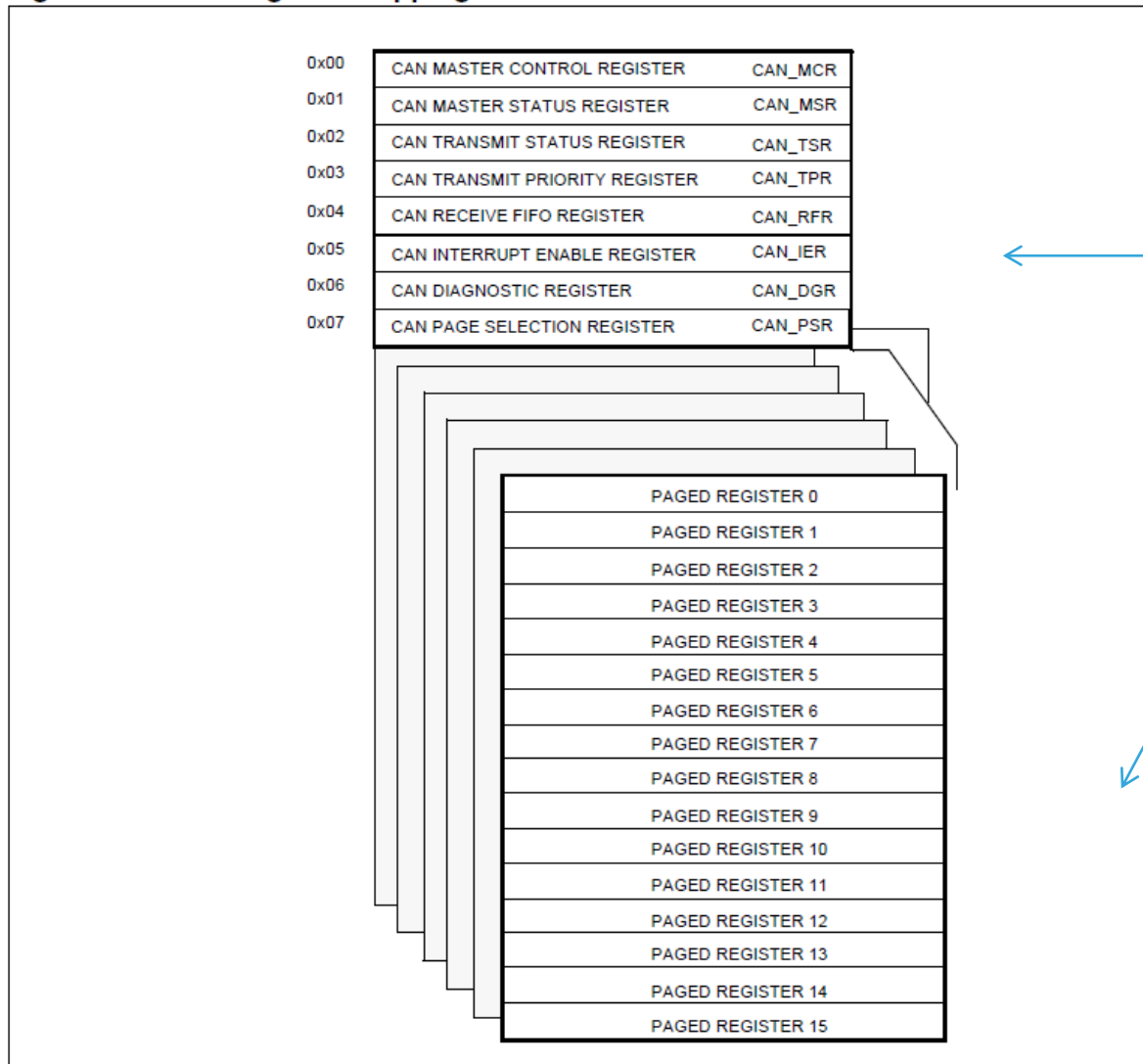
- write in beCAN paged registers
- ignored -

- An issue was discovered in STM8S / STM8A beCAN peripheral
- STM8S208 and STM8AF52xx errata have been updated and published, including this new limitation
  - Write in beCAN paged registers ignored
- A PIL number MMS-MIC/12/7189 has been issued to inform STM8A customers
- STM32 bxCAN and ST72F561 beCAN are NOT impacted at all by this issue

# Objective of this presentation

- To be ready to help customer to do a code review (in C source code, and generated assembly code) to confirm the application is not impacted by the issue

Figure 156. CAN register mapping



- Two kinds of registers:
  - True registers
  - Paged registers are physically a RAM area



# beCAN paged registers

Figure 157. CAN page mapping

	PAGE 0	PAGE 1	PAGE 2	PAGE 3	PAGE 4
0x00	CAN_MCSR	CAN_MCSR	CAN_F0R1	CAN_F2R1	CAN_F4R1
0x01	CAN_MDLCR	CAN_MDLCR	CAN_F0R2	CAN_F2R2	CAN_F4R2
0x02	CAN_MIDR1	CAN_MIDR1	CAN_F0R3	CAN_F2R3	CAN_F4R3
0x03	CAN_MIDR2	CAN_MIDR2	CAN_F0R4	CAN_F2R4	CAN_F4R4
0x04	CAN_MIDR3	CAN_MIDR3	CAN_F0R5	CAN_F2R5	CAN_F4R5
0x05	CAN_MIDR4	CAN_MIDR4	CAN_F0R6	CAN_F2R6	CAN_F4R6
0x06	CAN_MDAR1	CAN_MDAR1	CAN_F0R7	CAN_F2R7	CAN_F4R7
0x07	CAN_MDAR2	CAN_MDAR5	CAN_F0R8	CAN_F2R8	CAN_F4R8
0x08	CAN_MDAR3	CAN_MDAR6	CAN_F1R1	CAN_F3R1	CAN_F5R1
0x09	CAN_MDAR4	CAN_MDAR4	CAN_F1R2	CAN_F3R2	CAN_F5R2
0x0A	CAN_MDAR5	CAN_MDAR5	CAN_F1R3	CAN_F3R3	CAN_F5R3
0x0B	CAN_MDAR6	CAN_MDAR6	CAN_F1R4	CAN_F3R4	CAN_F5R4
0x0C	CAN_MDAR7	CAN_MDAR7	CAN_F1R5	CAN_F3R5	CAN_F5R5
0x0D	CAN_MDAR8	CAN_MDAR8	CAN_F1R6	CAN_F3R6	CAN_F5R6
0x0E	CAN_MTSRL	CAN_MTSRL	CAN_F1R7	CAN_F3R7	CAN_F5R7
0x0F	CAN_MTSRH	CAN_MTSRH	CAN_F1R8	CAN_F3R8	CAN_F5R8
	Tx Mailbox 0 PAGE 5	Tx Mailbox 1 PAGE 6	Acceptance Filter 0:1 PAGE 7	Acceptance Filter 2:3	Acceptance Filter 4:5
0x00	CAN_MCSR	CAN_ESR	CAN_MFMIR		
0x01	CAN_MDLCR	CAN_EIER	CAN_MDLCR		
0x02	CAN_MIDR1	CAN_TECR	CAN_MIDR1		
0x03	CAN_MIDR2	CAN_RECR	CAN_MIDR2		
0x04	CAN_MIDR3	CAN_BTR1	CAN_MIDR3		
0x05	CAN_MIDR4	CAN_BTR2	CAN_MIDR4		
0x06	CAN_MDAR1	Reserved	CAN_MDAR1		
0x07	CAN_MDAR2	Reserved	CAN_MDAR2		
0x08	CAN_MDAR3	CAN_FMR1	CAN_MDAR3		
0x09	CAN_MDAR4	CAN_FMR2	CAN_MDAR4		
0x0A	CAN_MDAR5	CAN_FCR1	CAN_MDAR5		
0x0B	CAN_MDAR6	CAN_FCR2	CAN_MDAR6		
0x0C	CAN_MDAR7	CAN_FCR3	CAN_MDAR7		
0x0D	CAN_MDAR8	Reserved	CAN_MDAR8		
0x0E	CAN_MTSRL	Reserved	CAN_MTSRL		
0x0F	CAN_MTSRH	Reserved	CAN_MTSRH		
	Tx Mailbox 2 (if TXM2E=1 in CAN_DGR register)	Configuration/Diagnostic	Receive FIFO		

- Paged registers consist of
  - TX mailbox 0, 1, 2
  - RX filters
  - Config / diag
  - RX FIFO
- Paged register (RAM) can be accessed concurrently by the **CPU** (software) or by the **CAN core**
- An HW arbitration mechanism is handling concurrent access

# Issue conditions & description

- In very specific conditions, a write to the beCAN paged registers may be ignored. This occurs when the **CPU is writing twice or more** into beCAN paged registers in **consecutive master clock cycles**, and **during the second or further writes**, the **CAN 2.0B active core is accessing** (read or write) one of the paged registers.
- A typical case is when the CPU is writing constants into paged registers (typically a mailbox for transmission) while the CAN 2.0B active core is reading the filters during a frame reception, or writing the FIFO after a frame reception. The beCAN paged registers range is from address 0x5428 to address 0x5437.
- Note: CAN 2.0B active core does not access the paged registers as long as the beCAN is in initialization mode. Therefore the issue can not occur while the software is writing into the paged registers to initialize the beCAN (filters configuration for example).
- The CPU write into two paged registers in two consecutive master clock cycles may only happen in case of **consecutive single-cycle load and transfer instructions or single-cycle bit operation instructions**, when the destination is an address within the range 0x5428 to 0x5437 inside the beCAN peripheral. The instructions are single-cycle if the source is a constant (embedded in instruction opcode) or in the A accumulator (when the same value has to be stored in two or more CAN registers).

# Impact on application & workaround

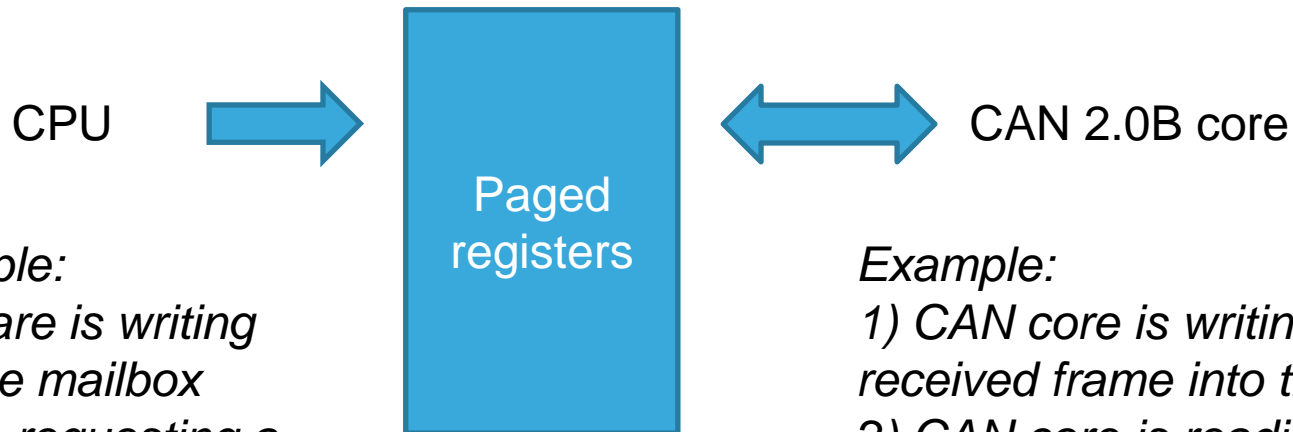
- Impact on application

- When all the specific conditions are met, a corrupted frame may be sent on the CAN bus

- Workaround

- In case you need to write constants consecutively into paged registers, insert a NOP instruction in between each write. You can make use of inline assembly in C code.

# Issue conditions & description



*Example:  
Software is writing  
into the mailbox  
before requesting a  
transmission*

*Example:  
1) CAN core is writing a  
received frame into the FIFO  
2) CAN core is reading the  
filters to know if the frame  
must be stored or not in the  
FIFO*

# Instructions able to generate consecutive write operations (critical case)

13

- MOV longmem, #byte ; LD longmem, A ; CLR longmem

- Consecutive MOV instructions with immediate addressing mode

MOV 0x5429, #0x08

MOV 0x542a, #0x0d

MOV 0x542b, #0x40

- Consecutive LD instructions with A register as source

LD 0x542d, A

LD 0x542e, A

- CLR instruction followed by another single-cycle instruction

CLR 0x542d

MOV 0x542e, #0x40

- Mix of single-cycle instructions

LD 0x5429, A

MOV 0x542a, #0x0d

Only **single cycles instructions** may generate the critical case

Destination must be a **beCAN paged register** address

Source must be **constant** or **A**

These instructions must be **consecutive** (no other instruction between)

# Other very unlikely instructions able to generate consecutive write operations

- Another very unlikely case is indirect or indexed addressing with beCAN address loaded in X or Y registers, which also generates single-cycle instructions.
  - LD (X), A ; LD (shortoff,X),A ; LD (longoff,X),A ; LD (Y), A ; LD (shortoff,Y) ; LD (longoff, Y) ; CLR (X) ; CLR (shortoff,X) ; CLR (longoff,X) ; CLR(Y) ; CLR (shortoff,Y) ; CLR (longoff,Y) ; CLR (shortoff, SP)
- This can also happen with a single 2-cycle LDW instruction, if two consecutive registers are written with a 16-bit data, with X or Y register as source.
  - List of instructions: LDW longmem, X ; LDW longmem, Y
    - LDW 0x542a, X ; with X containing 0x0d40.

# Example of critical code

- C code with constants usage

```
#define MSG_ID 0x0350
CAN_P1 = 0x08;
CAN_P2 = (MSG_ID >> 6) & 0x1F;
CAN_P3 = (MSG_ID << 2) & 0xFC;
```

- Generated assembly code

```
35085429    mov    _CAN_P1,#8
350d542a    mov    _CAN_P2,#13
3540542b    mov    _CAN_P3,#64
```

- Insert NOP in C code in case of constants usage

```
#define MSG_ID 0x0350
CAN_P1 = 0x08;
asm("NOP");
CAN_P2 = (MSG_ID >> 6) & 0x1F;
_asm("NOP");
CAN_P3 = (MSG_ID << 2) & 0xFC;
```



- C code with variables usage

```
MyBuffer1 = 0x08;  
MyBuffer2 = (MSG_ID >> 6) & 0x1F;  
MyBuffer3 = (MSG_ID << 2) & 0xFC;  
CAN_P1 = MyBuffer1;  
CAN_P2 = MyBuffer2;  
CAN_P3 = MyBuffer3;
```

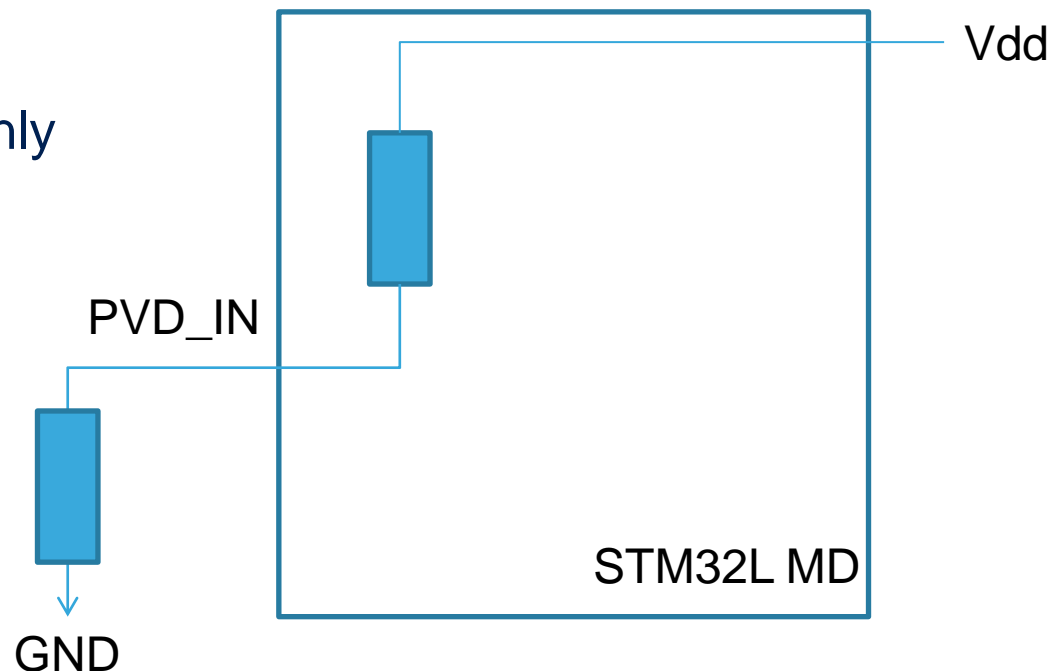
- Generated assembly code

```
35080003    mov    _MyBuffer1,#8  
350d0004    mov    _MyBuffer2,#13  
35400005    mov    _MyBuffer3,#64  
5500035429    mov    _CAN_P1,_MyBuffer1  
550004542a    mov    _CAN_P2,_MyBuffer2  
550105542b    mov    _CAN_P3,_MyBuffer3
```

- CAN drivers from Vector (third party) and STM8S/A standard peripheral software library should not be impacted by the issue, because some intermediate buffers are used to fill the mailboxes
- Nevertheless, it's recommended to crosscheck inside the C code and especially in generated assembly code

# STM32L MD Errata: PVD\_IN pull-up resistor is always active

- Once PB7 (PVD\_IN) is configured in the analog input mode and PVD level selection  $PLS[2:0] = 111$   
External input analog voltage is compared internally to VREFINT
- The internal pull-up resistor is activated even if the Programmable Voltage Detector (PVD) is disabled by PVDE bit causing unwanted current consumption
- Medium Density devices only



# STM32F2/F4xx datasheets Update

- STM32F205x and STM32F207x
  - Starting from rev "Y" on STM32F205x and STM32F215x devices, the USB 2.0 full-speed device/host/OTG controller with on-chip PHY peripheral is now available.
- STM32F405x and STM32F407x
  - Starting from rev "Z" on STM32F405x and STM32F415x devices, the USB 2.0 full-speed device/host/OTG controller with on-chip PHY peripheral is now available.
  - Full information on WLCSP90 package added with corresponding part numbers
- A PIL will be sent to our customers in Q3/2012

# STM32F2/4 Errata Sheet Update (1/8)

- AHB2APB Bridge could lock the master if APB clock is slowing down during the transfer:
  - **Description :**
    - A limitation could appear in case of a master (CPU or DMA) modify the APB clock (slow down the clock) while another master is performing write access to the same APB. In case of an APB prescaler modification (1-> 2, 1 -> 4, 1 -> 8, 1 -> 16) while another master is performing a write access to this target APB, the AHB2APB Bridge could lock the master as a consequence the master performing the write access will freeze and stop his current access.
  - **Workaround**
    - Avoid clock slow down on APB while a master is performing write access to this APB peripheral when AHB/APB prescaler is equal to 1.

# STM32F2/4 Errata Sheet Update (2/8)

- Non bufferable access to RTC and IWDG could be managed as bufferable access:
  - **Description**
    - A limitation could appear in case of the CPU performs non bufferable access to the RTC or IWDG register, the access could be treated as bufferable if there is no APB prescaler configured (AHB/APB prescaler is equal to 1).
    - In case of an APB prescaler modification (1-> 2, 1 -> 4, 1 -> 8, 1 -> 16) while another master is performing a write access to this target APB, the AHB2APB Bridge could lock the master. If consecutive accesses are performed only the last one is impacted.
  - **Workaround**
    - If the non bufferable attribute is required on this register the user could perform a read after the write to guaranty the completion of the write access.

# STM32F2/4 Errata Sheet Update (3/8)

- DMA2 could corrupt data when it managed AHB and APB peripherals in a concurrent way
  - **Description**
    - A limitation could appear when the DMA2 managed in a concurrent way AHB (only peripherals embedding FIFOs) and APB transfer. This will generate data corruption (multiple DMA access). When this limitation occurs, the data transferred by the DMA to the AHB peripherals could be corrupted in case of FIFO target,
    - For memory the impact is a multiple access (not visible by the user) and the data is not corrupted. For DCMI a multiple unacknowledged request could be generated that imply an unknown behavior of the DMA. AHB Peripherals embedding FIFO are DCMI, CRYPT, and HASH. On sales types without CRYPTO only the DCMI is impacted. External FIFO controlled by the FSMC is also impacted.
  - **Workaround**
    - Avoid concurrent AHB (DCMI, CRYPT, HASH, FSMC with external FIFO) and APB transfer management using DMA2.

# STM32F2/4 Errata Sheet Update (4/8)

- Flash noise impacting ADC accuracy
  - **Description**
    - Embedded Flash noise generated on VDD supplies may impact ADC accuracy; this noise is always active whatever the power mode of the product.
  - **Workaround**
    - Two steps could be applied to adapt accuracy level to the application requirements:
      - Configure the Flash ART as Prefetch OFF and (Data + Instruction) cache ON
      - Use averaging and filtering algorithms on ADC output codes
- For more workaround details of this limitation, please refer to AN4073.  
(On Web Q3-2012)



# STM32F2/4 Errata Sheet Update (5/8)

- Battery charge monitoring limitation

- Description

- A limitation could appear in case  $VDD = VDDA$  is  $\leq 2.4V$ , Vbat conversion correctness is not guaranteed in full range because:

- When Vbat is set, the voltage divider bridge is enabled and  $Vbat/2$  is connected to the ADC. In order to monitor the battery charge correctly, the input of the ADC must not be higher than  $(Vdda-0.6V)$ . It means  $Vbat/2 < Vdd - 0.6v \rightarrow VDD > 2,4$ ,

- Workaround

- None,  $VDD = VDDA$  should be more than 2.4V

# STM32F2/4 Errata Sheet Update (6/8)

- **SDIO clock divider BYPASS mode is not working properly**
  - **Description**
    - For High speed communication mode, when SDIOCLK is equal to 48MHz (PLL48\_output = 48MHz), BYPASS bit equal to '1' and NEGEDGE bit equal to '0' (respectively bit 10 and bit 13 in SDIO\_CLKCR register) hold timing at the chip I/O pin is not inline with SD/MMC specs
  - **Workaround**
    - When not using USB nor RNG, PLL48\_output (respectively SDIOCLK ) frequency can be raise up to 75MHz, allowing to reach 37.5 MHz on SDIO\_CK in High Speed mode. With BYPASS bit equal to '0', CLKDIV bits equal to '0', and NEGEDGE bit equal to '0'.
- **SDIO clock dephasing (NEGEDGE) is not working properly**
  - **Description**
    - When NEGEDGE bit is set to '1' it may leads to invalid data, command response read.
  - **Workaround**
    - No workaround available, NEGEDGE bit equal to '1' configuration should not be used.

# STM32F2/4 Errata Sheet Update (7/8)

- Delay between RCC write and peripheral enable takes effect
  - Description
    - A Delay between RCC peripheral clock enable and the effective peripheral enable should be taking into account in order to manage the peripheral read/write.  
This delay depends on the peripheral mapping:
      - If the peripheral is mapped on AHB: delay should be equal to 2 AHB cycle,
      - If the peripheral is mapped on APB: delay should be equal to 1 + AHB\_2\_APB prescaler
  - Workaround
    - Use DSB instruction to stall pipeline until instruction is completed
    - Insert two NOPs between RCC enable bit write and peripheral register write

# STM32F2/4 Errata Sheet Update (8/8)

- DAC, DMA Underrun flag : management issue

- Description

- A limitation could appear if DMA was not fast enough to input the next digital data to the DAC, As a consequence, the same digital data is converted twice. In these conditions, DMAUDR flag is set which lead normally to disable of DMA data transfers.
- This is not the case: DMA is not disabled by DMAUDR=1 and keeps servicing the DAC,

- Workaround

- None

- DAC, DMA request is not automatically cleared by DMAEN=0

- Description

- A limitation could appear if the application want to stop the current DMA to DAC transfer: DMA request is not automatically cleared by DMAEN=0, or by DACEN=0, If the application stops the DAC operation while the DMA request is high, DMA request will keep pending while DAC is re-initialized and re-started. With the risk that a spurious-unwanted DMA request is serviced as soon as DAC is re-enabled.

- Workaround

- Ensure that the current DMA stream is completed before of stopping DAC operation with DMA.

- Documents Major Updates
  - STM8
  - STM32F1/L1
  - STM32F2/F4
- Technical presentations
  - F0 family updates Introduction
  - Cortex-M0 CPU
  - STM32F05x system parts
  - STM32F05x peripherals

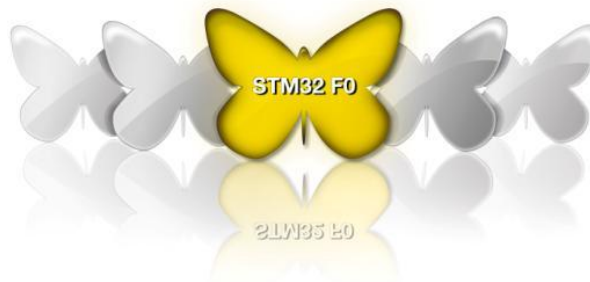


# STM32 F-0 Series For Cost-sensitive applications

Technical Updates

June '12

STM32  Releasing your **creativity**



life.augmented

# Cortex-M processors

- Forget traditional 8/16/32-bit classifications
  - Seamless architecture across all applications
  - Every product optimised for ultra low power and ease of use

<b>Cortex-M0</b>	<b>Cortex-M3</b>	<b>Cortex-M4</b>
“8/16-bit” applications	“16/32-bit” applications	“32-bit/DSC” applications

**Binary and tool compatible**



# STM32F05x For Cost-sensitive applications

- ARM® Cortex™ - M0
  - Low cost, low power core
  - 0.9 DMIPS/MHz at 48 MHz max
  - DMA on all products
- Enhanced features for appliances, consumer and industrial
  - Eleven PWM 16-bit timers including motor control timer
  - Full analog set - fast 1.0  $\mu$ s 12-bit ADC, 12-bit DAC, comparators
  - Consumer Electronic Control (CEC) hardware function
- From 16-Kbyte up to 64-Kbyte Flash
- From 28-pin to 64-pin packages



# Class B compliance

33

- Home appliance market regulation requires Class B certification
  - Our safety features are ready to support Class B (EN/IEC 60335-1) certification requirements
  - Due to HW implementation, some SW checks could be removed or simplified
    - CRC FLASH check at startup – CRC block available
    - HW parity on RAM instead SW run time checks
    - CSS clock monitoring
  - We provide a library for Class B applications (similar to STM32F1 / STM8)
    - Certification to be performed soon on STM32F0



# STM32F0x Series 64KB STM32F051

**ARM 32-bit Cortex-M0 CPU**

**Operating Voltage:**

- VDD = 2.0 V to 3.6 V
- VBAT = 1.65 V to 3.6 V

**Safe Reset System (Integrated Power On Reset (POR)/Power Down Reset (PDR) + Programmable voltage detector (PVD))**

**Embedded Memories:**

- FLASH: up to 64 Kbytes
- SRAM: up to 8 Kbytes

**CRC calculation unit**

**5 Channels DMA**

**Power Supply with software configurable internal regulator and low power modes.**

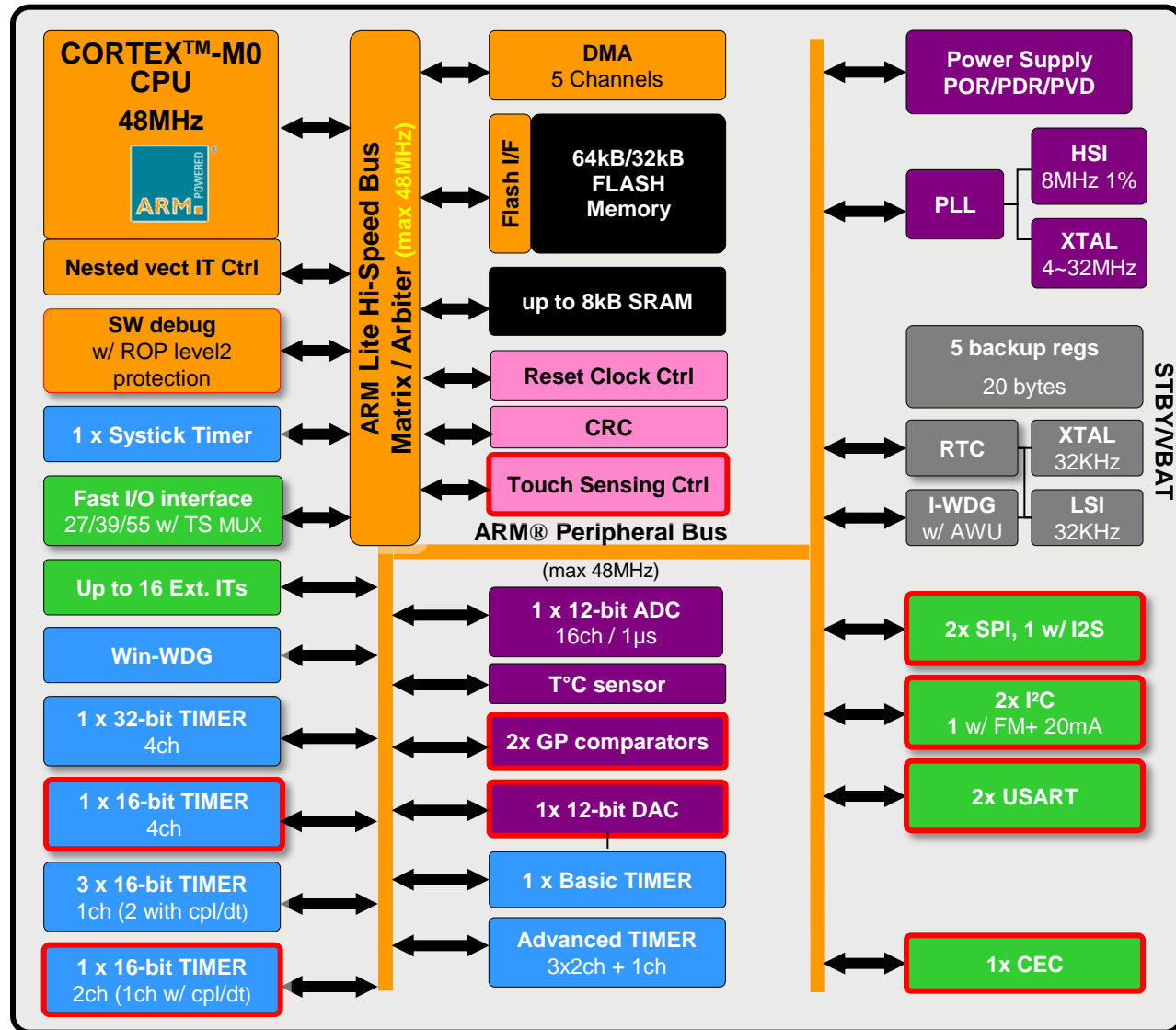
**Low Power Modes with Auto Wake-up**

**Low power calendar RTC with 20 bytes of backup registers**

**Up to 48 MHz frequency managed & monitored by the Clock Control w/ Clock Security System**

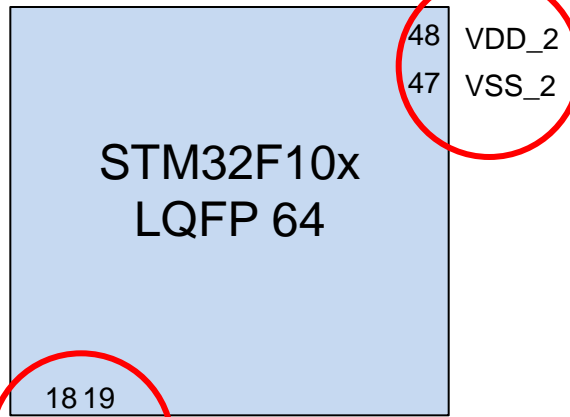
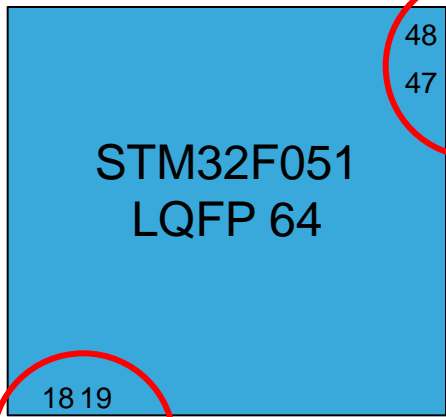
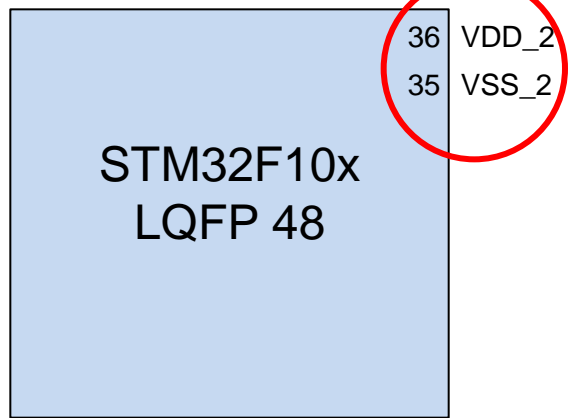
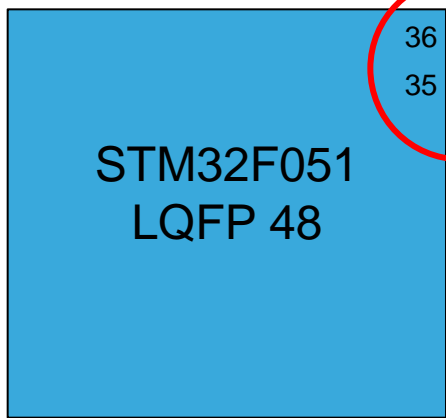
**Rich set of peripherals & IOs**

- 1 x 12-bit DAC with output buffer
- 2 low power comparators (Window mode and wakeup)
- Dual Watchdog Architecture
- 11 Timers w/ advanced control features (including Cortex SysTick, RTC and WDGs)
- 7 communications Interfaces
- Up to 55 fast I/Os all mappable on external interrupts/event
- 1x12-bits 1MSPS ADC w/ up to 16 external channels + Temperature sensor/ voltage reference/VBAT measurement



Note: Features outlined in red are not present on STM32F050 product (just one I²C, SPI & USART)

# Pin Compatibility across STM32 Family



✓ PF4, PF5, PF6 and PF7 added on STM32F051

Increased I/O versus Supply pins ratio with pin-to-pin compatibility

# STM32F05x Documents Major Updates

- **STM32F05x Reference Manual – Rev1 (on web)**
- **STM32F05x Cortex-M0 Programming Manual – Rev1 (on web)**
- **STM32F051 Datasheet – Rev1 (on web)**
- **STM32F050 Datasheet – Rev1 (on web)**
- **STM32F051 Erratasheet – Rev1 (On Web Q3)**
- **STM32F050 Erratasheet – Rev1 (On Web Q3)**



# STM32F0x Core and System Technical Introduction

# M0 – a low cost Cortex-M processor

38

- Address “low end” applications, as a 8/16-bit MCU replacement
  - Similar gate count to 16-bit processors
  - For high end/processing intensive applications, the global MCU price point is lower than most of 8/16-bit processors due to lower code memory footprint (similar, in a few percent range to the Cortex-M3)
  - High performance, only 25-30% lower than the standard Cortex-M3 32-bit architecture
- Very good power and area optimization
  - Designed for low cost and low power
- Automatic state saving on interrupt and exceptions
  - Low software overhead on exception entry and exit
- Deterministic instruction execution timing
  - Instructions always take same time to execute, from a deterministic memory system

# Cortex-M0 processor microarchitecture

39

- **ARMv6M Architecture**

- Thumb-2 Technology
- Integrated configurable NVIC
- Compatible with Cortex-M3

- **Microarchitecture**

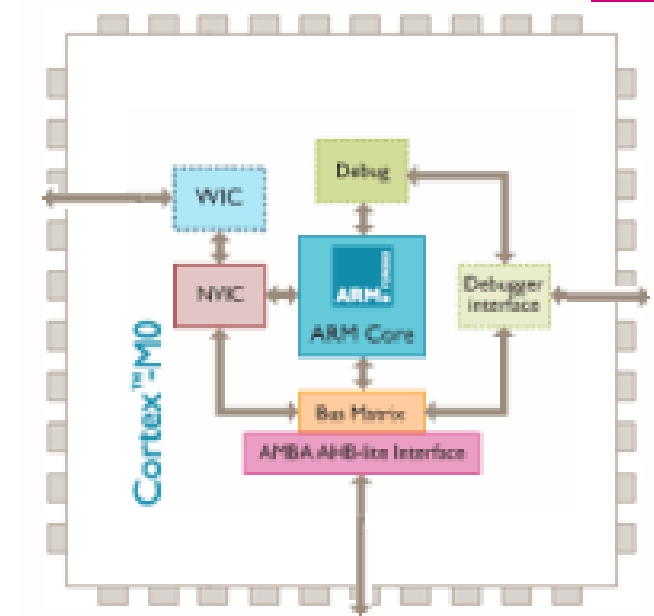
- 3-stage pipeline with branch speculation
- 1x AHB-Lite Bus Interfaces

- **Configurable for ultra low power**

- Deep Sleep Mode, Wakeup Interrupt Controller (*Not available for STM32F05*)

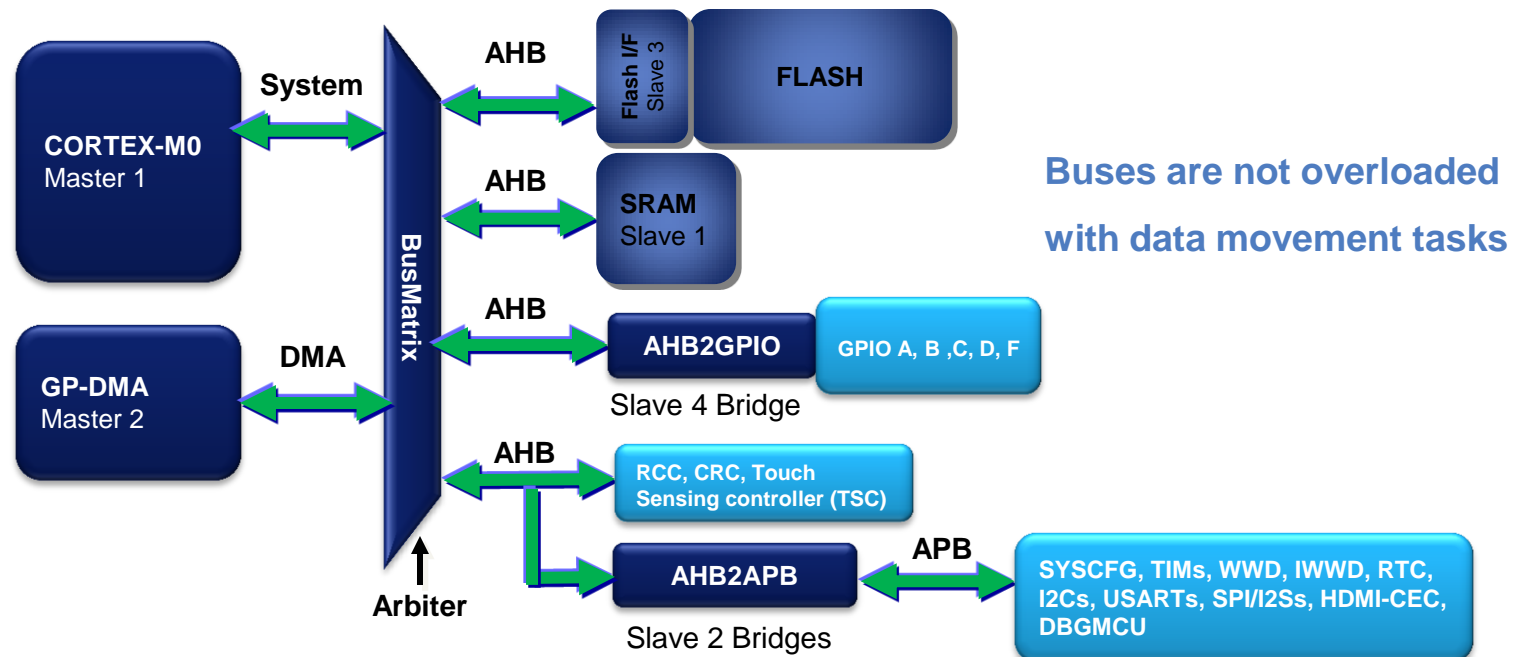
- **Flexible configurations for wider applicability**

- Configurable Interrupt Controller (1-32 Interrupts and Priorities)
- No Memory Protection Unit



 Optional Debug & Trace

- **Multiply possibilities of bus accesses to SRAM, Flash, Peripherals, DMA**
  - BusMatrix added to allow parallel access
- **Efficient DMA and Rapid data flow**
  - Direct path to SRAM through arbiter, guarantees alternating access
  - Von Neumann + BusMatrix allows Flash execution in parallel with DMA transfer
- **Increase Peripherals Speed for better performance**
  - Advanced Peripheral bus (APB) architecture up to 48MHz
  - ➔ Allows to optimize use of peripherals (18Mbits/s SPI, 6Mbps USART, 48MHz GP-Timer, 12MHz toggling I/Os)





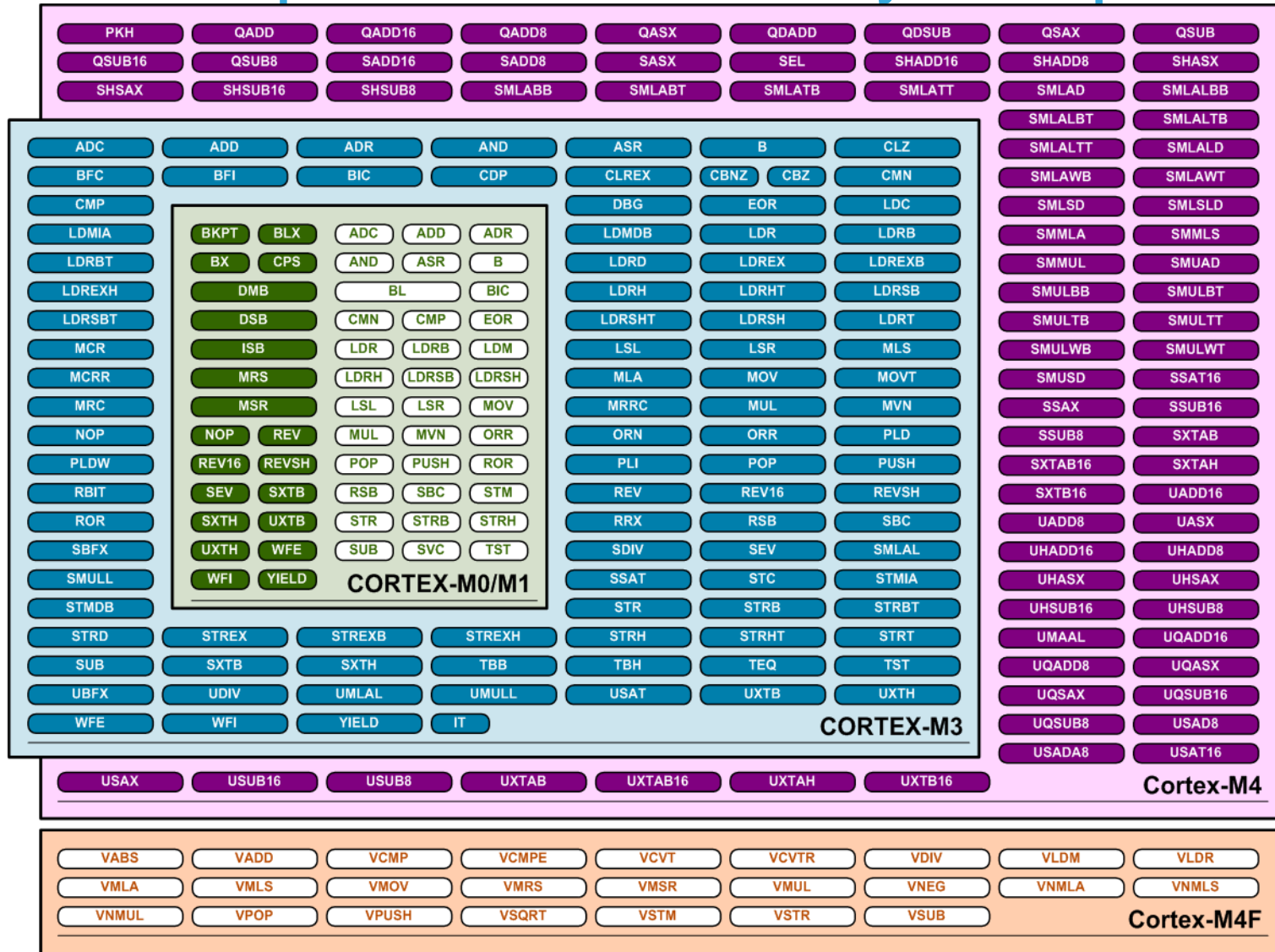
# Cortex-M feature set comparison

	<b>Cortex-M0</b>	<b>Cortex-M3</b>	<b>Cortex-M4</b>
Architecture Version	<b>V6M</b>	v7M	v7ME
Instruction set architecture	<b>Thumb, Thumb-2 System Instructions</b>	Thumb + Thumb-2	Thumb + Thumb-2, DSP, SIMD, FP
DMIPS/MHz	<b>0.9</b>	1.25	1.25
Bus interfaces	<b>1</b>	3	3
Integrated NVIC	<b>Yes</b>	Yes	Yes
Number interrupts	<b>1-32 + NMI</b>	1-240 + NMI	1-240 + NMI
Interrupt priorities	<b>4</b>	8-256	8-256
Breakpoints, Watchpoints	<b>4/2/0, 2/1/0</b>	8/4/0, 2/1/0	8/4/0, 2/1/0
Memory Protection Unit (MPU)	<b>No</b>	Yes (Option)	Yes (Option)
Integrated trace option (ETM)	<b>No</b>	Yes (Option)	Yes (Option)
Fault Robust Interface	<b>No</b>	Yes (Option)	No
Single Cycle Multiply	<b>Yes (Option)</b>	Yes	Yes
Hardware Divide	<b>No</b>	Yes	Yes
WIC Support	<b>Yes</b>	Yes	Yes
Bit banding support	<b>No</b>	Yes	Yes
Single cycle DSP/SIMD	<b>No</b>	No	Yes
Floating point hardware	<b>No</b>	No	Yes (Option)
Bus protocol	<b>AHB Lite</b>	AHB Lite, APB	AHB Lite, APB
CMSIS Support	<b>Yes</b>	Yes	Yes

# Cortex-M0 differences vs Cortex-M3

- Less Breakpoints/Watchpoints (4/2)
- Only Serial-Wire-Debug (SWD) to save JTAG pins but on same 20-pins connector
- Only one I/F ( AHB lite bus) to connect memories, external bus Matrix, etc....
- Cortex-M0 does not support different privilege levels, Software execution is always privileged, meaning software can access all the features of the processor.
- No Bit-Banding
- No support of un-aligned data

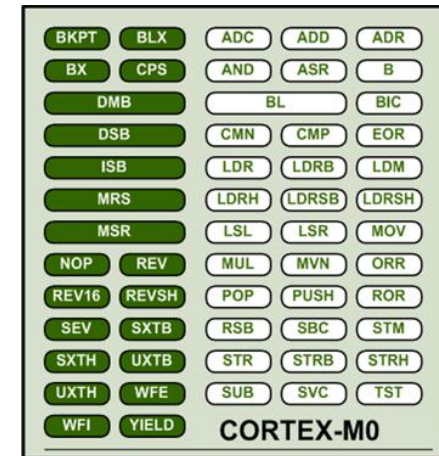
# Cortex-M processors binary compatible



# Cortex-M0 differences vs Cortex-M3

- V6M Architecture : Only 56 Instructions

- 46 instructions ( generally generated by a C compiler)
- 10 instructions for System & special usage
- Can run a Traditional ARM7/9 THUMB instructions
- 8/16/32-bits data transfers possible on One Instruction
- No Hardware Divide
- No IT ( If-Then-Else) blocks to avoid small Branches
- All instructions are 16-bits Thumb to save code memory,
- Except the 32-bit Thumb instructions BL, DMB,
- DSB, ISB, MRS and MSR.
- Load/Store instructions takes 2cycles each
- Most of MOV, ADD, SUBTRACT, Compare, Logical, SHIFT instructions take 1 cycle
- Branches takes from 1 to 4 cycles ( depends if conditional, not or with link)
- “MULS” instruction provides a 32-bit x 32-bit multiply that yields the least significant 32-bits : 1 cycle !



# Cortex-M – firmware compatibility(1/2)

- Cortex M processors are FW and binary compatible
  - Migrating path M0->M3->M4 is straight forward
    - Instruction set of Cortex-Mx is strictly included in the instruction set of Cortex-My (for  $x < y$ ), allowing direct migration, while taking advantage of higher MCU clock speed and von Neumann to Harvard performance increase
  - Re-compilation of the code is recommended
    - from Cortex-M0 to Cortex-M3, in order to fully take advantage of the higher performance ISA (e.g. HW division)
    - From M0/M3 to M4 w/ FPU, in order to generate the FPU code
- For a given STM32 family, a full peripheral set compatibility is guaranteed in order to allow this simple migration path

# Cortex-M – firmware compatibility(2/2)

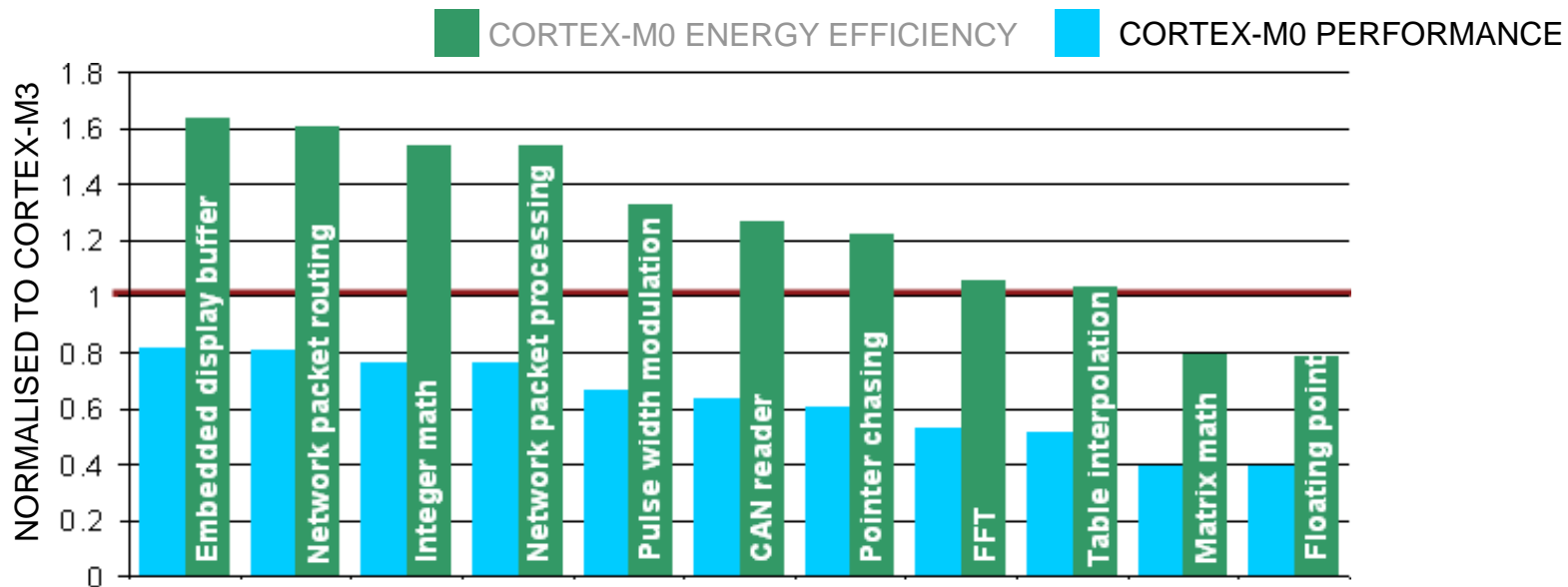
46

- Cortex M processors are FW and binary compatible
  - When moving from M0/M3 to M4, some part of the code might be re-coded using intrinsics, taking advantage of the advanced DSP/SIMD instructions
  - When moving backwards M3/M4 -> M0, the code needs to be recompiled in order to use only M0 instruction codes
- Code density is equivalent on the different Cortex-M implementations
  - Code size differences for usual codes are below few percents, provided that the same optimizations options are chosen in the compiler

# Cortex-M0 complements Cortex-M3

- Cortex-M3 is flagship MCU processor, offering higher performance
  - Delivers a performance efficiency advantage
- Cortex-M0 better suited for simpler control and communications
  - Delivers an energy efficiency and cost advantage

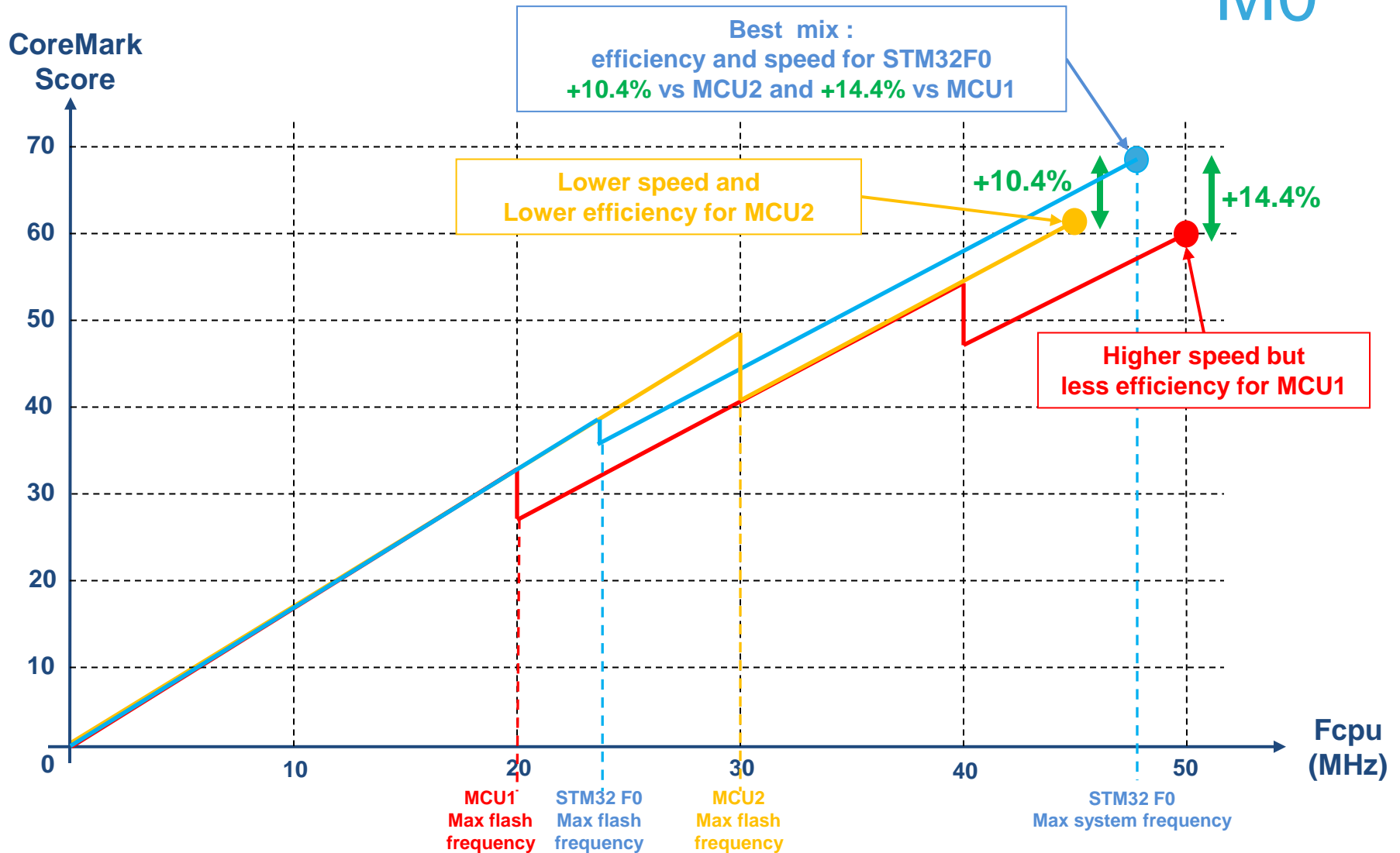
Note: Energy efficiency ONLY refers to CPU consumption, similar energy at system level



Normalised Cortex-M3 performance and energy efficiency vs. Cortex-M0 for algorithm types.

Energy efficiency based on Cortex-M3 and Cortex-M0 processor power estimates – ONLY refers to CPU consumption, similar energy at system level

# CoreMark score vs. Competitors on M0



\* Relative results based on ARM-MDK v4.50 compiler and in same conditions



# Simplified Register Set

- Very simple, linear 4GByte address space
- Very compiler friendly
- All registers are 32-bit wide
  - Registers R0 –R7 (Low registers)
  - Registers R8 –R12 (High registers)
- 3 registers with special usage
  - Stack Pointer (SP) –R13
  - Link Register (LR) –R14
  - Program Counter (PC) –R15
- Special-purpose register
  - xPSR : Application and Exception Program status bits and flags
- Vector Table contains Addresses



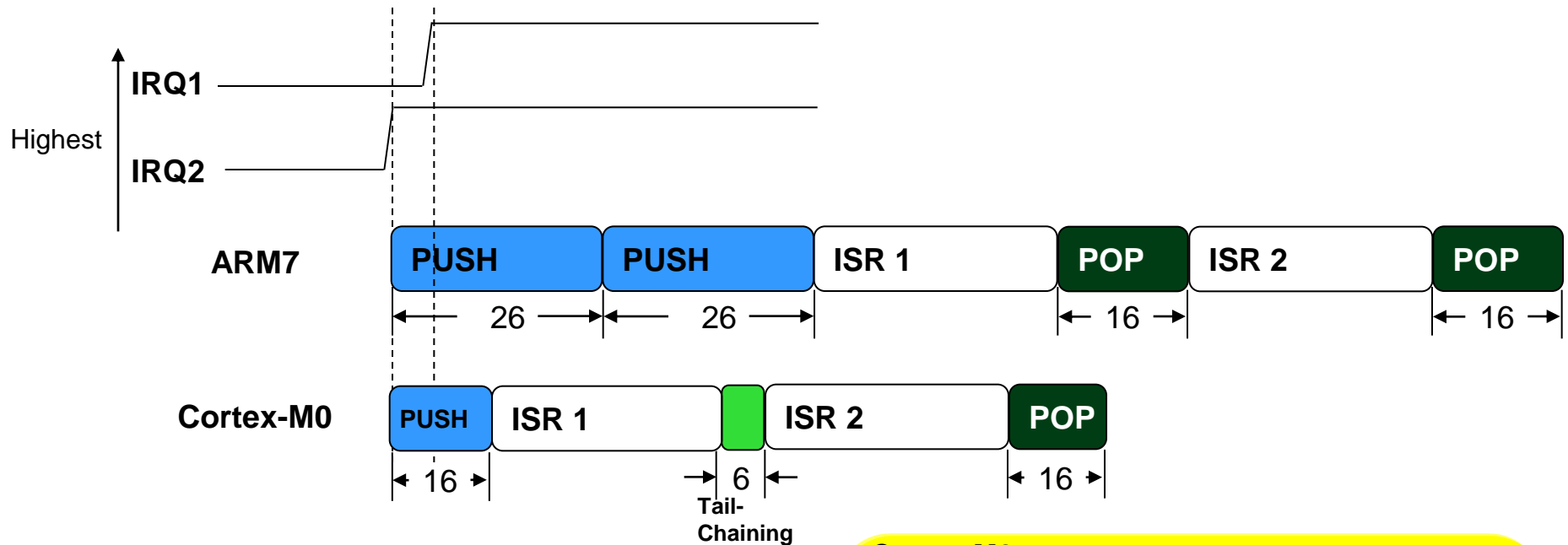
- **Very low latency interrupt processing**
  - Interruptible LDM/STM for low interrupt latency ( Load/store multiples)
  - Automatic processor state save and restore
    - Provides low latency ISR entry and exit
    - Allows handler to be written entirely in 'C'
- **The Cortex-M0 processor integrates an advanced Nested Vectored Interrupt Controller (NVIC)**
  - The NVIC supports up to 32 dynamically reprioritizable interrupts each with up to 4 levels of priority
  - Allows early processing of interrupts
  - Supports advanced features for next generation real-time applications
    - Tail-chaining of pending interrupts
    - Late-arrival interrupt handling and priority boosting / inversion

**Exceptional Control Capabilities Through Integrated Interrupt Handling**

# Interrupt Handling Method

- Interrupt handling is micro-coded. No instruction overhead
- Entry
  - Processor state automatically saved to the stack over the bus.
    - {PC, xPSR, R0-R3, R12, LR}
  - Then, ISR ready to start executing as soon as stack PUSH complete.
  - Late arriving interrupt will restart ISR fetching, but state saving does not need to be repeated.
- Exit
  - Processor state is automatically restored from the stack.
  - Then interrupted instruction is executed upon completion of stack POP.
  - Stack POP can be interrupted, allowing new ISR to be immediately executed without the overhead of state saving.

# Interrupt Response – Late Arriving



## 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-M0

- Stack push to ISR 2 is interrupted
- Stacking continues but new vector address is fetched in parallel
- Late-arrival to ISR1 entry will depend of the PUSH status, then 4 cycles will be necessary to read the vector table.
- Tail-chain into ISR 2

# Cortex-M0 Exception Types


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 handler not implemented
4-10	Reserved	N.A.	N.A.	
11	SVCall	Programmable	settable	System Service call
12-13	Reserved	N.A.	N.A.	
14	PendSV	Programmable	settable	Pendable request for System Device
15	SYSTICK	Programmable	settable	System Tick Timer
16	Interrupt #0	Programmable	settable	External Interrupt #0
.....	.....	.....	settable	.....
47	Interrupt#31	Programmable	settable	External Interrupt #31

- Vector Table starts at location 0
  - In the code section of the memory map
- Vector Table contains addresses (vectors) of exception handlers and ISRs
  - Not instructions
- Table size (in words) is = number of IRQ inputs + 16
  - Minimum size ( case of 1 IRQ) : 17 words
  - Maximum size ( case of 32 IRQs) 48 words
- Main stack pointer initial value in location 0
  - Set up by hardware during Reset
- Vector Table can not be relocated
  - The vector table is always seen at address 0x00000000,
  - Different Boot option or memory remap function, allow user to alias at this direction, SRAM, FLASH or System Memory

Address	Vector
<b>0x00</b>	<b>Initial Main SP</b>
<b>0x04</b>	<b>Reset</b>
<b>0x08</b>	<b>NMI</b>
<b>0x0C</b>	<b>Hard Fault</b>
<b>0x10-0x28</b>	<b>Reserved</b>
<b>0x2C</b>	<b>SVCall</b>
<b>0x30-0x34</b>	<b>Reserved</b>
<b>0x38</b>	<b>PendSV</b>
<b>0x3C</b>	<b>Systick</b>
<b>40</b>	<b>IRQ0</b>
<b>...</b>	<b>More IRQs</b>

# System Timer (SysTick)

- Flexible system timer
- 24-bit self-reloading down counter with end of count interrupt generation
- 2 configurable Clock sources (HCLK or HCLK/8)
- Suitable for Real Time OS or other scheduled tasks



Same as STM32F1xx  
just less channels

System Peripherals

# DIRECT MEMORY ACCESS (DMA)



- 5 independently configurable channels (requests)
- 4 levels of priority settable by software consisting of very high, high, medium, low
- Independent source and destination transfer size (byte, half word, word)
- Support for circular buffer management
- 3 event flags (DMA Half Transfer, DMA Transfer complete and DMA Transfer Error)
- Peripheral-to-memory, memory-to-peripheral, peripheral-to-peripheral & Memory-to-memory transfers
- Access to Flash, SRAM, APB and AHB peripherals as source and destination
- Programmable number of data to be transferred: up to 65536



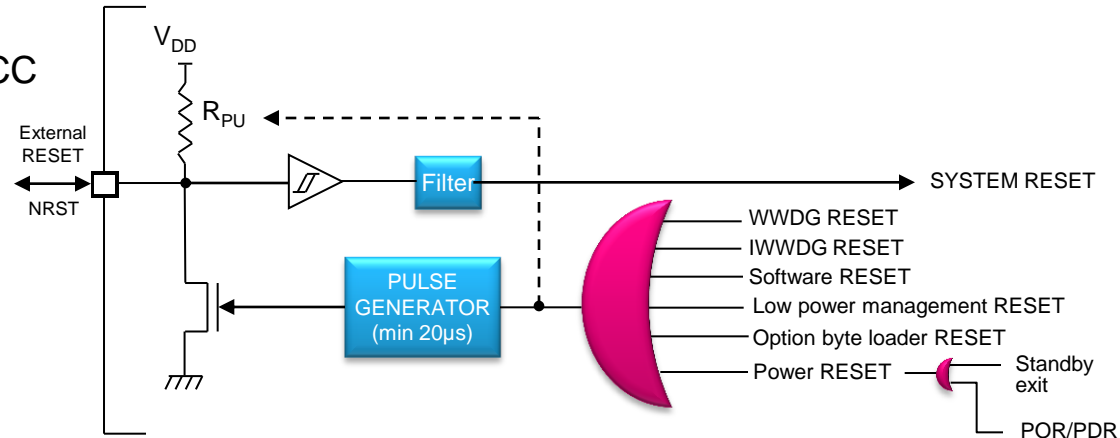
Same as STM32F1xx  
but with new features

System Peripherals

# RESET AND CLOCK CONTROL (RCC)

# RESET Sources

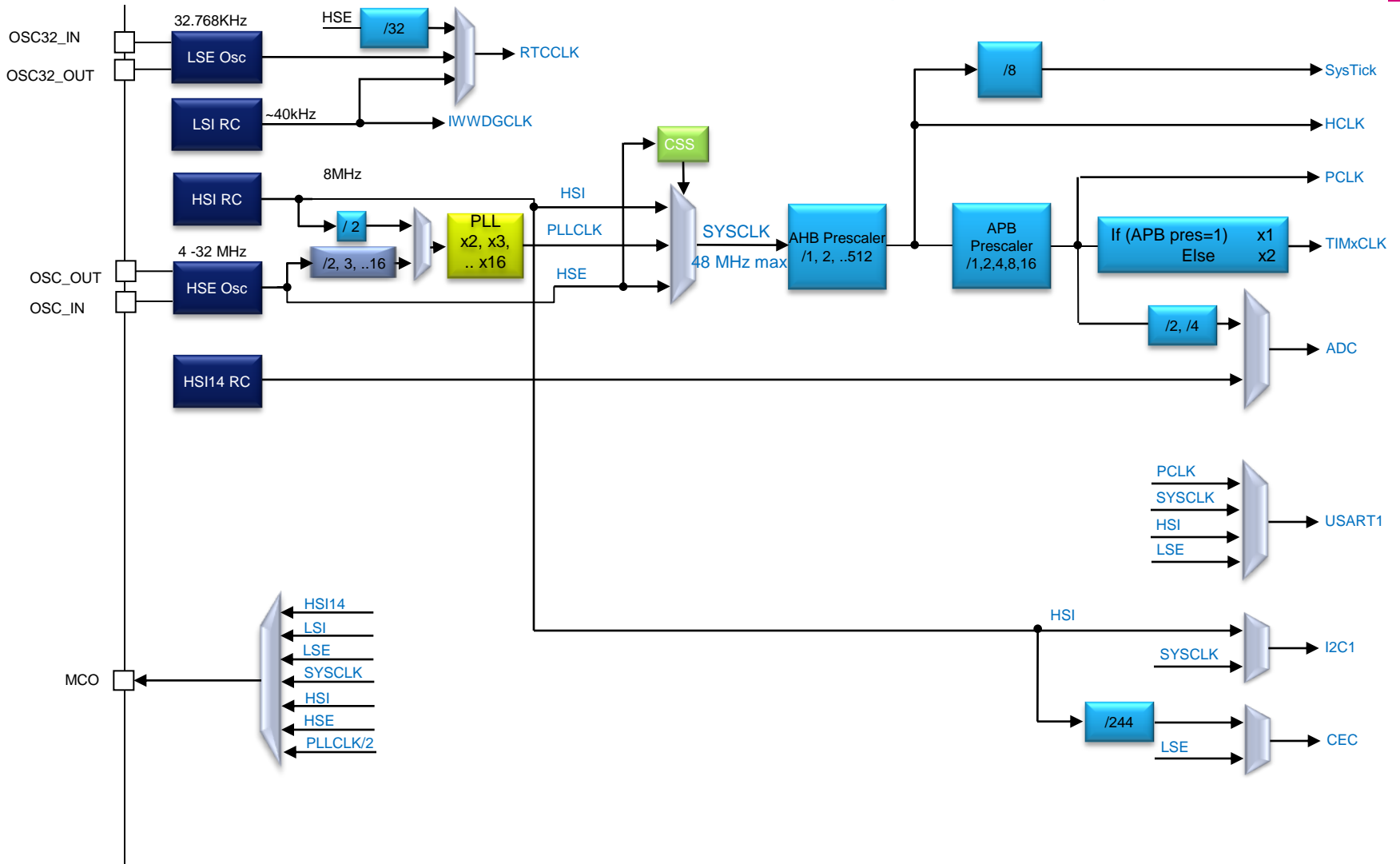
- System RESET
  - Resets all registers except some RCC registers and Backup domain
  - Sources
    - Low level on the NRST pin (External Reset)
    - WWDG end of count condition
    - IWWDG end of count condition
    - A software reset (through NVIC)
    - Low power management reset
    - Option byte loader reset (FORCE\_OBL bit)
- Power RESET
  - Resets all registers except the Backup domain
  - Sources
    - Power On/Power down Reset (POR/PDR)
    - Exit from STANDBY



- Backup domain RESET
  - Resets in the Backup domain: RTC registers + Backup Registers + RCC\_BDCR register
  - Sources
    - BDRST bit in RCC\_BDCR register
    - POWER Reset

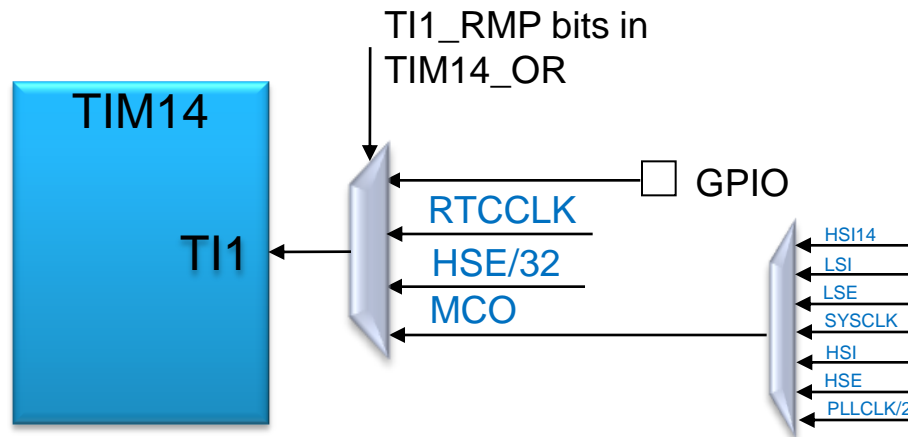
- System Clock (SYSCLK) sources:
  - **HSE** (High Speed External osc) 4MHz to 32MHz, can be bypassed by user clock
  - **HSI** (High Speed Internal RC): factory trimmed internal RC oscillator 8MHz +/- 1%
  - **PLL** x2, x3, .. x16 (16MHz min. output freq.)
- Additional clock sources:
  - **LSI** (Low Speed Internal RC): ~40kHz internal RC
  - **LSE** (Low Speed External oscillator): 32.768kHz, can be bypassed by user clock
    - configurable driving strength (power/robustness compromise)
  - **HSI14** (High Speed Internal RC 14MHz): dedicated oscillator for ADC
- Clock-out capability on the MCO (HSI14, LSI, LSE, SYSCLK, HSI, HSE, PLL/2)
- Clock Security System (CSS) to switch to backup clock in case of HSE clock failure
  - Enabled by SW w/ interrupt capability linked to Cortex NMI
- RTC Clock sources: LSE, LSI and HSE clock divided by 32
- USART, I2C & CEC have multiple possible clock sources

# Clock Scheme



# Internal/External clock measurement using TIM14

- TIM14 input capture can be triggered by an I/O or by RTCCLK, HSE/32 or MCO signal → Purpose:
  - Use the precise LSE clock to measure HSI frequency; HSI used as system clock, knowing the LSE frequency we can determine the HSI frequency (w/ the precision of the LSE)
  - Measure the LSI frequency (w/ the precision of the HSE or HSI) to fine tune IWWDG and/or RTC timing
  - Have rough indication of the external crystal frequency by comparing HSI (used as a system clock) and HSE/32





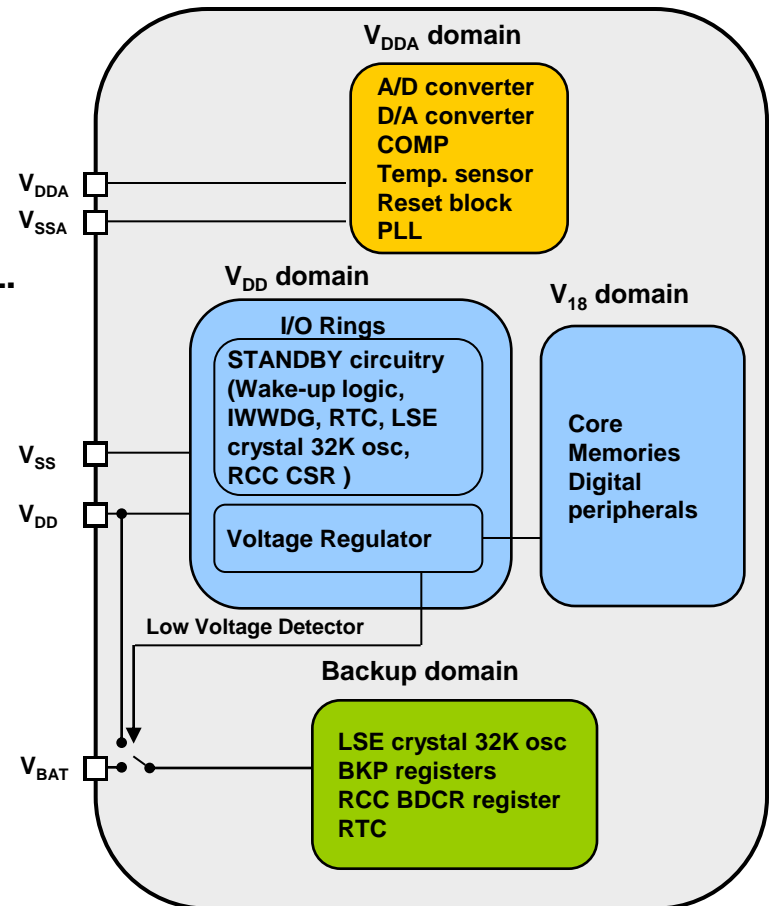
Same as STM32F1xx  
but with new features

System Peripherals

# POWER CONTROL (PWR)

## Power Supply Schemes

- **VDD = 2.0 to 3.6 V: External Power Supply for I/Os and the internal regulator.**
- **VDDA = 2.0 to 3.6 V: External Analog Power supplies for ADC,DAC, Reset blocks, RCs and PLL.**
  - ➔ ADC or DAC working only if VDDA  $\geq$  2.4 V
  - ➔ VDDA could be higher than VDD
    - Full analog performance at low consumption
- **VBAT = 1.65V to 3.6 V: For Backup domain when VDD is not present.**
- **Power pins connection:**
  - VDD and VDDA can be provided by a separated power supply source.
  - VSS and VSSA must be tight to ground



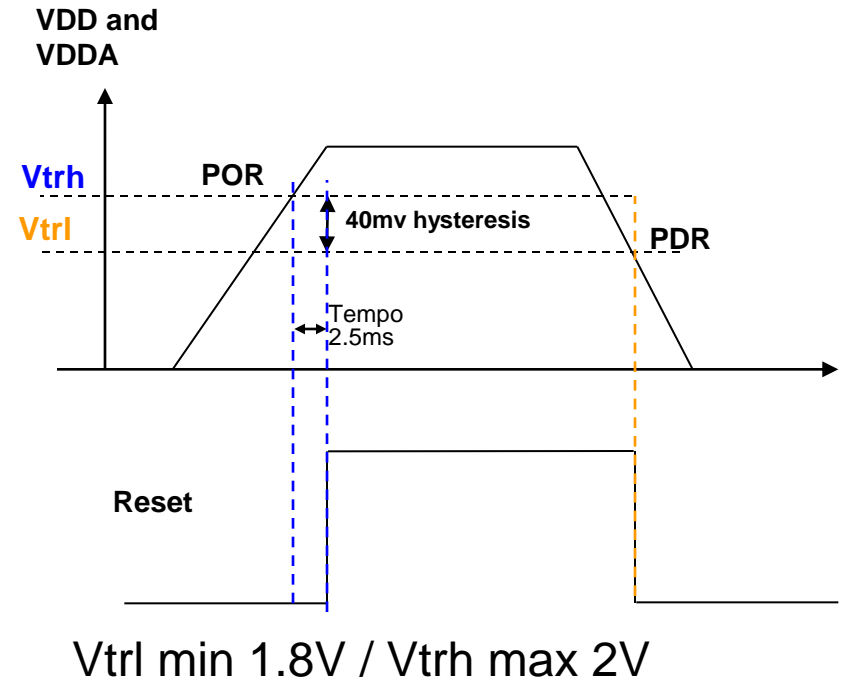


- When VDD power supply source is different from VDDA power supply source ( $VDD < VDDA$ )
  - The VDDA voltage **level must be always greater or equal** to the VDD voltage
  - During power-on, the VDDA must be provided first (before VDD)
  - During power-off, it is allowed to have temporarily  $VDD > VDDA$ , but the voltage difference must be  $<0.4V$ 
    - could be maintained by an external Schottky diode

# Power On Reset / Power Down Reset

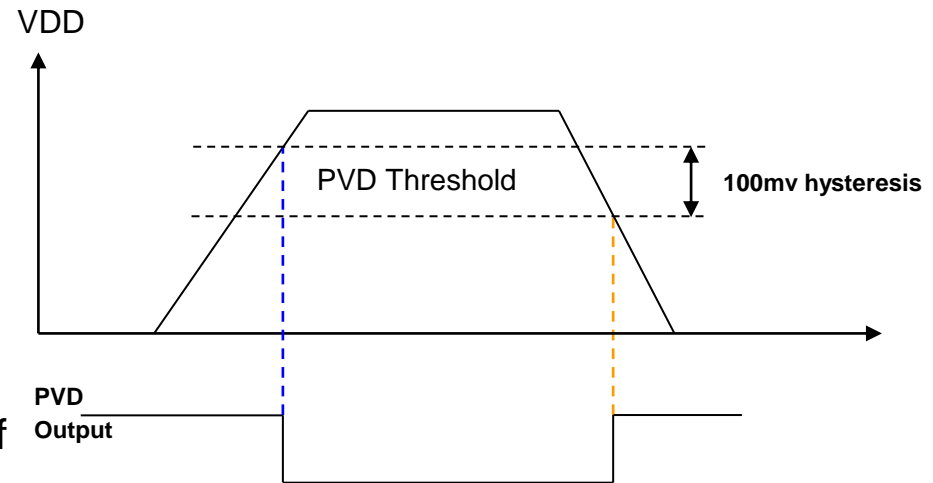
67

- Two Integrated POR / PDR circuitries 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

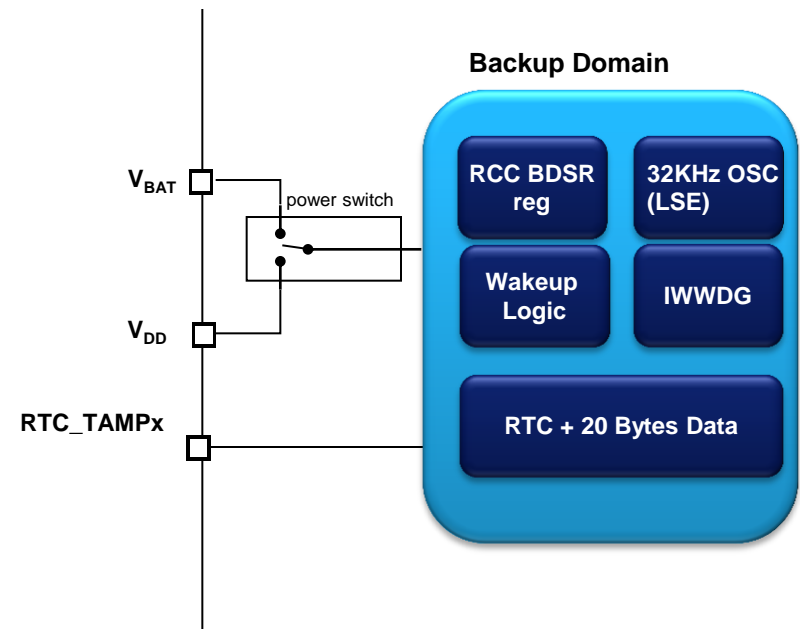


# Programmable Voltage Detector (PVD)

- Programmable Voltage Detector
  - Enabled by software
  - Monitor the VDD power supply by comparing it to a threshold
  - Threshold configurable from 2.1V to 2.9V by step of 90mV
  - Generate interrupt through EXTI Line16 (if enabled) when  $VDD < \text{Threshold}$  and/or  $VDD > \text{Threshold}$
  - → Can be used to generate a warning message and/or put the MCU into a safe state



- Backup Domain contains
  - Low power calendar RTC (Alarm, periodic wakeup from Stop/Standby)
  - 20 Bytes Data RTC registers
  - Separate 32kHz Osc (LSE) for RTC
  - RCC BCSR register: RTC clock source selection and enable + LSE config
- ➔ Reset only by RTC domain RESET
- VBAT independent voltage supply
  - Automatic switch-over to VBAT when VDD goes lower than PDR level
  - No current sunk on VBAT when VDD present
- Tamper event detection: resets all user backup registers
- TimeStamp event detection.



- 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 without context saving (6 cycles)
  - 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

# STM32F05x Low Power modes

Mode name	Entry	Wakeup	Effect on 1.8V domain clocks	Effect on VDD domain clocks	Voltage regulator	IO state	Wakeup latency
<b>SLEEP, SLEEP now or SLEEP on-exit</b>	WFI	Any interrupt	CPU CLK OFF no effect on other clocks or analog clock sources	None	ON	All I/O pins keep the same state as in the Run mode	None
	WFE	Wake-up event					
<b>STOP</b>	PDDS, LPSSDR bits + SLEEPDEEP bit + WFI or WFE	Any EXTI line (configured in the EXTI registers, <b>internal</b> and <b>external</b> lines)	All 1.8V domain clocks OFF	HSI and HSE and oscillators OFF	ON, in low power mode (depending on PWR_CR)	HSI RC wakeup time + regulator wakeup time from Low-power mode	
<b>STANDBY</b>	PDDS bit + SLEEPDEEP bit + WFI or WFE	WKUP pin rising edge, RTC alarm, RTC tamper event, external reset in NRST pin, IWDG reset			OFF		all I/O pins are high impedance (1)

**Note (1): Standby mode:** all I/O pins are high impedance except:

- Reset pad (still available)
- RTC pins, PC14 and PC15 if configured in the RTC registers.
- WKUP pin 1 (PA0) and WKUP pin 2(PC13), if enabled.

# Low Power Modes – typical consumption

- STM32F05x Low Power modes: uses CortexM0 Sleep modes
  - SLEEP, STOP and STANDBY

Feature	STM32F05x typ IDD/IDDA (*)
<b>RUN</b> mode w/ execute from <b>Flash</b> on <b>48MHz</b> (HSE bypass 8MHz x 6 PLL = 48MHz) All peripherals clock ON	<b>22.9 / 0.166 (mA)</b>
<b>RUN</b> mode w/ execute from <b>Flash</b> on <b>24MHz</b> (HSE bypass 8MHz x 3 PLL = 24MHz) All peripherals clock ON	<b>11.7 / 0.088 (mA)</b>
<b>RUN</b> mode w/ execute from <b>Flash</b> on <b>8MHz</b> (HSI) All peripherals clock ON	<b>4.15 / 0.079 (mA)</b>
<b>Sleep</b> mode w/ execute from <b>Flash</b> at <b>48MHz</b> (HSI 8MHz / 2 x 12 PLL = 48MHz) All peripherals clock ON	<b>12.9 / 0.243 (mA)</b>
<b>STOP</b> w/ Voltage Regulator in low power All oscillators OFF, PDR on VDDA is OFF	<b>3.6 / 1.34 (µA)</b>
<b>STANDBY</b> w/ LSI and IWWDG OFF PDR on VDDA is OFF	<b>1.1 / 1.21 (µA)</b>

Typical values are measured at TA = 25 °C, VDD =3.3V VDDA= 3.3 V.



**Same as STM32F1xx  
with new features**

System Peripherals

Flash memory

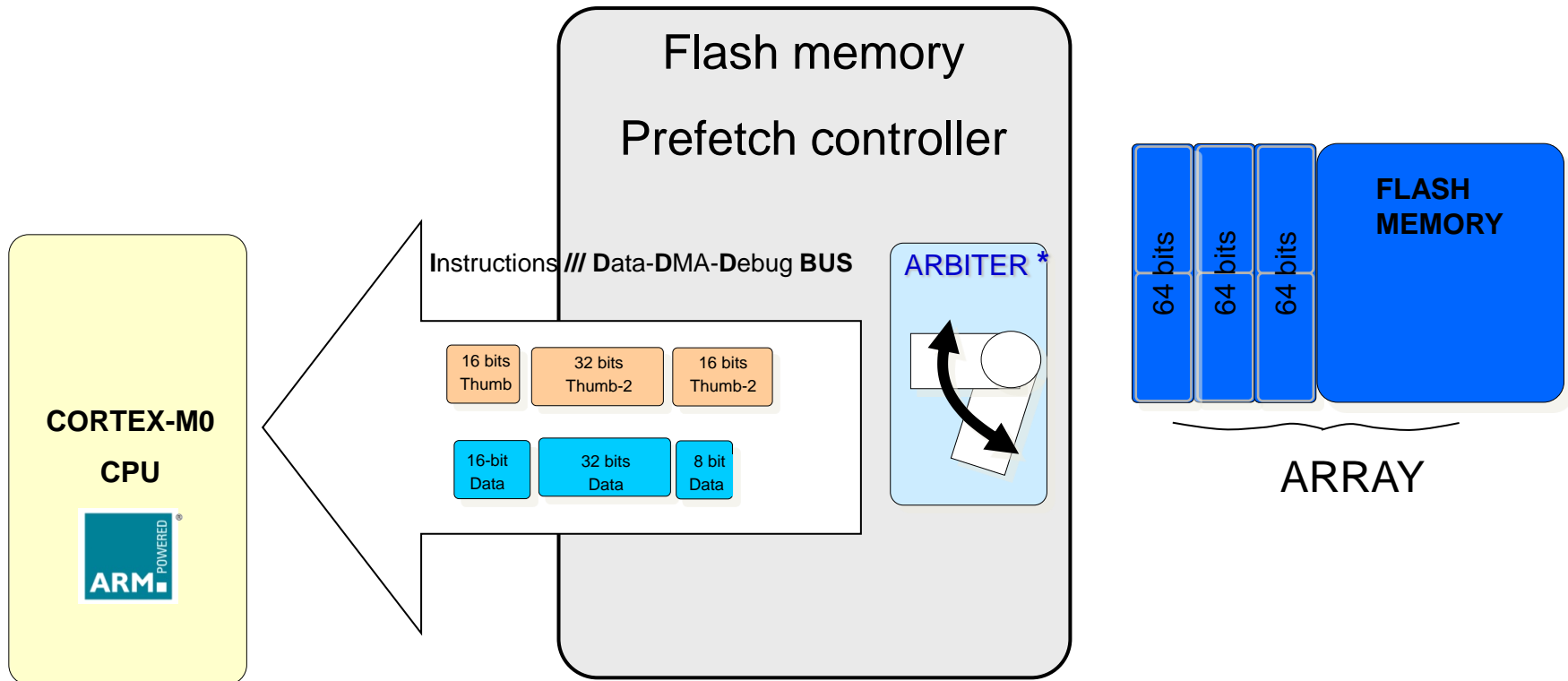


# Flash Features Overview

- Flash general features:
  - Up to 64KBytes
  - 64 pages of 1KBytes size and 16 Sectors of 4KBytes size (4 pages)
  - Endurance: 10k cycles
  - Access time: 35ns
  - Half word (16-bit) program time: 52.5µs (Typ)
  - Page erase time and Mass erase time: 20ms (Min), 40ms (Max)
- Flash interface features:
  - Read Interface with pre-fetch buffer
  - Option Bytes loader
  - Flash program/erase operations
  - Types of Protection:
    - Readout Protection: Level 0, Level 1 and Level 2 (No debug)
    - Write Protection
- Memory organization:
  - Main Program memory block (or Main Flash memory)
  - Information block : 3KBytes of System memory + 6 Option Bytes (12 with complements)
    - 2 option bytes for write protection and 1 for Readout protection
    - 1 option byte for device configuration : **VDDA supervisor, BOOT1**, Reset w/ STDBY/STOP, IWWG HW
    - 2 option bytes reserved for user data

# Flash memory prefetch controller

- Mission: Support 48 MHz operation directly from Flash memory
- Flash with Prefetch based on  $3 \times 64$ bits buffers

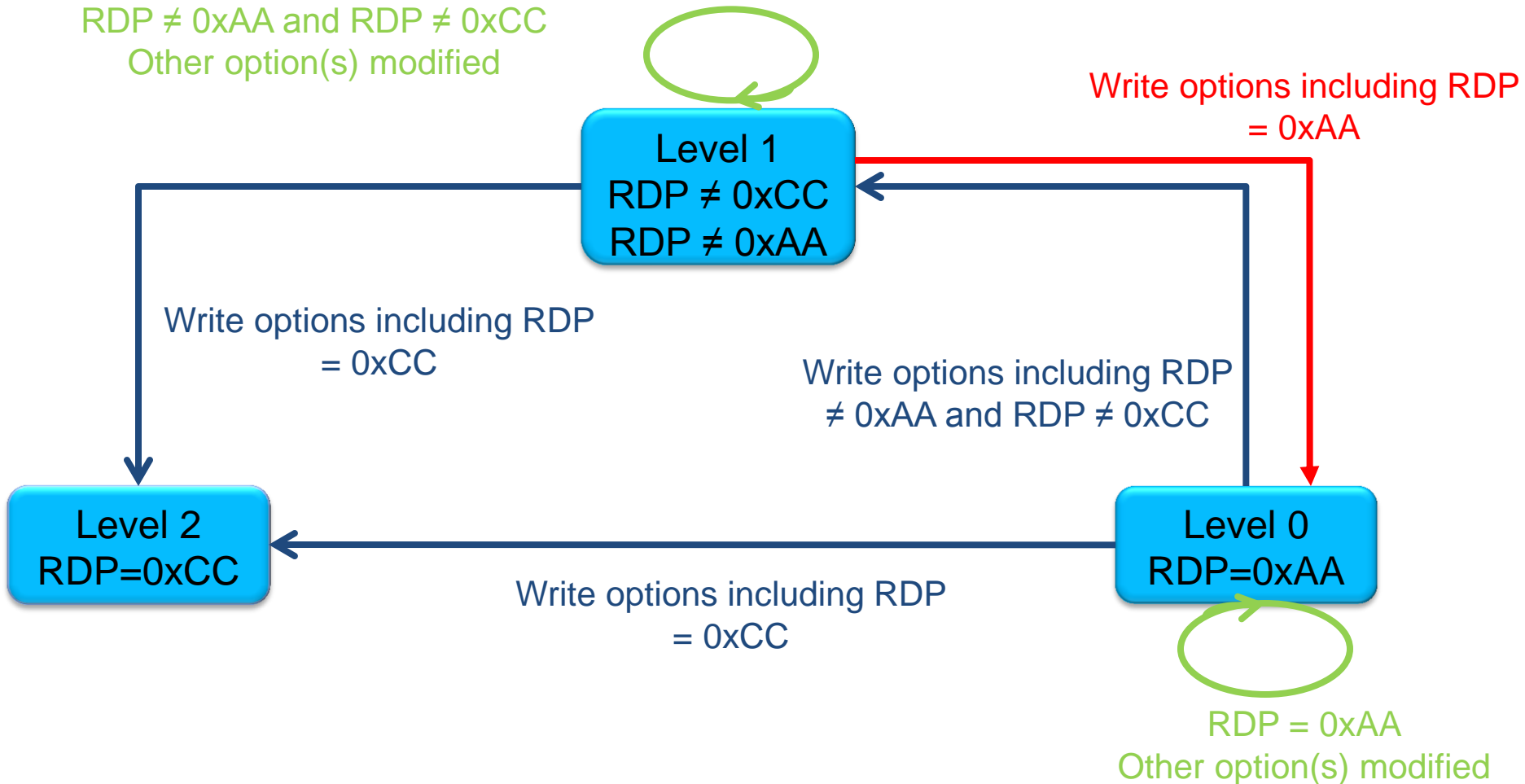


\* The read requests are managed with following priority order :

1=Option byte loader, 2=Prog and erase operation, 3=Instr/Data fetch, 4=prefetch access

# Flash Protections (1/2)

RDP ≠ 0xAA and RDP ≠ 0xCC  
Other option(s) modified



Option byte write (RDP level increase) includes: Option byte erase and New option byte programming

Option byte write (RDP level decrease) includes: Option byte erase, New option byte programming and Mass Erase

Option byte write (RDP level identical) includes : Option byte erase and New option byte programming

# Flash Protections (2/2)

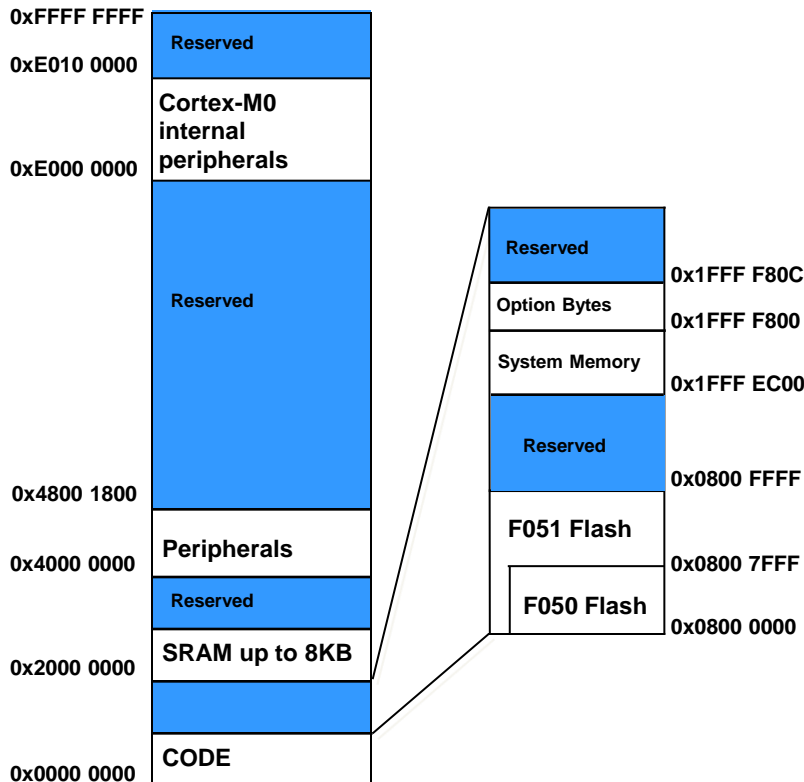
- Access status versus protection level and execution modes :

Area	Protection level	User execution			Debug, boot from RAM or boot from system memory (loader)		
		Read	Write	Erase	Read	Write	Erase
Main memory	1	Yes	Yes	Yes	No	No	No
	2	Yes	Yes	Yes	N/A	N/A	N/A
System memory	1	Yes	No	No	Yes	No	No
	2	Yes	No	No	N/A	N/A	N/A
Option bytes	1	Yes	Yes	Yes	Yes	Yes	Yes
	2	Yes	Yes	No	N/A	N/A	N/A
Backup registers	1	Yes	Yes	N/A	No	No	N/A
	2	Yes	Yes	N/A	N/A	N/A	N/A

# Memory Mapping and Boot Modes


- Addressable memory space of 4 Gbytes
  - FLASH : up to 64 Kbytes
  - RAM : up to 8 Kbytes **with parity check**
    - 4 bits per word for parity check
    - Automatic check when reading
    - NMI/BRK\_IN of TIM1 w/ SRAM\_PARITY\_LOCK bit
- + SRAM\_PEF Error Flag both in SYSCFG register 2

- **Boot modes**  
Depending on the Boot configuration, Embedded Flash memory, System memory or Embedded SRAM memory is aliased at @0x00. Even when aliased, these memories are still accessible from their original memory space.
- The boot configuration is defined with **BOOT0 pin** and **nBOOT1 bit** from **USER option byte**.



BOOT Mode Selection		Boot Mode	Aliasing
nBOOT1	BOOT0		
x	0	User Flash	User Flash is selected as boot space
1	1	System memory	SystemMemory is selected as boot space
0	1	Embedded SRAM	Embedded SRAM is selected as boot space

- **System memory** : contains the Bootloader used to re-program the FLASH through USART1 (same pins configuration as for STM32 products).
- **Boot from SRAM** : The relocation of the vector table is not allowed on the Cortex M0. When Booting from SRAM, System memory or User Flash, the vector table will be aliased at direction 0x00000000



Same as STM32L and  
STM32F2xx with  
some changes

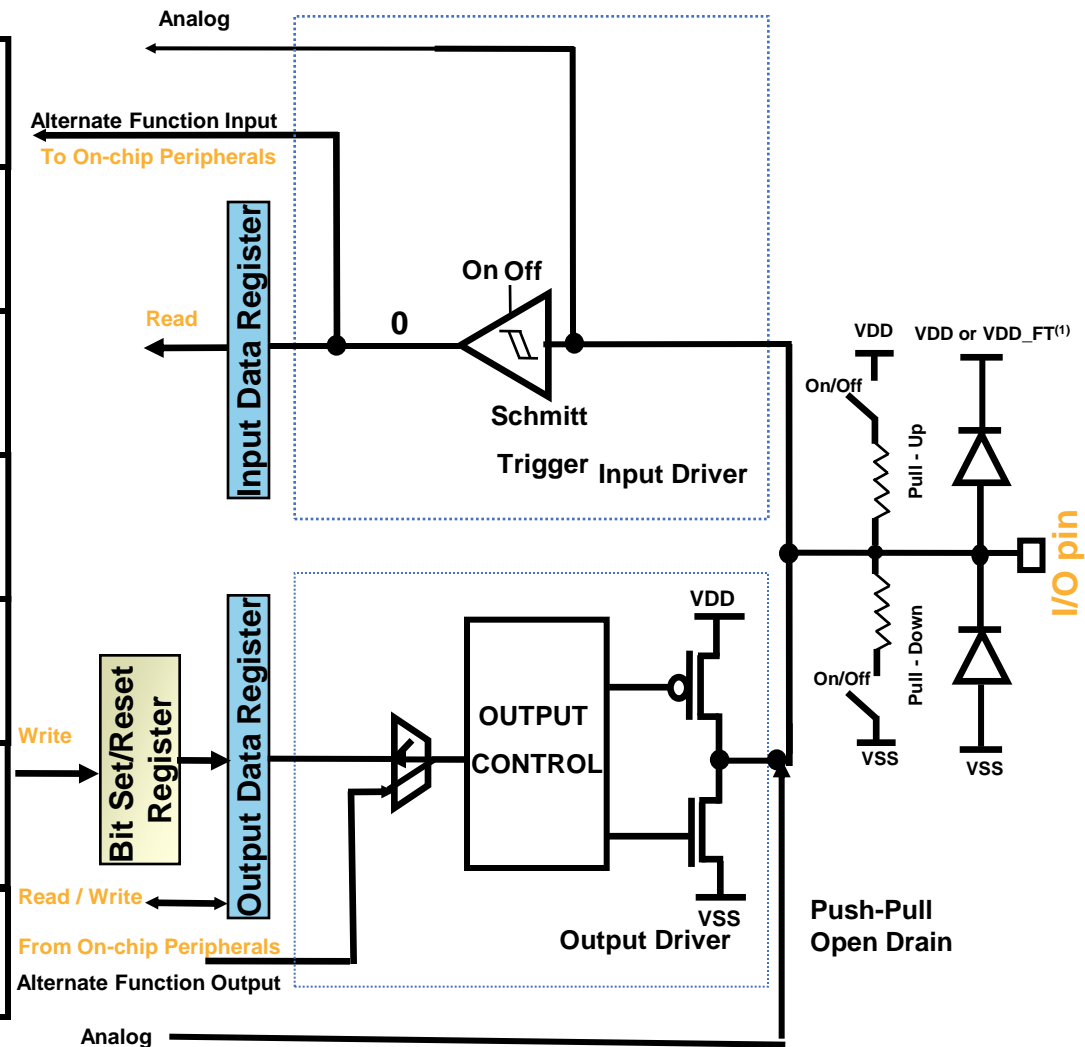
System Peripherals

# GENERAL-PURPOSE I/Os (GPIO)

- Up to 55 multifunction bi-directional I/O ports available on biggest package 64 pin. ( 86% ratio versus 80% STM32F1 series)
- Almost all standard I/Os are 5V tolerant (except ADC pins)
- All Standard I/Os are shared in 5 ports: GPIOA [0..15], GPIOB[0..15], GPIOC[0..15], GPIOD[2], GPIOF[0,1,4..7]
- Atomic Bit Set and Bit Reset using BSRR and BRR registers
- GPIO connected to AHB bus: max toggling frequency 12 MHz
- Configurable Output slew rate speed up to 50MHz
- Locking mechanism (GPIOx\_LCKR) provided to freeze the I/O configuration on ports A and B
- Up to 55 GPIOs can be set-up as external interrupt (up to 16 lines at time) able to wake-up the MCU from low power modes

# GPIO Configuration Modes

MODER(i) [1:0]	OTYPER(i) [1:0]	PUPDR(i) [1:0]	I/O configuration
01	0	0 0	Output Push Pull
		0 1	Output Push Pull with Pull-up
	1	0 0	Output Open Drain
		0 1	Output Open Drain with Pull-up
10	0	0 0	Alternate Function Push Pull
		0 1	Alternate Function PP Pull-up
		1 0	Alternate Function PP Pull-down
	1	0 0	Alternate Function Open Drain
		0 1	Alternate Function OD Pull-up
		1 0	Alternate Function OD Pull-down
00	x	0 0	Input floating
		0 1	Input with Pull-up
		1 0	Input with Pull-down
11	x	x	Analog mode



\* In output mode, the I/O speed is configurable through OSPEEDR register: 2MHz, 10MHz or 50MHz

(1) VDD\_FT is a potential specific to five-volt tolerant I/Os and different from VDD.





New feature on  
STM32F0xx

Analog Peripherals

# Touch Sensing controller (TSC)

Not available on STM32F050x products

- Proven and robust **surface charge transfer acquisition principle**
- **One sampling capacitor for up to 3 capacitive sensing channels** to reduce the system components
- Supports up to **18 capacitive sensing channels** split over 6 analog I/O groups
- Up to **6 capacitive sensing channels can be acquired in parallel** offering a very good response time
  - 1 counter per analog I/O group to store the current acquisition result
- **Full hardware management** of the charge transfer acquisition sequence
  - No CPU load during acquisition
- **Spread spectrum** feature to improve system robustness in noisy environments (minimum step of 20.8ns)

# TSC Features (2/2)

- Programmable charge transfer frequency (up to 6.8 MHz)
- Programmable sampling capacitor I/O pin and channel I/O pins
  - Any GPIO of an analog IO group can be used for the sampling capacitor
  - Any GPIO of an analog IO group can be used for the channel
- Programmable max count value to avoid long acquisition when a channel is faulty
- Dedicated **end of acquisition and max count error flags with interrupt capability**
- Compatible with **proximity, touchkey, linear and rotary touch sensor** implementation
- Designed to operate with **STMTouch touch sensing firmware library**

# STMTouch Touch Sensing Library

- Complete free C source code library with firmware examples
- Multifunction capability to combine capacitive sensing functions with traditional MCU features
- Enhanced processing features for optimized sensitivity and immunity
  - Calibration, environment control system (ECS), debounce filtering , detection exclusion system (DxS), etc
- Complete and simple API for status reporting and application configuration
- Touchkey, proximity, linear and rotary touch sensors support
- Compliance with MISRA and with all STM32 C compilers
- **STM32F051 support planned for end Q2 2012**



Similar set as STM32F100x  
functionality like STM32F2x

Standard Peripherals  
**TIMERS (TIM)**

# Timers overview

Timer	Width & Direction	CAPCOM + COMPL.	Prescaler & Max clock	DMA features	Synch. Module	External trigger	Break	Repetition counter
TIM1	16bit UP/DOWN	4 + 3	Linear 16-bit 48 MHz	All	Yes	Yes	Yes	Yes
TIM2	32bit UP/DOWN	4 + 0		All	Yes	Yes	No	No
TIM3	16bit UP/DOWN	4 + 0		All	Yes	Yes	No	No
<b>TIM6</b>	16bit UP only	0		1 DMA req.	Yes	No	No	No
TIM14	16bit UP only	1 + 0		No	No	Yes LSE	No	No
<b>TIM15</b>	16bit UP only	2 + 1		All	Yes	No	Yes	Yes
TIM16	16bit UP only	1 + 1		All	No	No	Yes	Yes
TIM17	16bit UP only	1 + 1		All	No	No	Yes	Yes



New IP coming  
from STM32F-2

Communication Peripherals

# SERIAL PERIPHERAL INTERFACE (SPI/I2S)

Only SPI1 available on STM32F050x products

# SPI Features (1/2)

- Full duplex synchronous transfers (3 lines)
- Half duplex/Simplex synchronous transfers (2 lines, bi-directional data line at half duplex)
- Programmable clock polarity & phase, data MSB/LSB first
- Master/multi Master/Slave operation
- Dynamic software/hardware NSS management (Master/Slave)
- Hardware CRC feature (8-bit & 16-bit data frames checking)
- Flags with IT capability (TxE, RxNE, MODF, OVR, CRCERR)
- Programmable bit rate: **up to  $f_{PCLK}/2$**
- BSY flag (ongoing communication check)
- DMA capability (separated Rx/Tx channels, automatic CRC & Tx/Rx access/threshold handling)







- **New enhanced NSS control:**
  - NSS pulse mode (NSSP)
  - TI mode
- **Programmable data frame from 4-bit to 16-bit**
- **Two 32-bit Tx/Rx FIFO buffers with DMA capability**
- **Data packed mode control**
- **(I2S mode)**

## When data packed mode is used

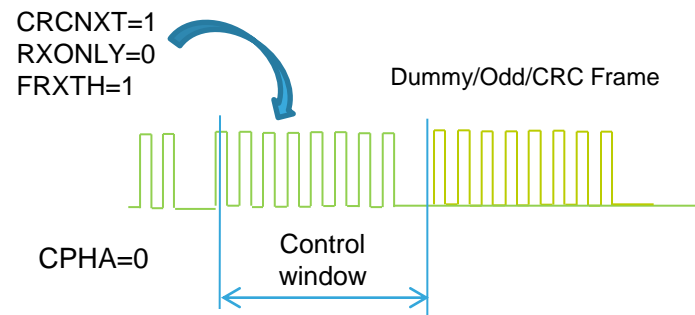
- Keep Rx threshold & read access of Rx FIFO always in line (8-bit/16-bit)
- Change Rx threshold just before last odd data frame is received

## When go to Halt or when disable the SPI

- check read FIFO occupancy and bus activity ( $FxLVL[1:0] = 00$  &  $BSY = 0$ )

## When communication is continuous (e.g. master Rx-only)

- Perform Rx threshold, change/CRC control or Stop at “Control window”



## When CRC is enabled at slave

- CRC clock input is sensitive for SCK signal even in case SPIE=0



Communication Peripherals

# INTER-INTEGRATED CIRCUIT INTERFACE (I2C)

Only I2C1 available on STM32F050x products

# I2C Features (1/2)

- I2C specification rev03 compatibility
- SMBus 2.0 HW support
- PMBus 1.1 Compatibility
- Multi Master and slave capability
- Controls all I<sup>2</sup>C bus specific sequencing, protocol, arbitration and timing
- Standard, fast and fast mode + I<sup>2</sup>C mode (up to 1MHz)
- 20mA output drive capability for FM+ mode

# I2C Features (2/2)

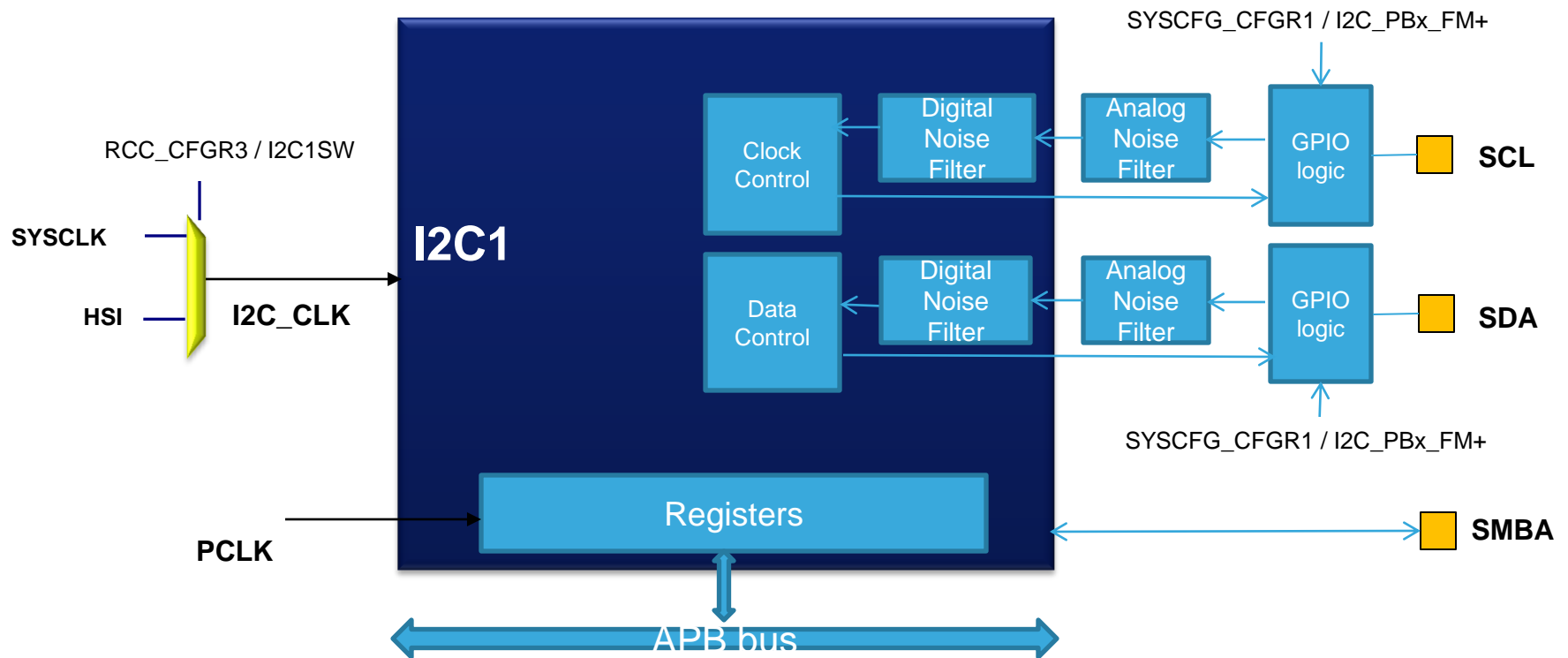
- 7-bit and 10-bit addressing modes
- Multiple 7-bit Addressing Capability with configurable mask
- Programmable setup and hold time
- Easy to use event management
- Programmable analog and digital noise filter
- Wakeup from STOP mode on address match
- Optional clock stretching
- Independent clock
- 1-byte buffer with DMA capability

# I2C Implementation

- I2C1 supports all features
- I2C2 supports a smaller set of features

I2C features	I2C1	I2C2
Independent clock	YES	NO
SMBUS HW support	YES	NO
Wakeup from STOP	YES	NO
20mA output drive	YES	NO

# I2C1 Block Diagram



# I2C SDA and SCL noise filter

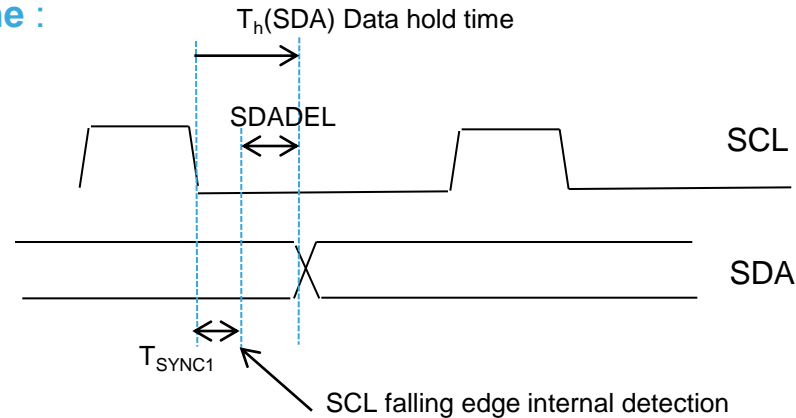
- Analog noise filter in SDA and SCL I/O
  - Can filter spikes with a length up to 50ns
  - This filter can be enabled or disabled by SW (enabled by default)
- Digital noise filter for SDA and SCL
  - Suppress spikes with a programmable length from **0 to 15 I2C\_CLK periods.**
- Only analog filter can be enabled when Wakeup from STOP feature is enable.
- Filters configuration must be programmed when the I2C is disable.



# I2C Programmable timings

- Setup and Hold timings between SDA and SCL in transmission are programmable by SW with PRESC, SDADEL and SCLDEL fields in I2C Timing Register (I2Cx\_TIMINGR).
  - SDADEL is used to generate Data Hold time.  $T_{SDA\Delta EL} = SDADEL * (PRESC+1) * T_{I2C\_CLK}$
  - SCLDEL is used to generate Data Setup time.  $T_{SCL\Delta EL} = (SCLDEL+1) * (PRESC+1) * T_{I2C\_CLK}$

## Example Data Hold Time :



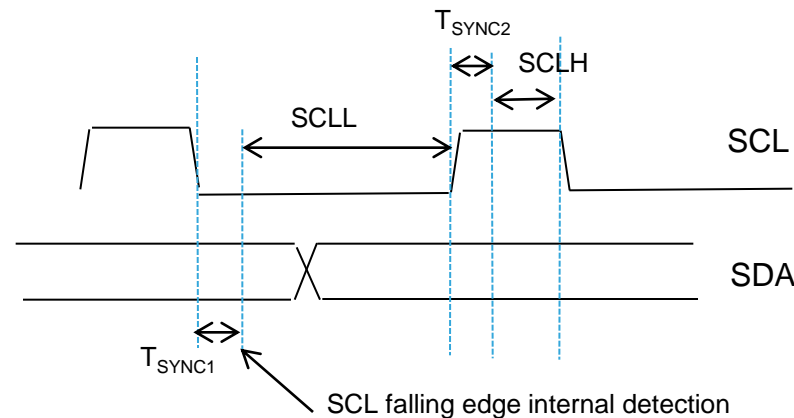
- The Setup and Hold configuration must be programmed when the I2C is disable.
- I2C\_Timing\_Config\_Tool is available in FWLib to calculate I2C\_TIMINGR value for your application.

# I2C Master clock generation

- SCL Low and High duration are programmable by SW with PRESC, SCLL and SCLH fields in I2C Timing Register (I2Cx\_TIMINGR).
  - SCL Low counter is  $(SCLL+1) * (PRESC+1) * T_{I2C\_CLK}$ . It starts counting after SCL falling edge internal detection. After counting, SCL is released.
  - SCL High counter is  $(SCLH+1) * (PRESC+1) * T_{I2C\_CLK}$ . It starts counting after SCL rising edge internal detection. After counting SCL is driven low.
- The total SCL period is :

$$T_{SYNC1} + T_{SYNC2} + [(SCLL+1) + (SCLH+1)] * (PRESC+1) * T_{I2C\_CLK}$$

## SCL Period:



- The SCLL and SCLH configuration must be programmed when the I2C is disable.
- I2C\_Timing\_Config\_Tool is available in FWLib to calculate I2C\_TIMINGR value for your application.

# Slave Addressing Mode

- I2C can acknowledge several slave addresses. 2 address registers :
  - I2Cx\_OAR1 : 7-bit or 10-bit mode.
  - I2Cx\_OAR2 : 7-bit mode only. OA2MSK[2:0] allow to mask from 0 to 7  
LSB of OAR2 :

OA2MSK[2:0]	Address match condition
000	address[7:1] = OA1[7:1]
001	address[7:2] = OA1[7:2] (Bit 1 is don't care)
010	address[7:3] = OA1[7:3] (Bit 2:1 are don't care)
...	
111	All addresses are acknowledged except I2C reserved addresses.

# Wakeup from STOP on address match

101

- When I2C\_CLK clock is HSI, the I2C is able to wakeup MCU from STOP when it receives its slave address. All addressing mode are supported.
  - During STOP mode and no address reception : HSI is switched off.
  - On START detection, I2C enables HSI, used for address reception.
- Wakeup from STOP is enabled by setting WUPEN in I2C1\_CR1.
- Clock stretching must be enabled to ensure proper operation: NOSTRETCH=0.

Interrupt event	Interrupt flag
Receive Buffer Not Empty	RXNE
Transmit buffer Interrupt Status	TXIS
Stop detection interrupt flag	STOPF
Transfer Complete Reload	TCR
Transfer Complete	TC
Address matched	ADDR
NACK reception	NACKF

# Easy Master mode management

- For payload  $\leq 255$  bytes : **only 1 write action needed !!** (apart data rd/wr)

I2Cx\_CR2 is written w/ :

- START=1
- SADD : slave address
- RD\_WRN : transfer direction
- NBYTES = N : number of bytes to be transferred
- **AUTOEND =1** : STOP automatically sent after N data.

## AUTOEND

0 : Software end mode	End of transfer SW control after NBYTES data transfer : <ul style="list-style-type: none"><li>• TC flag is set. Interrupt if TCIE=1.</li><li>• TC is cleared when START or STOP is set by SW<ul style="list-style-type: none"><li>➤ If START=1 : RESTART condition is sent</li></ul></li></ul>
1 : Automatic end mode	STOP condition sent after NBYTES data transfer

- Data transfer managed by Interrupts (TXIS / RXNE) or DMA

# Easy to use event management

- For payload > 255 : in addition, **RELOAD** must be set in I2Cx\_CR2.

RELOAD	
0 : No reload	NBYTES data transfer is followed by STOP or ReSTART
1 : Reload mode	NBYTES is reloaded after NBYTES data transfer (data transfer will continue) : <ul style="list-style-type: none"><li>• TCR flag is set. Interrupt if TCIE=1.</li><li>• TCR is cleared when I2Cx_CR2 is written w/ NBYTES≠0</li></ul>

- AUTOEND = 0 has no effect when RELOAD is set.

- By default : I2C slave uses clock stretching .
  - This can be disabled by setting **NOSTRETCH=1**
- Reception : Acknowledge control can be done on selected bytes in **Slave Byte Control (SBC)** mode with **RELOAD=1**
  - **SBC = 1** enables the **NBYTES** counter in slave mode (Tx and Rx modes).
  - **SBC = 1** is allowed only when **NOSTRETCH=0**.

## SBC

0 : Slave Byte Control disable

All received bytes are acknowledged.

1 : Slave Byte Control enable

If **RELOAD=1**, after **NBYTES** data are transferred :

- **TCR** set & **SCL** stretched before **ACK** pulse in reception.
- **TCR** is cleared when **I2Cx\_CR2** is written w/ **NBYTES≠0**
  - if **I2Cx\_CR2/NACK = 1**: received byte is **NOT** Acknowledged





Same as STM32F1xx  
but with new features

System Peripherals

# CRC CALCULATION UNIT (CRC)

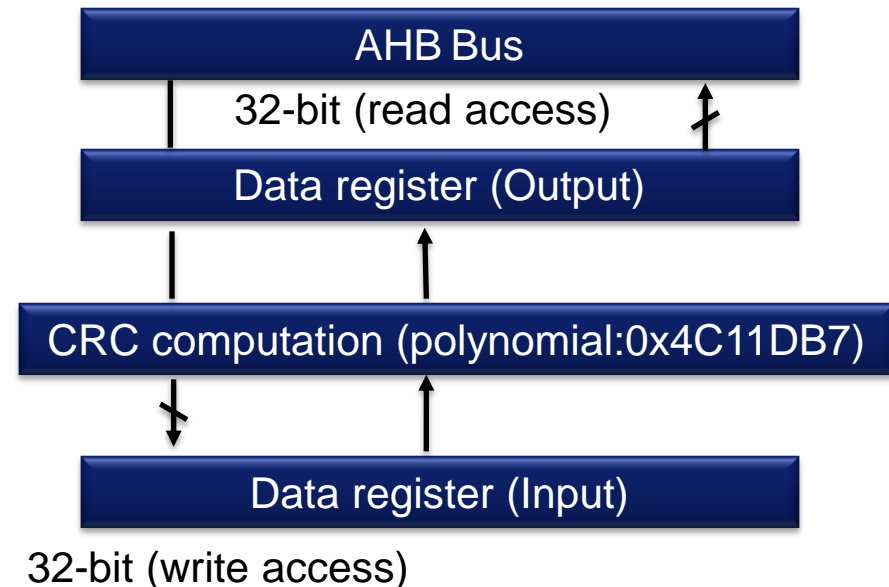
# CRC Features (1/3)

107

- CRC-based techniques are used to verify storage integrity or data transmission
- In functional safety standards (such as EN/IEC 60335-1), CRC peripheral offers a means of verifying the embedded Flash memory integrity
- Uses CRC-32 (Ethernet) polynomial: 0x4C11DB7 :

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

- Single input/output 32-bit data register, but handles 8, 16 and 32-bits data size
- CRC Peripheral is mapped to AHB bus for fast operation, CRC computation done in 4 AHB clock cycles (HCLK) maximum
- Computation duration depends on data size
  - 4 HCLK cycles for 32-bit
  - 2 HCLK cycles for 16-bit
  - 1 HCLK cycles for 8-bit

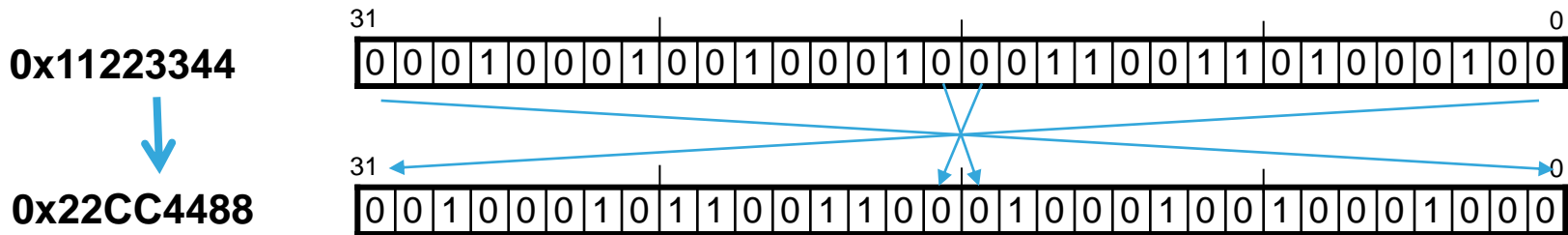


- The Input register can be accessed for write operations either :
  - by Cortex-M0 CPU or
  - In back-to-back with DMA in Memory to Memory transfer mode, keeping CPU free for other tasks or even in sleep mode ( for power optimization)
- A **new Input buffer** is available to avoid AHB bus stall during computation time, thus freeing AHB1 bus for others concurrent operations
- It has also a general-purpose 8-bit register (can be used for temporary storage)
- **New Modes**
  - Programmable CRC initial value, very useful when the CRC computation is stopped then continued without re-start again in the application ( Interrupts with higher priorities, several CRC computations etc.)
  - Initial value is re-loaded from CRC\_INIT (32-bits register) each time a CRC reset is applied, the default value is 0xFFFF\_FFFF to keep compatibility with STM32F1xx series

- **New Modes (Continued)**

- **Reversibility on Input and Output data** without extra cycles

- **Output:** the reversing is done at bit level only



- This replaces the same operation as “RBIT” instruction not available in Cortex-M0 instruction set but is in Cortex-M3/M4 and will save many CPU cycles if done by software...
- **Input :** The bits reversing can be performed on 8 bits, 16 bits and 32 bits to support various endianness schemes
  - example if input data 0x1A2B3C4D is used for CRC calculation as:
    - 0x58D43CB2 with bit-reversal done by byte
    - 0xD458B23C with bit-reversal done by half-word
    - 0xB23CD458 with bit-reversal done on the full word



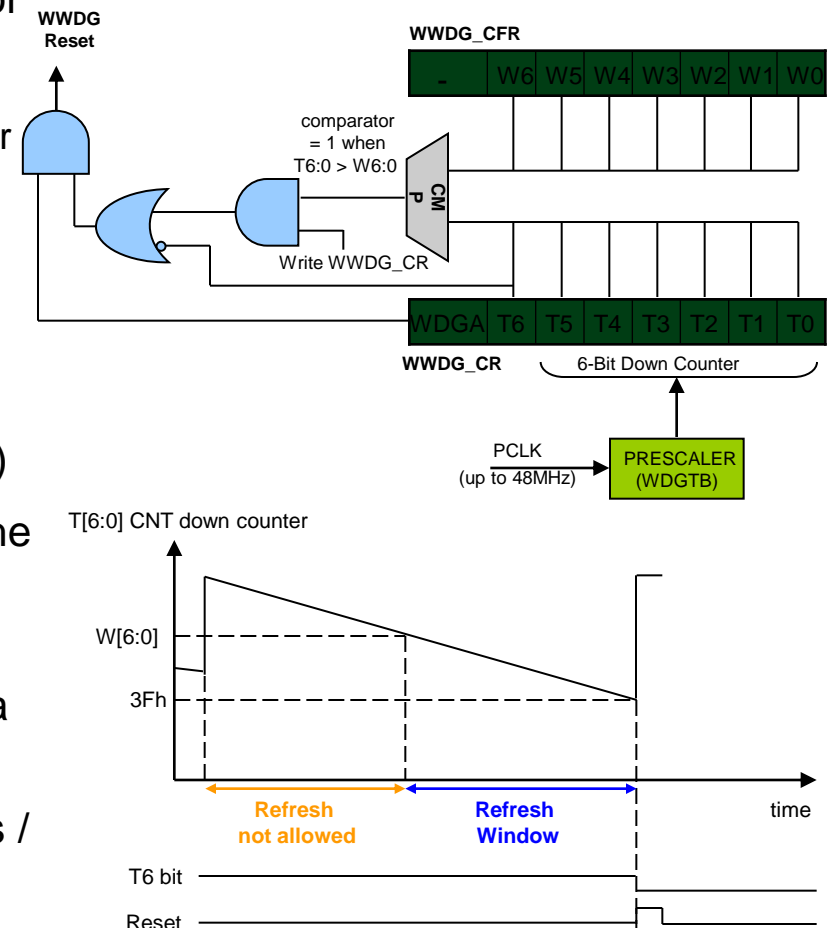
Same as STM32F1xx

System Peripherals

# System Window Watchdog (WWDG)

- Configurable time-window, can be programmed to detect abnormally late or early application behavior
- Conditional reset
  - Reset (if watchdog activated) when the down counter value becomes less than 40h (T6=0)
  - Reset (if watchdog activated) if the down counter is reloaded outside the time-window
- To prevent WWDG reset: write T[6:0] bits (with T6 equal to 1) at regular intervals while the counter value is lower than the time-window value (W[6:0])
- Early Wakeup Interrupt (EWI): occurs whenever the counter reaches 40h → can be used to reload the down counter
- WWDG reset flag (in RCC\_CSR) to inform when a WWDG reset occurs
- Min-max timeout value @48MHz (PCLK): 85.33μs / 43.69ms

**Best suited to applications which require the watchdog to react within an accurate timing window**



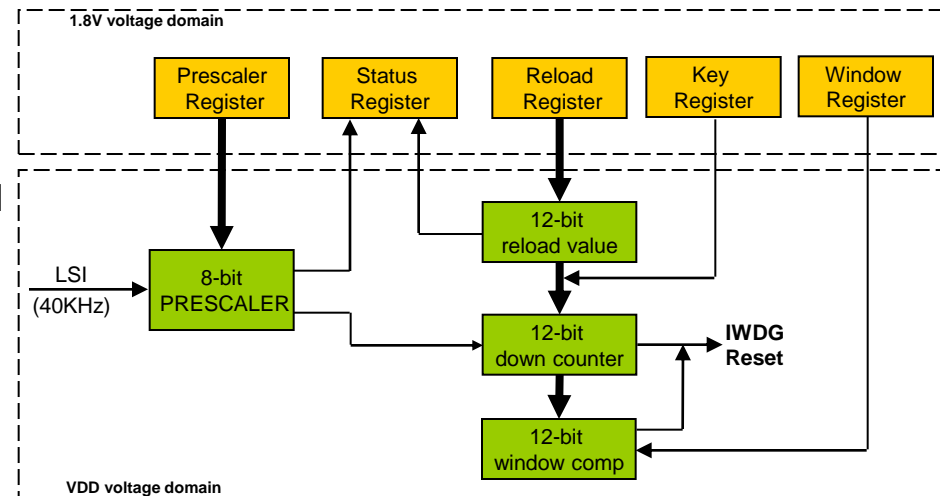


**Like STM32F1xx with  
added window**

System Peripherals


# Independent Watchdog (IWDG)

- Selectable HW/SW start through option byte
- Advanced security features:
  - IWDG clocked by its own dedicated low-speed clock (LSI) and thus stays active even if the main clock fails
  - Once enabled the IWDG can't be disabled (LSI can't be disabled too)
  - Safe Reload Sequence (key) + window
  - IWDG function implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake-up from STANDBY)
- To prevent IWDG reset: write IWDG\_KR with AAAAh key value at regular intervals before the counter reaches 0, while respecting the defined refresh window
- IWDG reset flag (in RCC\_CSR) to inform when a IWDG reset occurs
- Min-max timeout value @40KHz (LSI):  
100µs / 26.2s



***Best suited to applications which require the watchdog to run as a totally independent process outside the main application***



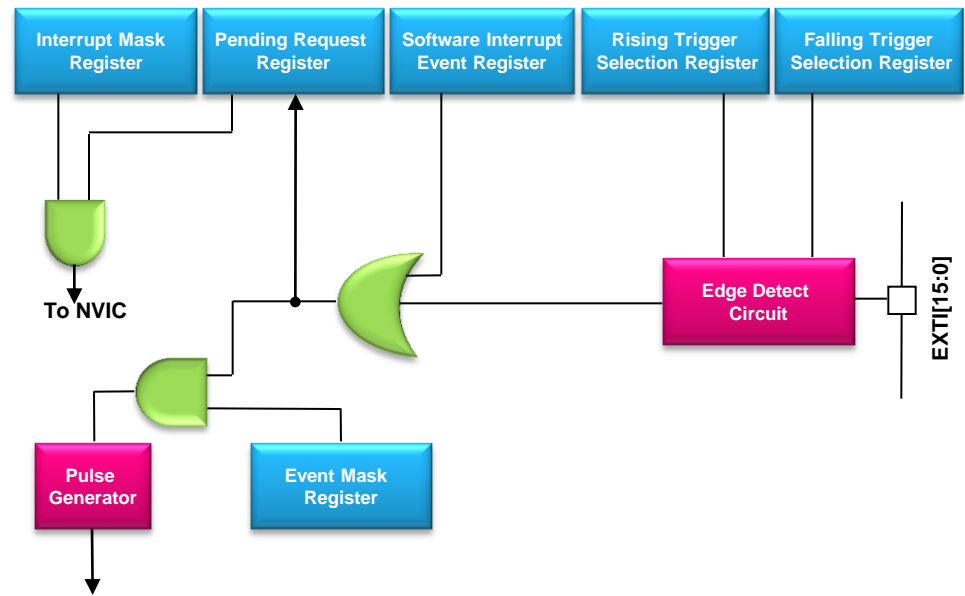


Same as STM32F1xx  
but with New features and  
new Name

System Peripherals

# **EXTENDED** INTERRUPT/EVENT CONTROLLER (EXTI)

- Manages the external and **internal asynchronous** events/interrupts and generates the event request to the CPU/Interrupt Controller and a wake-up request to the Power Manager
- Some communication peripherals (UART, I2C, CEC, comparators) are able to generate events when the system is in run/sleep mode and also when the **system is in stop mode** allowing to wake up the system from stop mode.
- These peripherals are able to generate both a synchronized (to the system APB clock) and an asynchronous version of the event.
- All others features are same as STM32F1x series



Up to 28 Interrupt/Events requests :

- Up to 55 GPIOs can be used as EXTI line(0..15)
- EXTI line 16 is connected to the PVD output
- EXTI line 17 is connected to the RTC Alarm event
- EXTI line 18 is reserved (internally held low)
- EXTI line 19 is connected to RTC tamper and Timestamps
- EXTI line 20 is reserved (internally held low)
- EXTI line 21 is connected to Comparator 1 output
- EXTI line 22 is connected to Comparator 2 output
- EXTI line 23 is connected to I2C1 wakeup
- EXTI line 24 is reserved (internally held low)
- EXTI line 25 is connected to USART1 wakeup
- EXTI line 26 is reserved (internally held low)
- EXTI line 27 is connected to CEC wakeup.



Same as STM32F-1  
with new features

Communication Peripherals

# Universal Synchronous Asynchronous Receiver Transmitter (USART)

Only USART1 available on STM32F050x products

# STM32F1/F2/L USART vs STM32F0 USART

## Main features (1/2)


Feature	STM32F0	STM32F1/2/L
Programmable data length (8 or 9 bits)	Yes	Yes
Configurable stop bits	1, 1.5, 2	0.5, 1, 1.5, 2
Synchronous mode (Master only)	Yes	Yes
Single wire Half duplex	Yes	Yes
Programmable parity	Yes	Yes
Hardware flow control (nCTS/nRTS)	Yes	Yes
Driver Enable (for RS485)	Yes	No
Swappable Tx/Rx pin	Yes	No
IrDA	Yes	Yes
Basic support for Modbus	Yes	No
LIN	Yes	Yes
Smartcard	Yes (T = 0, T=1)	Yes (T = 0)
Dual Clock domain and wake up from STOP mode	Yes	No
Programmable data order with MSB first or LSB first	Yes	No


# STM32F1/F2/L USART vs STM32F0 USART

## Main features (2/2)

Feature	STM32F0	STM32F1/2/L
Receiver timeout	Yes	No
Auto-baudrate detection	Yes	No
Continuous communication using DMA	Yes	Yes
Address/character match interrupt	Yes	No
End of Block interrupt	Yes	No
Multiprocessor communication	Yes	Yes

# Smart Card mode - STM32F0 vs STM32F1xx

Features		STM32F05x	STM32F1xx
Maximum USART baudrate in Smartcard mode		3Mbits/s	4.5Mbits/s
T = 0	In case of transmission error, according to the protocol specification, the USART should send the character.	The USART can handle automatic re-sending of data. The number of retries is programmable (8 max).	The data retry should be done by software.
	In case of reception error, according to the protocol specification, the smartcard must resend the same character.	The number of maximum retries is programmable (8 max). If the received character is still erroneous after the programmed number of retries, the USART will stop transmitting the NACK and will signal the error as a parity error.	
	A programmable guardtime is automatically inserted between two consecutive characters in transmission.	Yes	No
 T = 1	Character Wait Time (CWT)	Implemented using the new timeout feature.	All T = 1 feature should be implemented by software.
	Block Wait Time (BWT)		
	Block length and end of block detection		
	Direct/Inverse convention	Implemented using some new features: MSB/LSBFIRST, Binary data inversion etc...	



Same as STM32F-4  
with less features

System Peripherals

REAL-TIME CLOCK (RTC)

# RTC overview across families (1/2)

121

	<b>STM32L1x Med-density</b>	<b>STM32F2x</b>	<b>STM32F4x</b>	<b>STM32F0x</b>
RTC in VBAT	NO	<b>YES</b>		
Calendar in BCD	YES	YES	YES	<b>YES</b>
Calendar Sub seconds access	NO		<b>YES Resolution down to RTCCLK</b>	
Calendar synchronization on the fly	NO		<b>YES</b>	
Alarm on calendar	2	2	2 w/ subseconds	<b>1 w/ subseconds</b>
Calendar Calibration	Calib window : <b>64min</b> Calibration step: -2ppm/ +4ppm Range [-63ppm+126ppm]		Calib window : <b>8s/16s/32s</b> Calibration step: 3.81ppm/1.91ppm/ <b>0.95 ppm</b> Range [-480ppm +480ppm]	



# RTC overview across families (2/2)

122

	<b>STM32L1x Med-density</b>	<b>STM32F2x</b>	<b>STM32F4x</b>	<b>STM32F0x</b>
Synchronization on mains	<b>YES</b>			
Periodic wakeup	<b>YES</b>			<b>NO</b>
Timestamp	<b>YES</b> Sec, Min, Hour, Date		<b>YES</b> <b>Sec, Min, Hour, Date, Sub seconds</b>	
Tamper	<b>YES</b> 1 pins /1 event Edge Detection only	<b>YES</b> 2 pins /1 event Edge detection only	<b>YES</b> 2 pins /2 event Level Detection with Configurable filtering	<b>YES</b> <b>2 pins/ 2 events</b> <b>Level Detection with Configurable filtering</b>
32-bit Backup registers	20	20	20	<b>5</b>
PC13-14-15 output state kept in Standby (if not used by RTC/LSE)	<b>NO</b>			<b>YES</b>

# Smooth digital calibration

123

- Consists in masking/adding N (configurable) 32KHz clock pulses, fairly well distributed in a configurable window.
- A 1Hz output is provided to measure the quartz frequency and the calibration result.

Calibration window	Accuracy	Total range
8 s	<b><math>\pm 1.91</math> ppm</b>	<b>[0 <math>\pm 480</math>ppm]</b>
16s	<b><math>\pm 0.95</math> ppm</b>	<b>[0 <math>\pm 480</math>ppm]</b>
32s	<b><math>\pm 0.48</math> ppm</b>	<b>[0 <math>\pm 480</math>ppm]</b>



New peripheral

System Peripherals

# COMPARATOR (COMP)

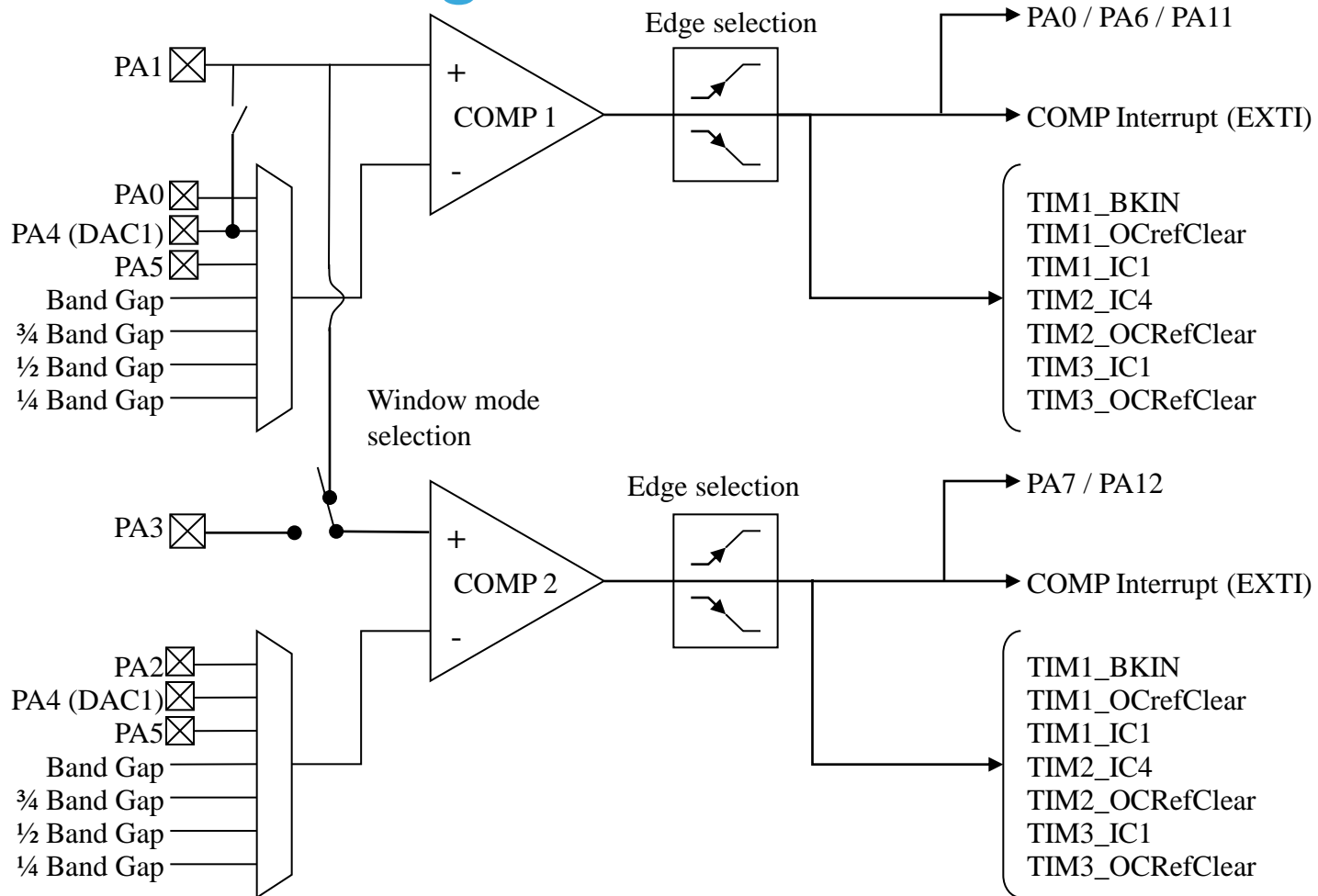
Feature not available on STM32F050x products

# COMPARATORS (COMP)

125

- 2 general purpose comparators
  - Rail-to-rail inputs
  - Programmable speed / consumption
  - It can be used as standalone devices; All I/Os including outputs are available
  - Can be combined into a window comparator
  - Multiple choices for threshold and output redirection
- Can be used for:
  - Exiting low power modes on analog event
  - Signal conditioning
  - Cycle-by-cycle current control (w/ DAC and TIM)
  - ...

# Block diagram COMP1 & COMP2

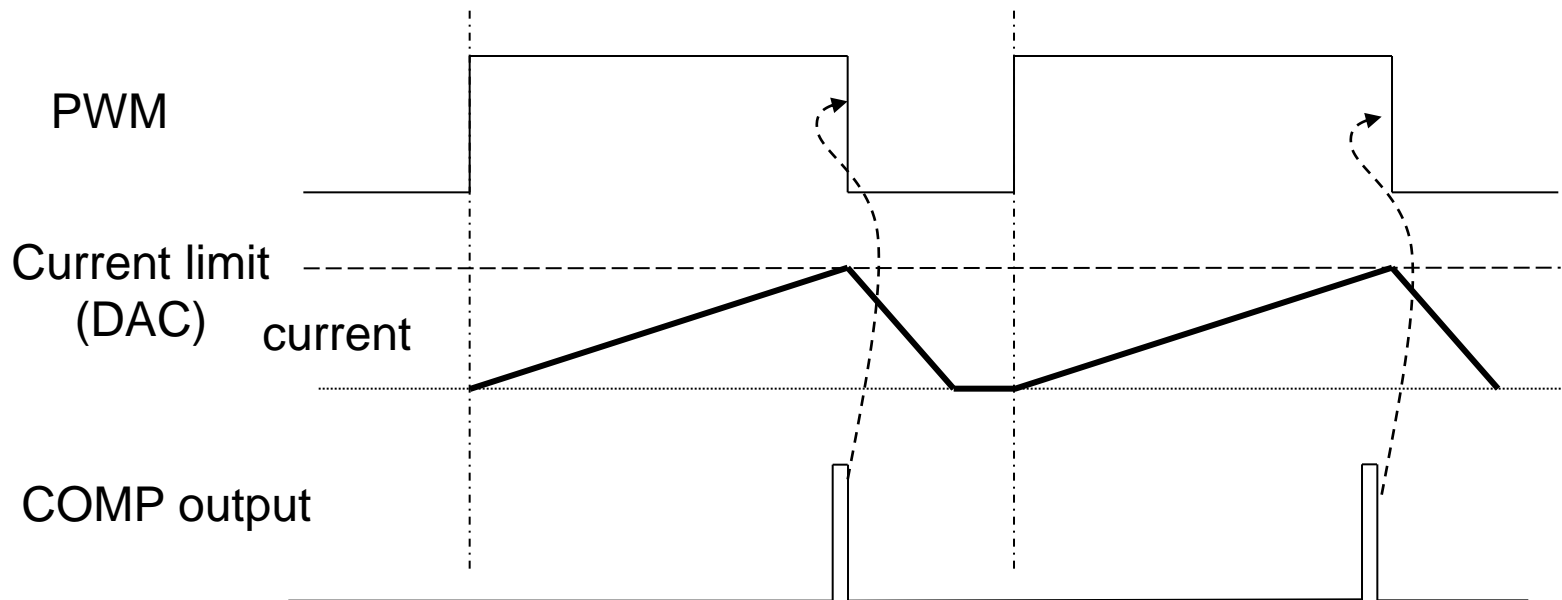


*BKIN: PWM's "Emergency stop" input*

*OCRefClear: PWM clear for cycle-by-cycle current controller*

- **Parametrics at a glance**
  - Full voltage range  $2V < V_{dda} < 3.6V$
  - Propagation time vs consumption (typ.  $2.7 < V_{dd} < 3.6V$ , for 200 mV step with 100 mV overdrive)
    - High speed mode: 120ns / 100 $\mu$ A
    - Medium speed mode: 400ns / 10 $\mu$ A
    - Low power mode: 1 $\mu$ s / 3 $\mu$ A
    - Ultra-low power mode: 3.5 $\mu$ s / 1 $\mu$ A
  - Input offset: +/-5mV typ, +/- 20mV max
  - Programmable hysteresis: 0, 8, 15, 31 mV
  
- **Fully asynchronous operation**
  - Comparators working in STOP mode
  - No clock related propagation delay
  
- **Functional safety (Class B)**
  - The comparator configuration can be locked with a write-once bit

- Cycle-by cycle current control
  - Uses DAC for threshold and Ocref\_clr timer input





From STM32F1  
with new features

Analog Peripherals

# ANALOG –TO-DIGITAL CONVERTER (ADC)



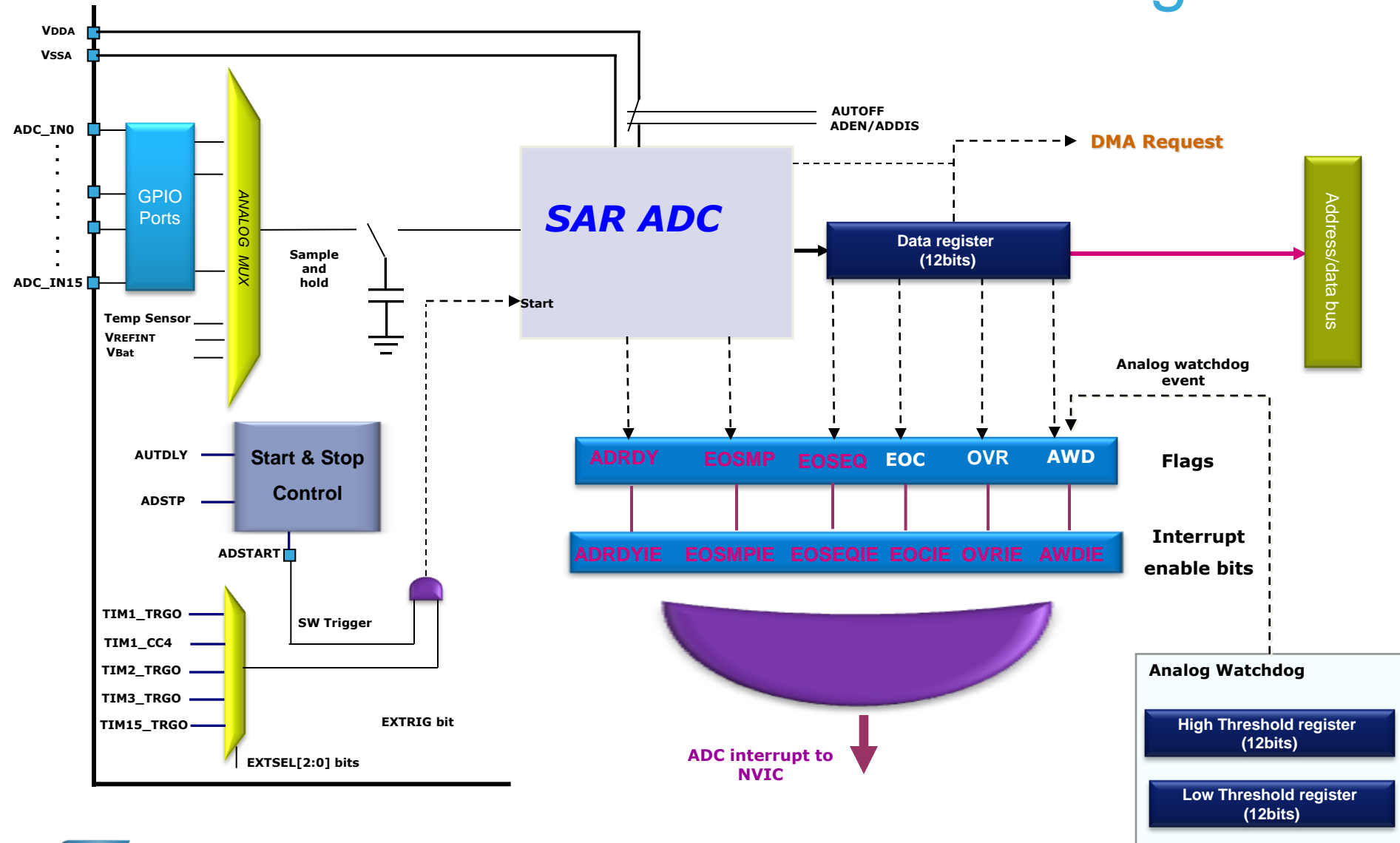
- ADC conversion rate 1 MHz and 12-bit resolution
- Programmable Conversion resolution : 12, 10, 8 or 6 bit
- ADC supply requirement:  $VDDA = 2.4\text{ V to }3.6\text{ V}$
- ADC input range:  $VSSA \leq VIN \leq VDDA$
- Up to 19 multiplexed channels:
  - 16 external channels
  - 3 internal channels connected to :
    - Temperature sensor VSense
    - Internal voltage reference VREFINT
    - VBAT power supply

# ADC Features (2/3)

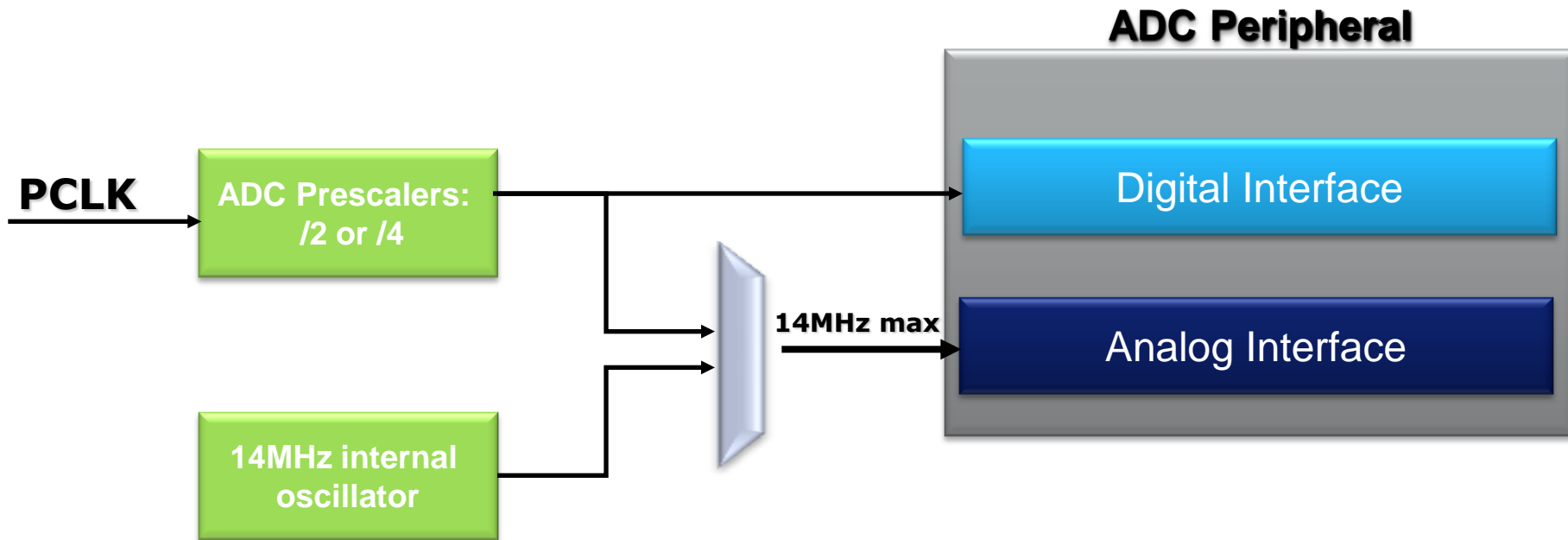
- Programmable sampling time
- Self-calibration
- Configurable scanning direction
- Single, continuous and discontinuous conversion modes
- Left or right Data alignment with inbuilt data coherency
- Software or Hardware start of conversion
- Analog Watchdog on high and low thresholds

- DMA capability
- Auto Delay insertion between conversions
- Auto-OFF mode (Power Saving)
- Interrupt generation on:
  - ADC ready
  - End of Conversion
  - End of sequence of conversions
  - End of sampling phase
  - Analog watchdog
  - Overrun

# ADC Block Diagram



- The ADC has a dual clock-domain architecture:
  - Dedicated 14MHz clock,
  - PCLK clock divided by 2 or divided by 4



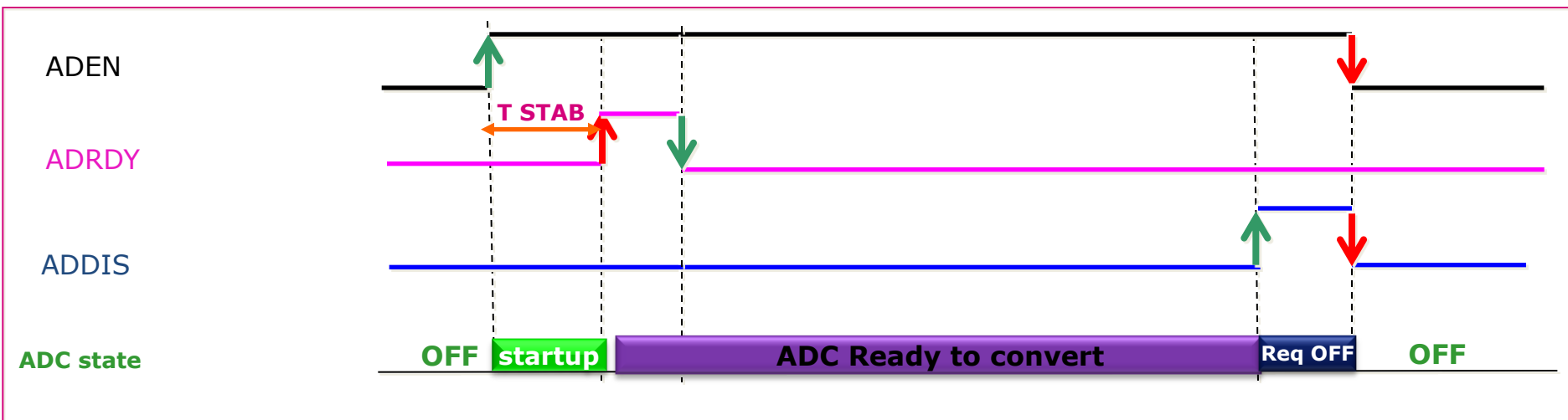
# How to choose ADC Clock

ADC clock source	benefits	drawbacks
Dedicated 14 MHz clock	<ul style="list-style-type: none"><li>▪ Guaranteed maximum speed whatever the MCU operating frequency,</li><li>▪ Capability to use the low power auto-off mode (automatically ON/OFF switch of 14 MHz internal oscillator)</li></ul>	<ul style="list-style-type: none"><li>▪ Jitter is induced by the resynchronization of the trigger signal</li></ul> <p>➔ Latency between the trigger event and the start of conversion is not deterministic</p>
PCLK divided by 2 or by 4	<ul style="list-style-type: none"><li>▪ Avoiding clock domain resynchronization (no jitter),</li><li>▪ Latency between the trigger event and the start of conversion is deterministic: guarantees that the interval from one conversion to the other is constant</li></ul>	<ul style="list-style-type: none"><li>▪ ADC conversion time depends on system clock frequency</li></ul>

# ADC ON OFF control

136

- To enable ADC: Set ADEN=1 then wait till ADRDY flag will be equal to 1,
- When ADC is clocked by 14MHz internal oscillator while ADC interface is clocked by the APB, ADRDY signal guarantees that ADC data will be transmitted from one domain to the other.
- ADC cannot be re-programmed unless it is stopped (ADSTART = 0).



↕↕ By Software

↕↕ By Hardware

■ ADC Startup

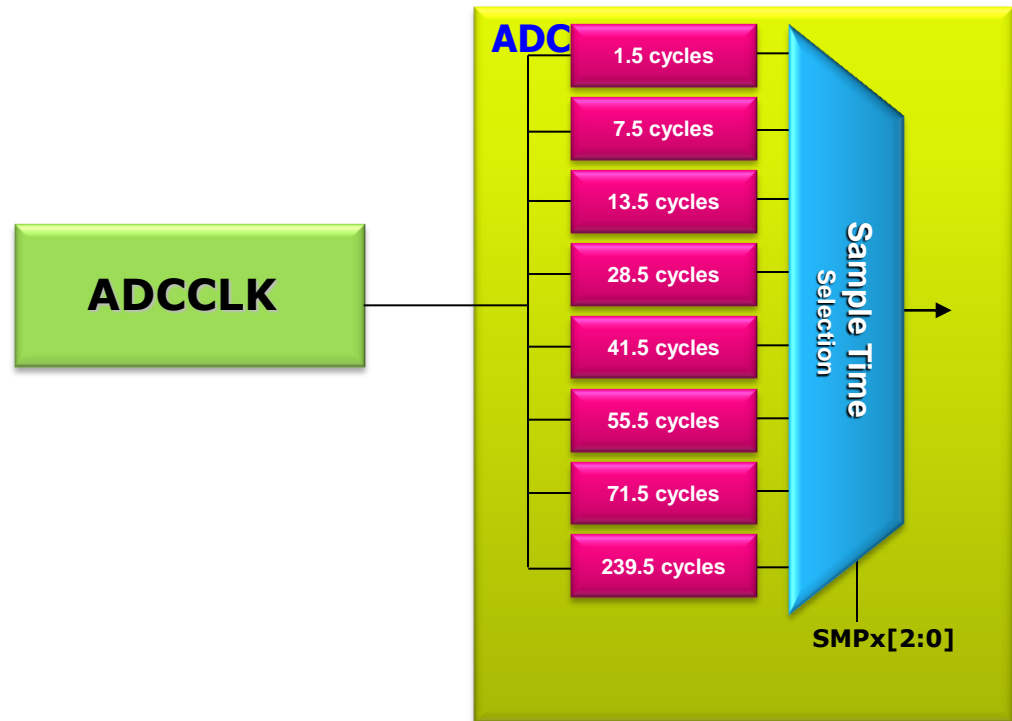
■ ADC ready

■ OFF Request

# ADC Sampling Time ( $T_{\text{Sampling}}$ )

- Three bits programmable sampling time which once programmed, it applies to all channels to be converted (Channel by channel programmable sampling time is not allowed).

- 1.5 ADC clock cycles
- 7.5 ADC clock cycles
- 13.5 ADC clock cycles
- 28.5 ADC clock cycles
- 41.5 ADC clock cycles
- 55.5 ADC clock cycles
- 71.5 ADC clock cycles
- 239.5 ADC clock cycles





# End of sampling

138

- The ADC indicates the end of sampling phase by setting the EOSMP flag in the ADC\_ISR register.
- The EOSMP flag is cleared by software by writing 1 to it.
- An interrupt can be generated if the EOSMPIE bit is set in the ADC\_IER register.



➔ As soon as the sampling is completed it is possible to prepare next conversion (for instance switching I/Os) during the conversion phase.

# Total Conversion Time

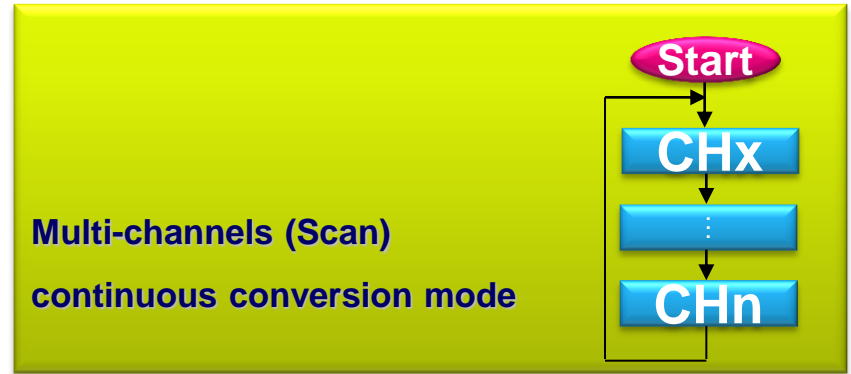
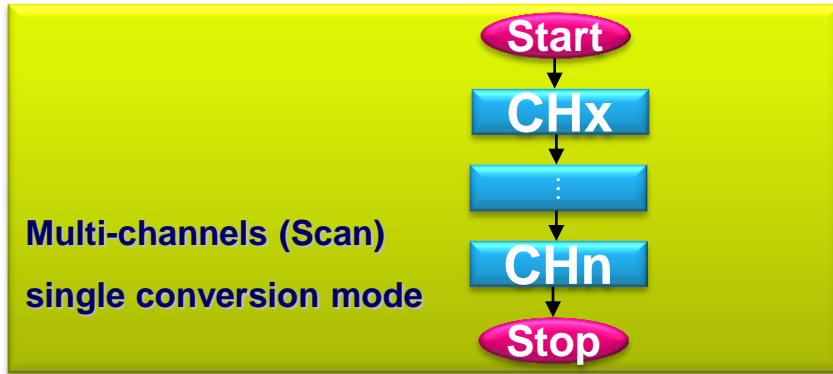
- Total conversion Time =  $T_{\text{Sampling}} + T_{\text{Conversion}}$

Resolution	$T_{\text{Conversion}}$	$T_{\text{Sampling}}$	Total conversion time	tADC at fADC = 14MHz
12 bit	12.5 Cycles	1.5 Cycles	14 Cycles	1uS
10 bit	11.5 Cycles	1.5 Cycles	13 Cycles	928ns
8 bit	9.5 Cycles	1.5 Cycles	11 Cycles	785ns
6 bit	7.5 Cycles	1.5 Cycles	9 Cycles	643ns

➔ Lower resolution allows faster conversion times for applications where high data precision is not required.

# ADC conversion modes

- Five conversion modes are available:



Discontinuons  
conversion mode

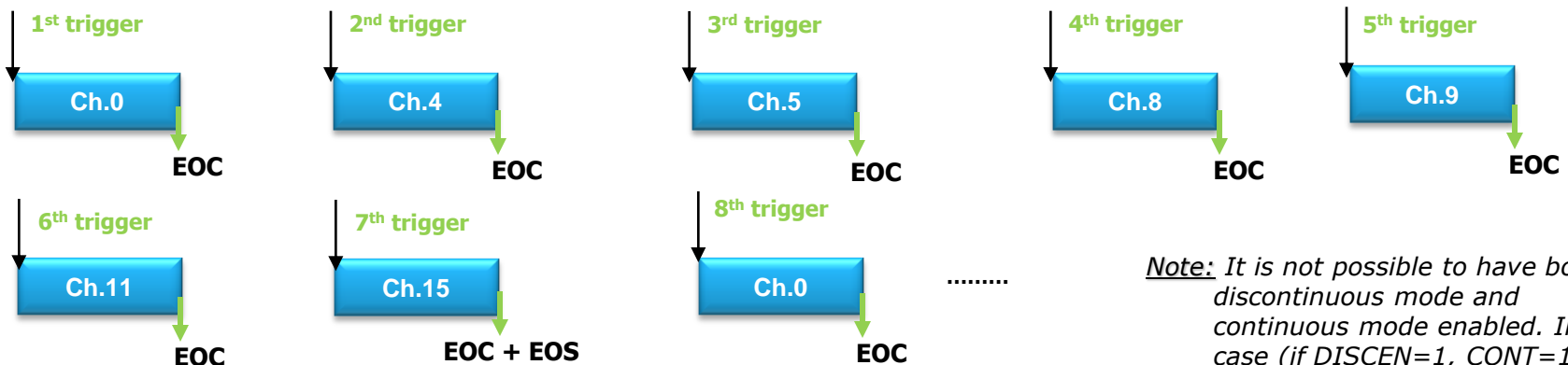


# ADC discontinuous conversion mode

- Enabled when DISCEN bit equal to 1,
- Split channels conversion sequence into sub-sequences of only one channel,
- Need a hardware or software trigger event to start each conversion defined in the sequence

## **Example:**

- Conversion of channels: 0, 4, 5, 8, 9, 11, and 15
- DISCEN = 1

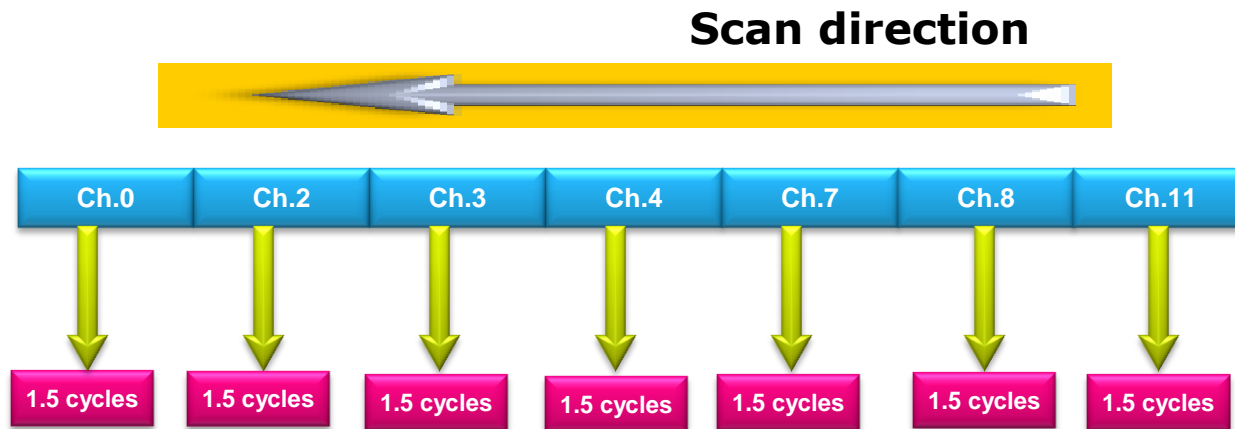


**Note:** It is not possible to have both discontinuous mode and continuous mode enabled. In this case (if DISCEN=1, CONT=1), the ADC behaves as if continuous mode was disabled.

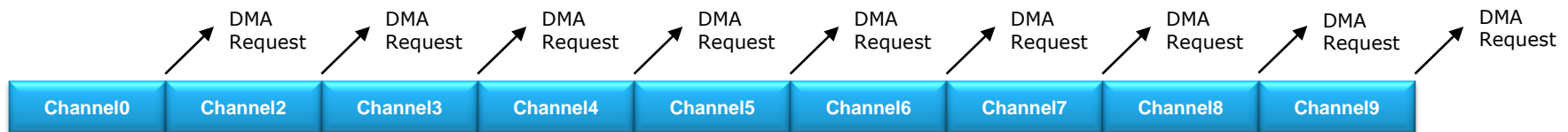
- Up to 19 conversions with programmable order, fixed sampling time and programmable scanning direction

**Example:** - Conversion of channels: 0, 2, 3, 4, 7, 8 and 11

- Only one sampling time to be applied to all the channels
- SCANDIR=1: backward scan



- DMA request generated on each ADC end of channel conversion.



ConvertedValue\_Tab[9]



### **Example:**

- Conversion of regular channels: 0, 2, 3, 4, 5, 6, 7, 8 and 9
- Converted data stored in ConvertedValue\_Tab[9]
- DMA transfer enabled (destination address auto incremented)

**Note: Each time DMA access to ADC\_DR register, EOC flag is automatically cleared**

# ADC Overrun management

- The overrun flag (OVR) indicates a buffer overrun event,
- An interrupt can be generated if the OVRIE bit is set in the ADC\_IER register,
- It is possible to configure if the data is preserved or overwritten when an overrun event occurs by programming the OVRMOD bit:
  - **OVRMOD=0:**

An overrun event preserves the data register from being overwritten: the old data is maintained and the new conversion is discarded. If OVR remains at 1, further conversions can be performed but the resulting data is discarded.
  - **OVRMOD=1:**

The data register is overwritten with the last conversion result and the previous unread data is lost. If OVR remains at 1, further conversions can be performed and the ADC\_DR register always contains the data from the latest conversion.

# ADC Analog Watchdogs

- 12-bit programmable analog watchdog low and high thresholds
- Enabled on one, some or all converted channels
- Interrupt generation on low or high thresholds detection



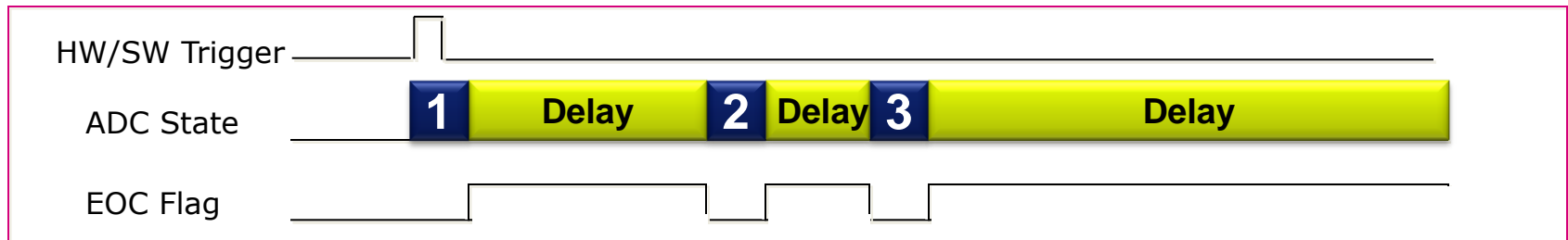



# Auto delayed conversion

146

- **Auto Delay Mode**

- When  $AUTDLYbits = 1$ , a new conversion can start only if the previous data has been treated, once the `ADC_DR` register has been read or if the `EOC` bit has been cleared.



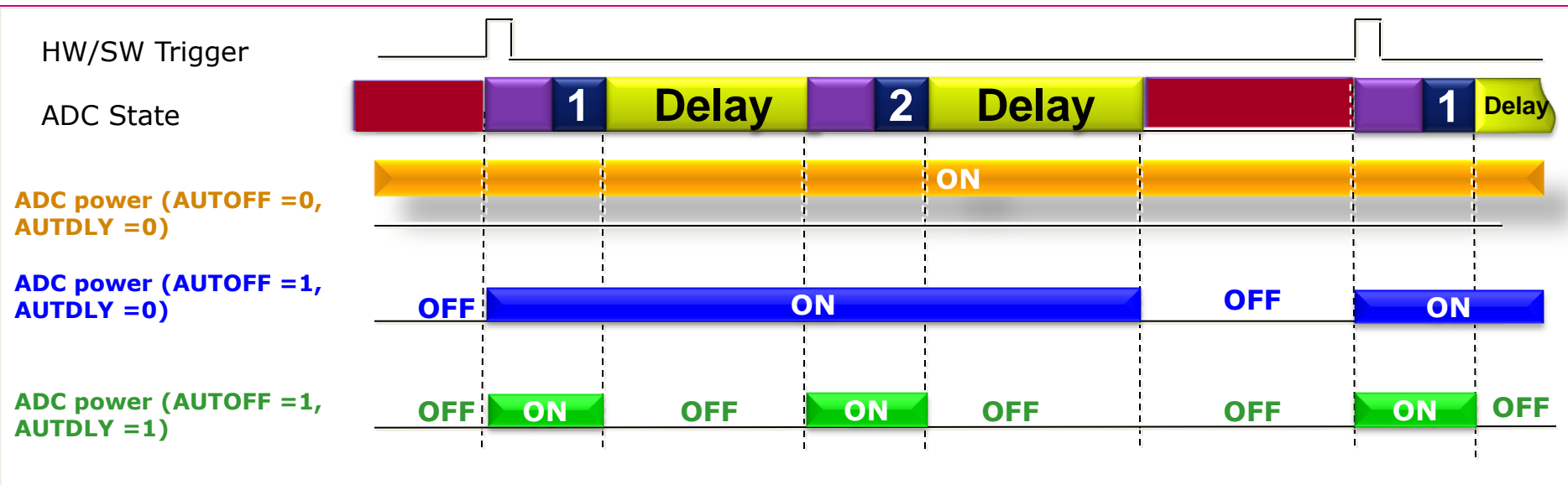
 Channel conversion #i

Note : A trigger event (for the same group of conversions) occurring during this delay is ignored.

➔ This is a way to automatically adapt the speed of the ADC to the speed of the system that reads the data.

# Auto-OFF mode (Power Saving)

- ADC power-on and power-off can be managed by hardware and turn OFF the 14 MHz internal oscillator in order to save power. The ADC can be powered down:
  - During the ADC is waiting for a trigger event (AUTOFF= 1) → The ADC is powered up at the next trigger event.
  - During the delay and waiting for a trigger event (AUTDLY= 1 and AUTOFF= 1) → The ADC is powered up again at the end of the delay and at the next trigger event.



ADC Waiting for Trigger



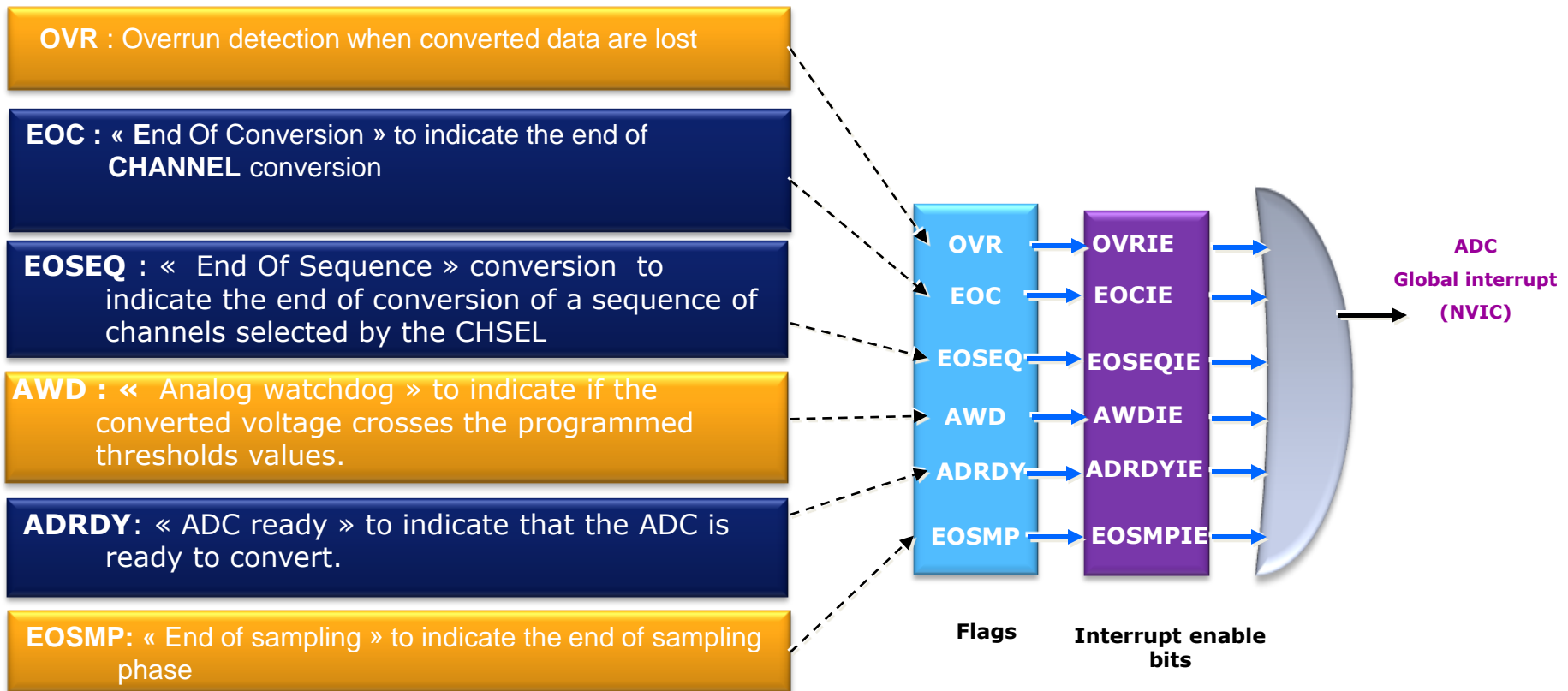
Startup Time



Channel conversion #i

# ADC Flags and interrupts

148



# STM32F0 ADC Vs. STM32F1 ADC (1/2)

ADC	STM32F0 ADC	STM32F1 ADC
ADC Type	SAR structure	SAR structure
Instances	1 ADC	Up to 3 ADC
Max Sampling freq	1 MSPS	1 MSPS
Number of channels	up to 16 channels + 3 internals	up to 21 channels
Resolution	12-bit, 10-bit, 8-bit, 6-bit	12-bit
Sequencer	No, Simple scanning logic	Yes
DMA	Yes	Yes
Supply requirement	2.4 V to 3.6 V	2.4 V to 3.6 V
ADC Clock	PCLK /2 or /4 or 14MHz	APB2 divided by a prescaler

# STM32F0 ADC Vs. STM32F1 ADC (2/2)

ADC	STM32F0 ADC	STM32F1 ADC
Auto delayed mode	Yes	No
Auto-OFF mode	Yes	No
End of sampling interrupt	Yes	No
Analog watchdog	Yes	Yes
Channel Type	No Regular / injected notion	Regular / Injected
Supply requirement	2.4 V to 3.6 V	2.4 V to 3.6 V



Same as STM32F-2

Analog Peripherals

# DIGITAL-TO-ANALOG CONVERTER (DAC)

Feature not available on STM32F050x products

# DAC Features (1/2)

152

- Two DAC converters: one output channel for each one
- 8-bit or 12-bit monotonic output
- Left or right data alignment in 12-bit mode
- Synchronized update capability
- Noise-wave or Triangular-wave generation
- Dual DAC channel independent or simultaneous conversions

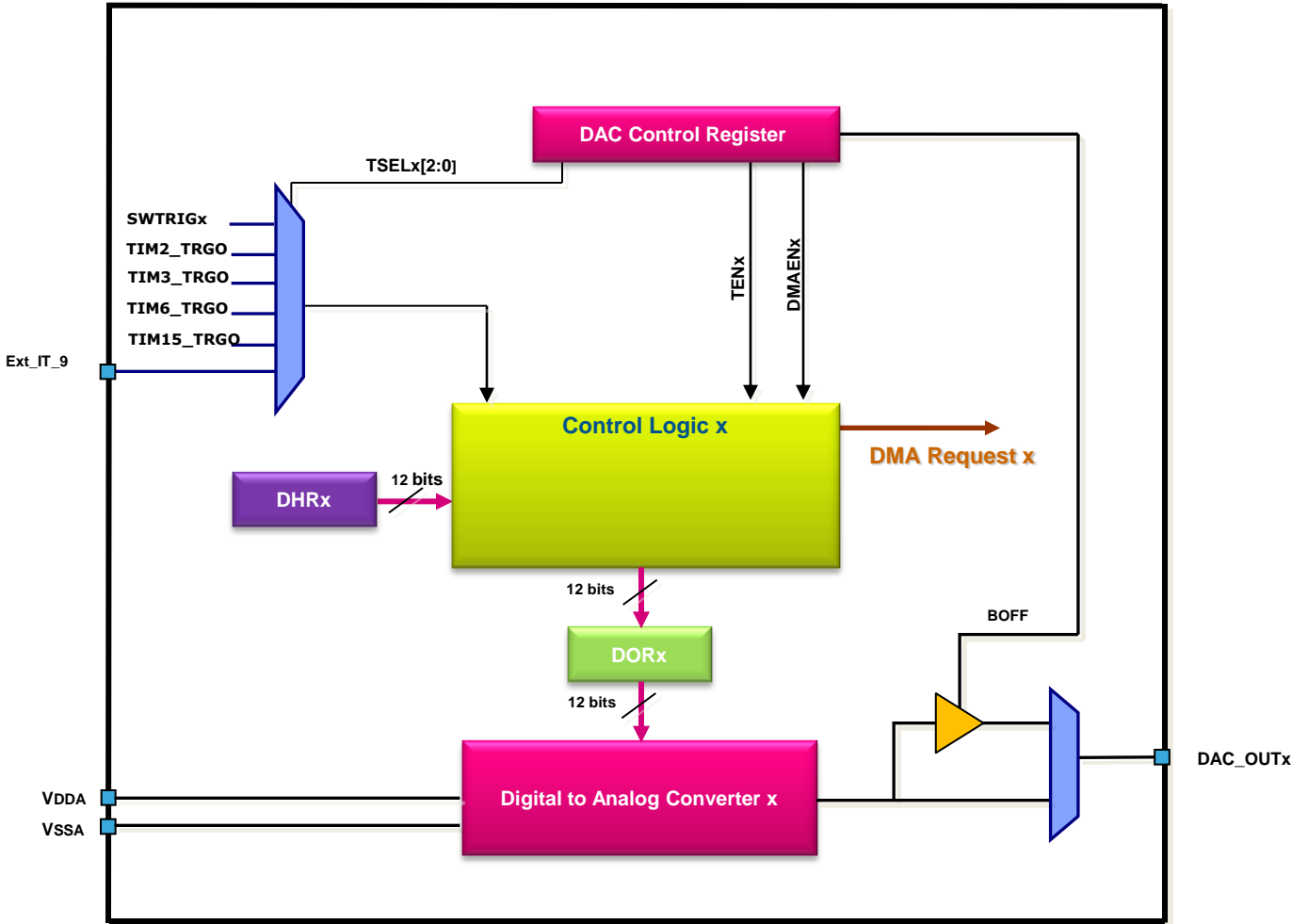
# DAC Features (2/2)

153

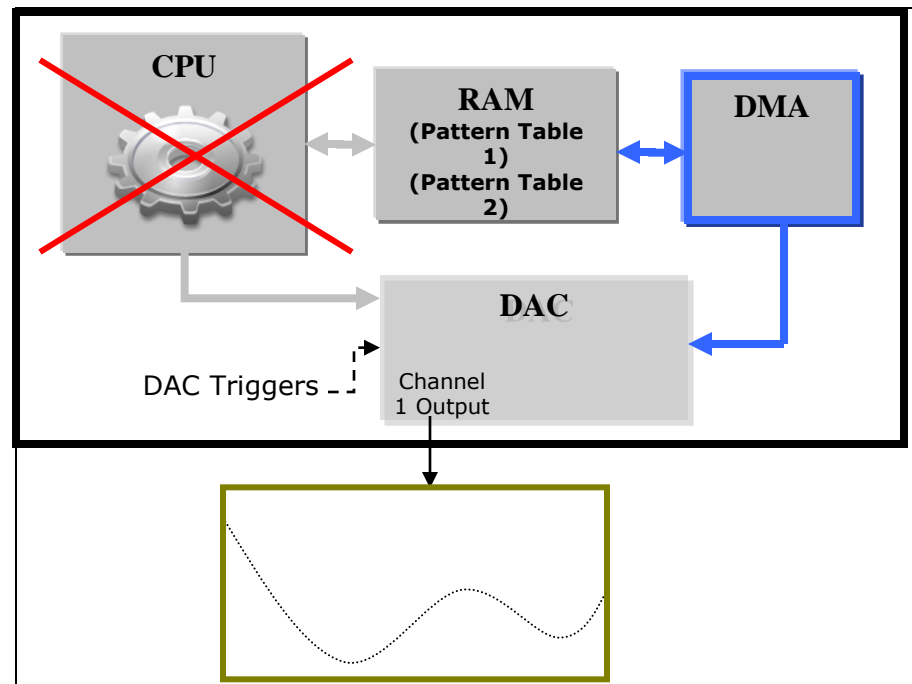
- DMA capability for each channel
- DMA underrun error detection
- External triggers for conversion
- Programmable internal buffer
- DAC supply requirement:  $VDDA = 2.4V$  to  $3.6 V$
- DAC outputs range:  $0 \leq DAC\_OUTx \leq VDDA$



# DAC Channelx Block Diagram



- A DAC DMA request is generated when an external trigger occurs:  
The value of the DAC\_DHR register is then transferred to the DAC\_DOR register.
- DMA underrun error detection with interrupt capability

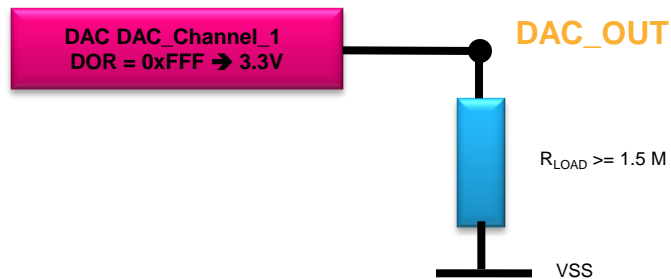


# DAC Buffered output

156

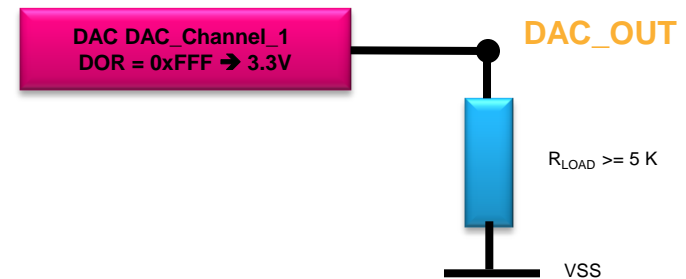
- To drive external loads without using an external operational amplifier, DAC channels have embedded output buffers which can be enabled and disabled depending on the user application.
- When the DAC output is not buffered, and there is a load in the user application circuit, the voltage output will be lower than the desired voltage. Enabling the buffer, the voltage output and the voltage desired are similar.

## Output buffer disabled



- Due to the  $R_{LOAD}$  resistor the measured voltage on output could be lower than the required.

## Output buffer enabled

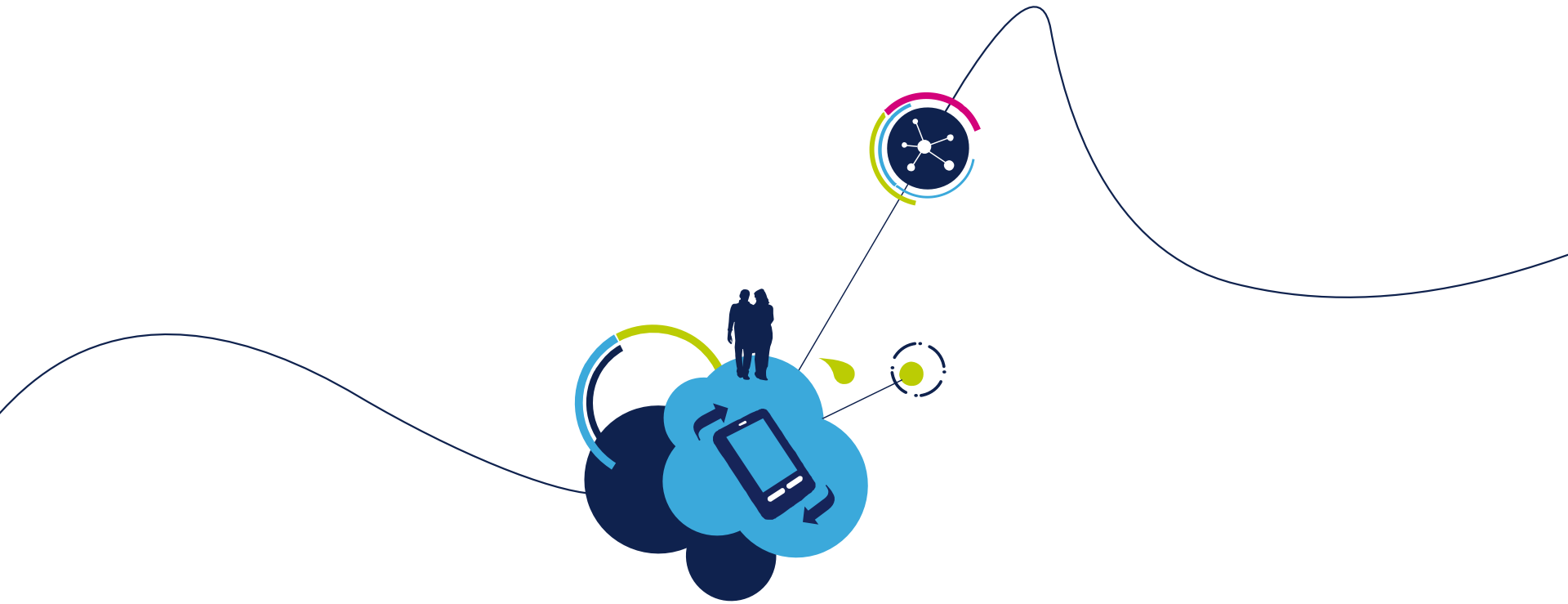


- The measured output voltage is similar to the required voltage despite of  $R_{LOAD}$ : The output buffer reduced the output impedance.

STM32<sup>★</sup> Releasing your **creativity**



THANK YOU!



END