

## Applicability

This document applies to the part numbers of STM32F031x4/x6 devices listed in [Table 1](#) and their variants shown in [Table 2](#).

[Section 1](#) gives a summary and [Section 2](#) a description of device limitations and documentation errata, with respect to the device datasheet and reference manual RM0091.

**Table 1. Device summary**

Reference	Part numbers
STM32F031x4	STM32F031C4, STM32F031F4, STM32F031G4, STM32F031K4
STM32F031x6	STM32F031C6, STM32F031F6, STM32F031G6, STM32F031K6, STM32F031E6

**Table 2. Device variants**

Reference	Silicon revision codes	
	Device marking <sup>(1)</sup>	REV_ID <sup>(2)</sup>
STM32F031x4/x6	A or 1	0x1000

1. Refer to the device data sheet for how to identify this code on different types of package.
2. REV\_ID[15:0] bit field of DBGMCU\_IDCODE register. Refer to the reference manual.

# Contents

<b>1</b>	<b>Summary of device errata</b>	<b>4</b>
<b>2</b>	<b>Description of device errata</b>	<b>6</b>
2.1	System	6
2.1.1	Wakeup sequence from Standby mode when using more than one wakeup source	6
2.2	GPIO	6
2.2.1	Extra consumption on GPIO PBO on 20-pin devices	6
2.2.2	GPIOx locking mechanism not working properly for GPIOx_OTYPER register	7
2.3	DMA	7
2.3.1	DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	7
2.3.2	Byte and half-word accesses not supported	7
2.4	ADC	8
2.4.1	Overrun flag not set if EOC reset coincides with new conversion end	8
2.4.2	ADEN bit cannot be set immediately after the ADC calibration	8
2.5	IWDG	8
2.5.1	RVU, PVU and WVU flags are not reset in STOP mode	8
2.5.2	RVU, PVU and WVU flags are not reset with low-frequency APB	9
2.6	RTC	9
2.6.1	Spurious tamper detection when disabling the tamper channel	9
2.6.2	A tamper event preceding the tamper detect enable not detected	9
2.6.3	RTC calendar registers are not locked properly	10
2.6.4	RTC interrupt can be masked by another RTC interrupt	10
2.7	I2C	12
2.7.1	Wrong data sampling when data setup time ( $t_{SU,DAT}$ ) is shorter than one I2C kernel clock period	12
2.7.2	Spurious bus error detection in master mode	12
2.7.3	10-bit slave mode: wrong direction bit value upon Read header receipt	12
2.7.4	10-bit combined with 7-bit slave mode: ADDCODE may indicate wrong slave address detection	13
2.7.5	Wakeup frames may not wake up the MCU when Stop mode entry follows I2C enabling	14

2.7.6	Wakeup frame may not wake up the MCU from Stop mode if $t_{HD;STA}$ is close to I2C kernel clock startup time	14
2.7.7	Wrong behavior in Stop mode when wakeup from Stop mode is disabled in I2C	15
2.7.8	10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave	15
2.7.9	Last-received byte loss in reload mode	16
2.7.10	Spurious master transfer upon own slave address match	17
2.8	USART	18
2.8.1	Non-compliant sampling for NACK signal from smartcard	18
2.8.2	Break request preventing TC flag from being set	18
2.8.3	RTS is active while RE = 0 or UE = 0	18
2.8.4	Consistency not checked in mode 1 of automatic baud rate detection	19
2.8.5	Framing error (FE) flag low upon automatic baud rate detection error	19
2.8.6	Communication parameters reprogramming after ATR in Smartcard mode when SCLK is used to clock the card	19
2.8.7	Last byte written in TDR might not be transmitted if TE is cleared just after writing in TDR	19
2.9	SPI/I2S	20
2.9.1	BSY bit may stay high when SPI is disabled	20
2.9.2	BSY bit may stay high at the end of data transfer in slave mode	20
2.9.3	CRC error in SPI slave mode if internal NSS changes before CRC transfer	21
2.9.4	SPI CRC corruption upon DMA transaction completion by another peripheral	21
2.9.5	Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback	21
2.9.6	Packing mode limitation at reception	22
2.9.7	In I <sup>2</sup> S slave mode, enabling I2S while WS is active causes desynchronization	23
2.9.8	Data flow corruption in master receiver TI half-duplex mode	23
2.10	Package	24
2.10.1	Inversed date code in the package marking versus datasheet	24
<b>3</b>	<b>Revision history</b>	<b>26</b>

# 1 Summary of device errata

The following table gives a quick reference to all documented device limitations of STM32F031x4/x6 and their status:

A = workaround available

N = no workaround available

P = partial workaround available

Applicability of a workaround may depend on specific conditions of target application. Adoption of a workaround may cause restrictions to target application. Workaround for a limitation is deemed partial if it only reduces the rate of occurrence and/or consequences of the limitation, or if it is fully effective for only a subset of instances on the device or in only a subset of operating modes, of the function concerned.

**Table 3. Summary of device limitations**

Function	Section	Limitation	Status
			Rev. A or 1
System	2.1.1	Wakeup sequence from Standby mode when using more than one wakeup source	A
GPIO	2.2.1	Extra consumption on GPIO PB0 on 20-pin devices	A
	2.2.2	GPIOx locking mechanism not working properly for GPIOx_OTYPER register	P
DMA	2.3.1	DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	A
ADC	2.4.1	Overrun flag not set if EOC reset coincides with new conversion end	A
	2.4.2	ADEN bit cannot be set immediately after the ADC calibration	A
IWDG	2.5.1	RVU, PVU and WVU flags are not reset in STOP mode	A
	2.5.2	RVU, PVU and WVU flags are not reset with low-frequency APB	N
RTC	2.6.1	Spurious tamper detection when disabling the tamper channel	P
	2.6.2	A tamper event preceding the tamper detect enable not detected	A
	2.6.3	RTC calendar registers are not locked properly	A
	2.6.4	RTC interrupt can be masked by another RTC interrupt	A
I2C	2.7.1	Wrong data sampling when data setup time ( $t_{SU,DAT}$ ) is shorter than one I2C kernel clock period	P
	2.7.2	Spurious bus error detection in master mode	A
	2.7.3	10-bit slave mode: wrong direction bit value upon Read header receipt	A
	2.7.4	10-bit combined with 7-bit slave mode: ADDCODE may indicate wrong slave address detection	N
	2.7.5	Wakeup frames may not wake up the MCU when Stop mode entry follows I2C enabling	A

**Table 3. Summary of device limitations (continued)**

Function	Section	Limitation	Status
			Rev. A or 1
I2C	2.7.6	Wakeup frame may not wake up the MCU from Stop mode if $t_{HD;STA}$ is close to I2C kernel clock startup time	P
	2.7.9	Last-received byte loss in reload mode	A
	2.7.10	Spurious master transfer upon own slave address match	A
USART	2.8.1	Non-compliant sampling for NACK signal from smartcard	N
	2.8.2	Break request preventing TC flag from being set	A
	2.8.3	RTS is active while RE = 0 or UE = 0	A
	2.8.4	Consistency not checked in mode 1 of automatic baud rate detection	N
	2.8.5	Framing error (FE) flag low upon automatic baud rate detection error	A
	2.8.6	Communication parameters reprogramming after ATR in Smartcard mode when SCLK is used to clock the card	A
	2.8.7	Last byte written in TDR might not be transmitted if TE is cleared just after writing in TDR	A
SPI/I2S	2.9.1	BSY bit may stay high when SPI is disabled	A
	2.9.2	BSY bit may stay high at the end of data transfer in slave mode	A
	2.9.3	CRC error in SPI slave mode if internal NSS changes before CRC transfer	A
	2.9.4	SPI CRC corruption upon DMA transaction completion by another peripheral	P
	2.9.5	Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback	A
	2.9.6	Packing mode limitation at reception	P
	2.9.7	In I <sup>2</sup> S slave mode, enabling I2S while WS is active causes desynchronization	P
	2.9.8	Data flow corruption in master receiver TI half-duplex mode	P
Package	2.10.1	Inversed date code in the package marking versus datasheet	P

The following table gives a quick reference to the device documentation errata.

**Table 4. Summary of documentation errata**

Function	Section	Documentation erratum
DMA	2.3.2	Byte and half-word accesses not supported
I2C	2.7.7	Wrong behavior in Stop mode when wakeup from Stop mode is disabled in I2C

## 2 Description of device errata

The following sections describe limitations of the applicable devices with Arm<sup>®(a)</sup> core and provide workarounds if available. They are grouped by device functions.



### 2.1 System

#### 2.1.1 Wakeup sequence from Standby mode when using more than one wakeup source

##### Description

The various wakeup sources are logically OR-ed in front of the rising-edge detector that generates the wakeup flag (WUF). The WUF needs to be cleared prior to Standby mode entry, otherwise the MCU wakes up immediately.

If one of the configured wakeup sources is kept high during the clearing of the WUF (by setting the CWUF bit), it may mask further wakeup events on the input of the edge detector. As a consequence, the MCU might not be able to wake up from Standby mode.

##### Workaround

Apply the following sequence before entering Standby mode:

1. Disable all used wakeup sources
2. Clear all related wakeup flags
3. Re-enable all used wakeup sources
4. Enter Standby mode

*Note:* Be aware that, when applying this workaround, if one of the wakeup sources is still kept high, the MCU enters Standby mode but then it wakes up immediately, generating a power reset.

### 2.2 GPIO

#### 2.2.1 Extra consumption on GPIO PB0 on 20-pin devices

##### Description

For lower pin count devices, some GPIOs are not available on the package. The hardware forces them to safe configuration.

In this situation the software reconfiguration of PB0 to analog mode opens a path between  $V_{DDA}$  and  $V_{DDIOx}$ . Additional current consumption in the range of tens of  $\mu A$  can be observed if  $V_{DDA}$  is higher than  $V_{DDIOx}$ .

---

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

**Workaround**

Do not reconfigure PB0 to the analog mode when PB0 GPIO is not available on the device.

**2.2.2 GPIOx locking mechanism not working properly for GPIOx\_OTYPER register****Description**

Locking of GPIOx\_OTYPER[i] with i = 15..8 depends from setting of GPIOx\_LCKR[i-8] and not from GPIOx\_LCKR[i]. GPIOx\_LCKR[i-8] is locking GPIOx\_OTYPER[i] together with GPIOx\_OTYPER[i-8]. It is not possible to lock GPIOx\_OTYPER[i] with i = 15..8, without locking also GPIOx\_OTYPER[i-8].

**Workaround**

The only way to lock GPIOx\_OTYPER[i] with i=15..8 is to lock also GPIOx\_OTYPER[i-8].

**2.3 DMA****2.3.1 DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear****Description**

Upon a data transfer error in a DMA channel x, both the specific TEIFx and the global GIFx flags are raised and the channel x is normally automatically disabled. However, if in the same clock cycle the software clears the GIFx flag (by setting the CGIFx bit of the \_IFCR register), the automatic channel disable fails and the TEIFx flag is not raised.

This issue does not occur with ST's HAL software that does not use and clear the GIFx flag, but uses and clears the HTIFx, TCIFx, and TEIFx specific event flags instead.

**Workaround**

The only way to lock GPIOx\_OTYPER[i] with i=15..8 is to lock also GPIOx\_OTYPER[i-8].

**2.3.2 Byte and half-word accesses not supported****Description**

Some reference manual revisions may wrongly state that the DMA registers are byte- and half-word-accessible. Instead, the DMA registers must always be accessed through aligned 32-bit words. Byte or half-word write accesses cause an erroneous behaviour.

ST's low-level driver and HAL software only use aligned 32-bit accesses to the DMA registers.

This is a description inaccuracy issue rather than a product limitation.

**Workaround**

No application workaround is required.

## 2.4 ADC

### 2.4.1 Overrun flag not set if EOC reset coincides with new conversion end

#### Description

If the EOC flag is cleared by ADC\_DR register read operation or by software during the same APB cycle in which the data from a new conversion are written in the ADC\_DR register, the overrun event duly occurs (which results in the loss of either current or new data) but the overrun flag (OVR) may stay low.

#### Workaround

Clear the EOC flag through ADC\_DR register read operation or by software within less than one ADC conversion cycle period from the last conversion cycle end, so as to avoid the coincidence with the new conversion cycle end.

### 2.4.2 ADEN bit cannot be set immediately after the ADC calibration

#### Description

At the end of the ADC calibration, an internal reset of ADEN bit occurs four ADC clock cycles after the ADCAL bit is cleared by hardware. As a consequence, if the ADEN bit is set within those four ADC clock cycles, it is reset shortly after by the calibration logic and the ADC remains disabled.

#### Workaround

1. Keep setting the ADEN bit until the ADRDY flag goes high.
2. After the ADCAL is cleared, wait for a minimum of four ADC clock cycles before setting the ADEN bit.

## 2.5 IWDG

### 2.5.1 RVU, PVU and WVU flags are not reset in STOP mode

#### Description

The RVU, PVU and WVU flags of the IWDG\_SR register are set by hardware after a write access to the IWDG\_RLR and the IWDG\_PR registers, respectively. If the Stop mode is entered immediately after the write access, the RVU, PVU and WVU flags are not reset by hardware. Before performing a second write operation to the IWDG\_RLR or the IWDG\_PR register, the application software must wait for the RVU, PVU and WVU flags to be reset. However, since the RVU/PVU/WPU bit is not reset after exiting the Stop mode, the software goes into an infinite loop and the independent watchdog (IWDG) generates a reset after the programmed timeout period.

#### Workaround

Wait until the RVU, PVU and WVU flags of the IWDG\_SR register are reset, before entering the Stop mode.



## 2.5.2 RVU, PVU and WVU flags are not reset with low-frequency APB

### Description

The RVU, PVU and WVU flags of the IWDG\_SR register are set by hardware after a write access to the IWDG\_RLR and the IWDG\_PR registers, respectively. If the APB clock frequency is two times slower than the IWDG clock frequency, the RVU, PVU and WVU flags will never be reset by hardware.

### Workaround

None

## 2.6 RTC

### 2.6.1 Spurious tamper detection when disabling the tamper channel

#### Description

If the tamper detection is configured for detecting on the falling edge event (TAMPFLT = 00 and TAMPxTRG = 1) and if the tamper event detection is disabled when the tamper pin is at high level, a false tamper event is detected, which may result in the erasure of backup registers.

#### Workaround

The false detection of tamper event cannot be avoided. The erasure of the backup registers can be avoided by setting the TAMPxNOERASE bit before clearing the TAMPxE bit, in two separate RTC\_TAMPCR write accesses.

### 2.6.2 A tamper event preceding the tamper detect enable not detected

#### Description

When the tamper detect is enabled, set in edge detection mode (TAMPFLT[1:0]=00), and

- set to active rising edge (TAMPxTRG=0): if the tamper input is already high (tamper event already occurred) at the moment of enabling the tamper detection, the tamper event may not be detected. The probability of detection increases with the APB frequency.
- set to active falling edge (TAMPxTRG=1): if the tamper input is already low (tamper event already occurred) at the moment of enabling the tamper detection, the tamper event is not detected.

#### Workaround

The I/O state should be checked by software in the GPIO registers, after enabling the tamper detection and before writing sensitive values in the backup registers, in order to ensure that no active edge occurred before enabling the tamper event detection.

### 2.6.3 RTC calendar registers are not locked properly

#### Description

When reading the calendar registers with BYPSHAD = 0, the RTC\_TR and RTC\_DR registers may not be locked after reading the RTC\_SSR register. This happens if the read operation is initiated one APB clock period before the shadow registers are updated. This can result in a non-consistency of the three registers. Similarly, the RTC\_DR register can be updated after reading the RTC\_TR register instead of being locked.

#### Workaround

Apply one of the following measures:

- use BYPSHAD = 1 mode (bypass shadow registers), or
- if BYPSHAD = 0, read SSR again after reading SSR/TR/DR to confirm that SSR is still the same, otherwise read the values again.

### 2.6.4 RTC interrupt can be masked by another RTC interrupt

#### Description

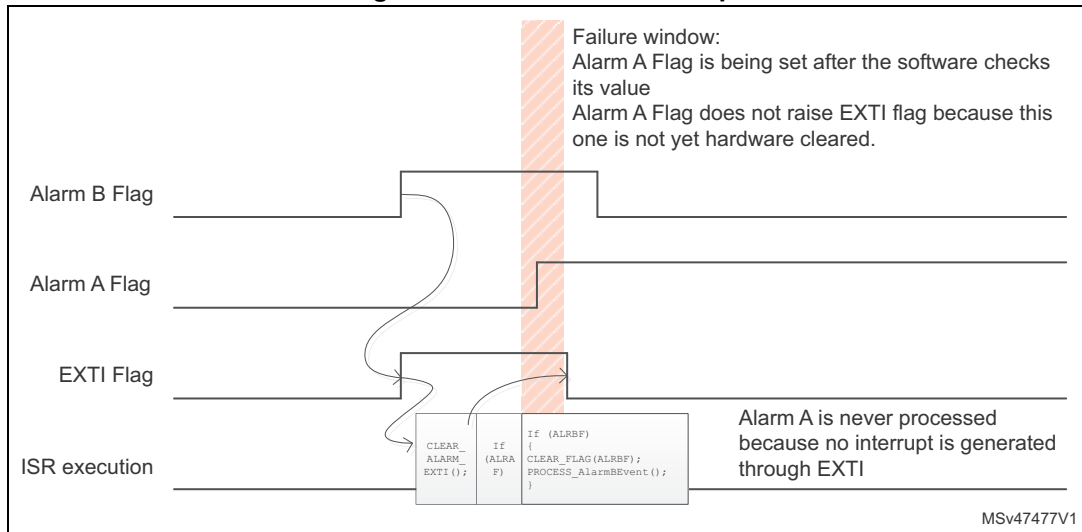
One RTC interrupt can mask another RTC interrupt if both share the same EXTI configurable line, such as the RTC Alarm A and Alarm B, of which the event flags are OR-ed to the same EXTI line (refer to the **EXTI line connections** table in the **Extended interrupt and event controller (EXTI)** section of the reference manual).

The following code example and figure illustrate the failure mechanism: The Alarm A event is lost (fails to generate interrupt) as it occurs in the failure window, that is, after checking the Alarm A event flag but before the effective clear of the EXTI interrupt flag by hardware. The effective clear of the EXTI interrupt flag is delayed with respect to the software instruction to clear it.

Alarm interrupt service routine:

```
void RTC_Alarm_IRQHandler(void)
{
    CLEAR_ALARM_EXTI(); /* Clear the EXTI line flag for RTC alarms*/
    If(ALRAF) /* Check if Alarm A triggered ISR */
    {
        CLEAR_FLAG(ALRAF); /* Clear the Alarm A interrupt pending bit */
        PROCESS_AlarmAEvent(); /* Process Alarm A event */
    }
    If(ALRBF) /* Check if Alarm B triggered ISR */
    {
        CLEAR_FLAG(ALRBF); /* Clear the Alarm B interrupt pending bit */
        PROCESS_AlarmBEvent(); /* Process Alarm B event */
    }
}
```

Figure 1. Masked RTC interrupt



**Workaround**

In the interrupt service routine, apply three consecutive event flag checks - source one, source two, and source one again, as in the following code example:

```
void RTC_Alarm_IRQHandler(void)
{
    CLEAR_ALARM_EXTI(); /* Clear the EXTI's line Flag for RTC Alarm */
    If(ALRAF) /* Check if AlarmA triggered ISR */
    {
        CLEAR_FLAG(ALRAF); /* Clear the AlarmA interrupt pending bit */
        PROCESS_AlarmAEvent(); /* Process AlarmA Event */
    }
    If(ALRBF) /* Check if AlarmB triggered ISR */
    {
        CLEAR_FLAG(ALRBF); /* Clear the AlarmB interrupt pending bit */
        PROCESS_AlarmBEvent(); /* Process AlarmB Event */
    }
    If(ALRAF) /* Check if AlarmA triggered ISR */
    {
        CLEAR_FLAG(ALRAF); /* Clear the AlarmA interrupt pending bit */
        PROCESS_AlarmAEvent(); /* Process AlarmA Event */
    }
}
```

## 2.7 I2C

### 2.7.1 Wrong data sampling when data setup time ( $t_{\text{SU;DAT}}$ ) is shorter than one I2C kernel clock period

#### Description

The I<sup>2</sup>C-bus specification and user manual specify a minimum data setup time ( $t_{\text{SU;DAT}}$ ) as:

- 250 ns in Standard mode
- 100 ns in Fast mode
- 50 ns in Fast mode Plus

The MCU does not correctly sample the I<sup>2</sup>C-bus SDA line when  $t_{\text{SU;DAT}}$  is smaller than one I2C kernel clock (I<sup>2</sup>C-bus peripheral clock) period: the previous SDA value is sampled instead of the current one. This can result in a wrong receipt of slave address, data byte, or acknowledge bit.

#### Workaround

Increase the I2C kernel clock frequency to get I2C kernel clock period within the transmitter minimum data setup time. Alternatively, increase transmitter's minimum data setup time. If the transmitter setup time minimum value corresponds to the minimum value provided in the I<sup>2</sup>C-bus standard, the minimum I2CCLK frequencies are as follows:

- In Standard mode, if the transmitter minimum setup time is 250 ns, the I2CCLK frequency must be at least 4 MHz.
- In Fast mode, if the transmitter minimum setup time is 100 ns, the I2CCLK frequency must be at least 10 MHz.
- In Fast-mode Plus, if the transmitter minimum setup time is 50 ns, the I2CCLK frequency must be at least 20 MHz.

### 2.7.2 Spurious bus error detection in master mode

#### Description

In master mode, a bus error can be detected spuriously, with the consequence of setting the BERR flag of the I2C\_SR register and generating bus error interrupt if such interrupt is enabled. Detection of bus error has no effect on the I<sup>2</sup>C-bus transfer in master mode and any such transfer continues normally.

#### Workaround

If a bus error interrupt is generated in master mode, the BERR flag must be cleared by software. No other action is required and the ongoing transfer can be handled normally.

### 2.7.3 10-bit slave mode: wrong direction bit value upon Read header receipt

#### Description

Under specific conditions, the transfer direction bit DIR (bit 16 of status register I2C\_ISR) remains low upon receipt of 10-bit addressing Read header, while normally it should be set

high. Nevertheless, I2C operates correctly in slave transmission mode, and data can be sent using the TXIS flag.

The failure described occurs when the following conditions are all met:

- I2C is configured in 10-bit addressing mode (OA1MODE is set in the I2C\_OAR1 register).
- High LSBs of the slave address are equal to the 10-bit addressing Read header value (that is, OA1[7:3] = 11110, OA1[2] = OA1[9], OA1[1] = OA1[8], and OA1[0] = 1, in the I2C\_OAR1 register).
- I2C receives 10-bit addressing Read header (0X 1111 0XX1) after repeated START condition, to enter slave transmission mode.

### Workaround

Avoid using the following 10-bit slave addresses:

- OA1[9:0] = 0011110001
- OA1[9:0] = 0111110011
- OA1[9:0] = 1011110101
- OA1[9:0] = 1111110111

If the use of one of these slave addresses cannot be avoided, do not use the DIR bit in the firmware.

## 2.7.4 10-bit combined with 7-bit slave mode: ADDCODE may indicate wrong slave address detection

### Description

Under specific conditions, the ADDCODE (address match code) bitfield in the I2C\_ISR register indicates a wrong slave address.

The failure occurs when the following conditions are all met:

- A 10-bit slave address OA1 is enabled (OA1EN = 1 and OA1MODE = 1)
- A 7-bit slave address OA2 is enabled (OA2EN = 1) and it matches the non-masked bits of OA1[7:1], that is, one of the following configurations is set:
  - OA2EN = 1 and OA2MSK = 0 and OA1[7:1] = OA2[7:1]
  - OA2EN = 1 and OA2MSK = 1 and OA1[7:2] = OA2[7:2]
  - OA2EN = 1 and OA2MSK = 2 and OA1[7:3] = OA2[7:3]
  - OA2EN = 1 and OA2MSK = 3 and OA1[7:4] = OA2[7:4]
  - OA2EN = 1 and OA2MSK = 4 and OA1[7:5] = OA2[7:5]
  - OA2EN = 1 and OA2MSK = 5 and OA1[7:6] = OA2[7:6]
  - OA2EN = 1 and OA2MSK = 6 and OA1[7] = OA2[7]
  - OA2EN = 1 and OA2MSK = 7
  - GCEN = 1 and OA1[7:1] = 0000000
  - ALERTEN = 1 and OA1[7:1] = 0001100
  - SMBDEN = 1 and OA1[7:1] = 1100001
  - SMBHEN = 1 and OA1[7:1] = 0001000
- The MCU is addressed by a bus master with its 10-bit slave address OA1.

Upon the address receipt, the ADDCODE value is OA1[7:1] equal to the 7-bit slave address, instead of 0b11110 & OA1[9:8].

#### Workaround

None. If several slave addresses are enabled, mixing 10-bit and 7-bit addresses, the OA1 [7:1] part of the 10-bit slave address must be different than the 7-bit slave address.

### 2.7.5 Wakeup frames may not wake up the MCU when Stop mode entry follows I2C enabling

#### Description

If I2C is enabled (PE = 1) and wakeup from Stop mode is enabled in I2C (WUPEN = 1) while a transfer occurs on the I<sup>2</sup>C-bus and Stop mode is entered during the same transfer while SCL = 0, I2C is not able to detect the following START condition. As a consequence, the MCU does not wake up from Stop mode when it is addressed on the I<sup>2</sup>C-bus and it does not acknowledge the receipt of the address.

#### Workaround

After enabling I2C (by setting PE to 1), do not enter Stop mode until any I<sup>2</sup>C-bus transaction in progress ends.

### 2.7.6 Wakeup frame may not wake up the MCU from Stop mode if $t_{HD;STA}$ is close to I2C kernel clock startup time

#### Description

Under specific conditions and if the START condition hold time  $t_{HD;STA}$  is very close to the startup time of the internal oscillator selected for I2C kernel clock, I2C is not able to detect the address match and, as a consequence, to wake up the MCU from Stop mode.

The failure described occurs when one of the following conditions is met:

1. Timeout detection is enabled (TIMOUTEN = 1 or TEXTEN = 1) and the frame before the wakeup frame is finished abnormally due to I2C timeout detection (TIMOUT = 1).
2. Slave arbitration is lost during the frame preceding the wakeup frame (ARLO = 1).
3. The MCU enters Stop mode while another slave is addressed, after the address phase and before STOP condition (BUSY = 1).
4. The MCU is in Stop mode and another slave is addressed before the MCU itself is addressed.

*Note:* The conditions 2, 3 and 4 can only occur in a multi-slave network.

In Stop mode, the internal oscillator selected for I2C kernel clock is switched on by I2C when START condition is detected. The I2C kernel clock is then used to receive the address. The internal oscillator is switched off upon the address receipt if the address received does not match the own slave address. If one of the conditions listed is met and if the SCL falling edge following the START condition occurs within the first cycle of the I2C kernel clock, the address is received incorrectly and the address match wakeup interrupt is not generated.

#### Workaround

None at MCU level.

Upon non-acknowledge by the MCU of a wakeup frame, the I<sup>2</sup>C-bus master with programmable START condition hold time can set that hold time such that it exceeds one MCU internal oscillator period, then resend the wakeup frame.

### 2.7.7 Wrong behavior in Stop mode when wakeup from Stop mode is disabled in I2C

#### Description

If the wakeup from Stop mode by I2C is disabled (WUPEN = 0), the correct use of the I2C peripheral is to disable it (PE = 0) before entering Stop mode, and re-enable it when back in Run mode.

Some reference manual revisions may omit this information.

Failure to respect the above while the MCU operating as slave or as master in multi-master topology enters Stop mode during a transfer ongoing on the I<sup>2</sup>C-bus may lead to the following:

1. BUSY flag is wrongly set when the MCU exits Stop mode. This prevents from initiating a transfer in master mode, as the START condition cannot be sent when BUSY is set.
2. If clock stretching is enabled (NOSTRETCH = 0), the SCL line is pulled low by I2C and the transfer stalled as long as the MCU remains in Stop mode.  
The occurrence of such condition depends on the timing configuration, peripheral clock frequency, and I<sup>2</sup>C-bus frequency.

This is a description inaccuracy issue rather than a product limitation.

#### Workaround

No application workaround is required.

### 2.7.8 10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave

#### Description

An I<sup>2</sup>C-bus master generates STOP condition upon non-acknowledge of I<sup>2</sup>C address that it sends. This applies to 7-bit address as well as to each byte of 10-bit address.

When the MCU set as I<sup>2</sup>C-bus master transmits a 10-bit address of which the first byte (5-bit header + 2 MSBs of the address + direction bit) is not acknowledged, the MCU duly generates STOP condition but it then cannot start any new I<sup>2</sup>C-bus transfer. In this spurious state, the NACKF flag of the I2C\_ISR register and the START bit of the I2C\_CR2 register are both set, while the START bit should normally be cleared.

#### Workaround

In 10-bit-address master mode, if both NACKF flag and START bit get simultaneously set, proceed as follows:

1. Wait for the STOP condition detection (STOPF = 1 in I2C\_ISR register).
2. Disable the I2C peripheral.
3. Wait for a minimum of three APB cycles.
4. Enable the I2C peripheral again.

## 2.7.9 Last-received byte loss in reload mode

### Description

If in master receiver mode or slave receive mode with SBC = 1 the following conditions are all met:

- I<sup>2</sup>C-bus stretching is enabled (NOSTRETCH = 0)
- RELOAD bit of the I2C\_CR2 register is set
- NBYTES bitfield of the I2C\_CR2 register is set to N greater than 1
- byte N is received on the I<sup>2</sup>C-bus, raising the TCR flag
- N - 1 byte is not yet read out from the data register at the instant TCR is raised,

then the SCL line is pulled low (I<sup>2</sup>C-bus clock stretching) and the transfer of the byte N from the shift register to the data register inhibited until the byte N-1 is read and NBYTES bitfield reloaded with a new value, the latter of which also clears the TCR flag. As a consequence, the software cannot get the byte N and use its content before setting the new value into the NBYTES field.

The last-received data is definitively lost (never transferred from the shift register to the data register) if the data N - 1 is read within four APB clock cycles preceding the receipt of the last data bit of byte N and thus the TCR flag raising. Refer to the product reference manual or datasheet for the I2C implementation table.

### Workaround

- In slave mode with SBC = 1, use the reload mode with NBYTES = 1.
- In master receiver mode, if the number of bytes to transfer is greater than 255 bytes, do not use the reload mode. Instead, split the transfer into sections not exceeding 255 bytes and separate them with repeated START conditions.
- Make sure, for example through the use of DMA, that the byte N - 1 is always read earlier than four APB clock cycles before the receipt of the last data bit of byte N and thus the TCR flag raising.

The last workaround in the list must be evaluated carefully for each application as the timing depends on factors such as the bus speed, interrupt management, software processing latencies, and DMA channel priority.



## 2.7.10 Spurious master transfer upon own slave address match

### Description

When the device is configured to operate at the same time as master and slave (in a multi-master I<sup>2</sup>C-bus application), a spurious master transfer may occur under the following condition:

- Another master on the bus is in process of sending the slave address of the device (the bus is busy).
- The device initiates a master transfer by writing the I2C\_CR2 register with its START bit set before the slave address match event (the ADDR flag set in the I2C\_ISR register) occurs.
- After the ADDR flag is set:
  - the device does not write I2C\_CR2 before clearing the ADDR flag, or
  - the device writes I2C\_CR2 earlier than three I2C kernel clock cycles before clearing the ADDR flag

In these circumstances, even though the START bit is automatically cleared by the circuitry handling the ADDR flag, the device spuriously proceeds to the master transfer as soon as the bus becomes free. The transfer configuration depends on the content of the I2C\_CR2 register when the master transfer starts. Moreover, if the I2C\_CR2 is written less than three kernel clocks before the ADDR flag is cleared, the I2C peripheral may fall into an unpredictable state.

### Workaround

Upon the address match event (ADDR flag set), apply the following sequence.

Normal mode (SBC = 0):

1. Set the ADDRCONF bit.
2. Before Stop condition occurs on the bus, write I2C\_CR2 with the START bit low.

Slave byte control mode (SBC = 1):

1. Write I2C\_CR2 with the slave transfer configuration and the START bit low.
2. Wait for longer than three I2C kernel clock cycles.
3. Set the ADDRCONF bit.
4. Before Stop condition occurs on the bus, write I2C\_CR2 again with its current value.

The time for the software application to write the I2C\_CR2 register before the Stop condition is limited, as the clock stretching (if enabled), is aborted when clearing the ADDR flag.

Polling the BUSY flag before requesting the master transfer is not a reliable workaround as the bus may become busy between the BUSY flag check and the write into the I2C\_CR2 register with the START bit set.

## 2.8 USART

### 2.8.1 Non-compliant sampling for NACK signal from smartcard

#### Description

According to ISO/IEC 7816-3 standard, when a character parity error is detected, the receiver must assert a NACK signal, by pulling the transmit line low for one ETU period, at 10.3 to 10.7 ETU after the character START bit falling edge. The transmitter is expected to sample the line for NACK (for low level) from 10.8 to 11.2 ETU after the character START bit falling edge.

Instead, the USART peripheral in Smartcard mode samples the transmit line for NACK from 10.3 to 10.7 ETU after the character START bit falling edge. This is unlikely to cause issues with receivers (smartcards) that respect the ISO/IEC 7816-3 standard. However, it may cause issues with respect to certification.

#### Workaround

None.

### 2.8.2 Break request preventing TC flag from being set

#### Description

After the end of transmission of data (D1), the transmission complete (TC) flag is not set when the following condition is met:

- CTS hardware flow control is enabled
- D1 transmission is in progress
- a break transfer is requested before the end of D1 transfer
- nCTS is de-asserted before the end of D1 transfer

As a consequence, an application relying on the TC flag fails to detect the end of data transfer.

#### Workaround

In the application, only allow break request after the TC flag is set.

### 2.8.3 RTS is active while RE = 0 or UE = 0

#### Description

The RTS line is driven low as soon as RTSE bit is set, even if the USART is disabled (UE = 0) or the receiver is disabled (RE = 0), that is, not ready to receive data.

#### Workaround

Upon setting the UE and RE bits, configure the I/O used for RTS into alternate function.

## 2.8.4 Consistency not checked in mode 1 of automatic baud rate detection

### Description

In mode 1 (ABRMOD = 01) of automatic baud rate detection, the Start bit then the first data bit duration is measured. If either single value measured is within an allowed range, the baud rate detection ends with success, even if the two values are inconsistent. As a consequence, the automatic baud rate detection result in mode 1 is reliable with regular input frames but not with abnormal frames.

### Workaround

None

## 2.8.5 Framing error (FE) flag low upon automatic baud rate detection error

### Description

When the ABRE flag is set to indicate an error of automatic baud rate detection, the framing error flag FE remains low although it should go high.

### Workaround

Poll exclusively the ABRE flag when checking for automatic baud rate error.

## 2.8.6 Communication parameters reprogramming after ATR in Smartcard mode when SCLK is used to clock the card

### Description

If the USART is used in Smartcard mode and the card cannot use the default communication parameters after Answer To Reset and doesn't support clock stop, it is not possible to use SCLK to clock the card. This is due to the fact that the USART and its clock output must be disabled while reprogramming some of the parameters.

### Workaround

Use another clock source to clock the card (e.g. a timer output programmed to the desired clock frequency).

## 2.8.7 Last byte written in TDR might not be transmitted if TE is cleared just after writing in TDR

### Description

If the USART clock source is slow (for example LSE) and TE bit is cleared immediately after the last write to TDR, the last byte will probably not be transmitted.

### Workarounds

1. Wait until TXE flag is set before clearing TE bit
2. Wait until TC flag is set before clearing TE bit

## 2.9 SPI/I2S

### 2.9.1 BSY bit may stay high when SPI is disabled

#### Description

The BSY flag may remain high upon disabling the SPI while operating in:

- master transmit mode and the TXE flag is low (data register full).
- master receive-only mode (simplex receive or half-duplex bidirectional receive phase) and an SCK strobing edge has not occurred since the transition of the RXNE flag from low to high.
- slave mode and NSS signal is removed during the communication.

#### Workaround

When the SPI operates in:

- master transmit mode, disable the SPI when TXE = 1 and BSY = 0.
- master receive-only mode, ignore the BSY flag.
- slave mode, do not remove the NSS signal during the communication.

### 2.9.2 BSY bit may stay high at the end of data transfer in slave mode

#### Description

BSY flag may sporadically remain high at the end of a data transfer in slave mode. This occurs upon coincidence of internal CPU clock and external SCK clock provided by master.

In such an event, if the software only relies on BSY flag to detect the end of SPI slave data transaction (for example to enter low-power mode or to change data line direction in half-duplex bidirectional mode), the detection fails.

As a conclusion, the BSY flag is unreliable for detecting the end of data transactions.

#### Workaround

Depending on SPI operating mode, use the following means for detecting the end of transaction:

- When NSS hardware management is applied and NSS signal is provided by master, use NSS flag.
- In SPI receiving mode, use the corresponding RXNE event flag.
- In SPI transmit-only mode, use the BSY flag in conjunction with a timeout expiry event. Set the timeout such as to exceed the expected duration of the last data frame and start it upon TXE event that occurs with the second bit of the last data frame. The end of the transaction corresponds to either the BSY flag becoming low or the timeout expiry, whichever happens first.

Prefer one of the first two measures to the third as they are simpler and less constraining.

Alternatively, apply the following sequence to ensure reliable operation of the BSY flag in SPI transmit mode:

1. Write last data to data register
2. Poll the TXE flag until it becomes high, which occurs with the second bit of the data frame transfer
3. Disable SPI by clearing the SPE bit mandatorily before the end of the frame transfer
4. Poll the BSY bit until it becomes low, which signals the end of transfer

*Note: The alternative method can only be used with relatively fast CPU speeds versus relatively slow SPI clocks or/and long last data frames. The faster is the software execution, the shorter can be the duration of the last data frame.*

### 2.9.3 CRC error in SPI slave mode if internal NSS changes before CRC transfer

#### Description

When the device is configured as SPI slave, the transition of the internal NSS signal after the CRCNEXT flag is set may result in wrong CRC value computed by the device and, as a consequence, in a CRC error. As a consequence, the NSS pulse mode cannot be used along with the CRC function.

#### Workaround

Prevent the internal NSS signal from changing in the critical period, by configuring the device to software NSS control, if the SPI master pulses the NSS (for example in NSS pulse mode).

### 2.9.4 SPI CRC corruption upon DMA transaction completion by another peripheral

#### Description

When the following conditions are all met:

- CRC function for the SPI is enabled
- SPI transaction managed by software (as opposed to DMA) is ongoing and CRCNEXT flag set
- another peripheral using the DMA channel on which the SPI is mapped completes a DMA transfer,

the CRCNEXT bit is unexpectedly cleared and the SPI CRC calculation may be corrupted, setting the CRC error flag.

#### Workaround

Ensure that the DMA channel on which the SPI is mapped is not concurrently in use by another peripheral.

### 2.9.5 Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback

#### Description

In receive transaction, in both I<sup>2</sup>S and SPI master modes, the last bit of the transacted frame is not captured when signal provided by internal feedback loop from the SCK pin exceeds a

critical delay. The lastly transacted bit of the stored data then keeps value from the previously received pattern. As a consequence, the last receive data bit may be wrong and/or the CRCERR flag can be unduly asserted in the SPI mode if any data under check sum and/or the CRC pattern is wrongly captured.

In SPI mode, data are synchronous with the APB clock. A delay of up to two APB clock periods can thus be tolerated for the internal feedback delay. The I<sup>2</sup>S mode is more sensitive than the SPI mode, and in particular when I2S prescaler is set to divide by an odd number and APB clock frequency is half the system clock frequency. In this case, margin of the internal feedback delay is lower than 1.5 APB clock period.

Main factors contributing to the delay increase are low V<sub>DD</sub> level, high temperature, high SCK pin capacitive load and low SCK I/O output speed. The SPI communication speed has no impact.

The following table gives the maximum allowable APB frequency versus GPIOx\_OSPEEDR output speed bitfield setting for the SCK pin, at 30pF of capacitive load. The operation is safe up to that frequency.

**Table 5. Maximum allowable APB frequency at 30pF load**

GPIOx_OSPEEDR [1:0] for SCK pin	Max. APB frequency for SPI mode [MHz]	Max. APB frequency for I <sup>2</sup> S mode [MHz]
11 (high)	48	48
01 (medium)	36	36
x0 (low)	28	20

### Workaround

The following measures can be adopted, jointly or individually:

- Decrease the APB clock speed.
- Configure the I/O pad of the SCK pin to higher speed.

## 2.9.6 Packing mode limitation at reception

**Table 6. Maximum allowable APB frequency at 30pF load**

GPIOx_OSPEEDR [1:0] for SCK pin	Max. APB frequency for SPI mode [MHz]
11 (high)	48
01 (medium)	36
x0 (low)	28

### Description

When the SPI is configured to short data frame, the use of packing mode on the reception side may result in spurious RXNE event generating an interrupt or a DMA request. As a consequence, the FIFO pointers on the reception FIFO become misaligned, which leads to inconsistent data read back by software.

If the packing mode is used during data reception, the FIFO reception threshold must be set to 16 bits. At that condition, when a read operation (half-word to read two data bytes in one APB access) takes place while the RXFIFO level is equal to 3/4 (new data byte arrives before the two first ones are read), the 16-bit read decreases the RXFIFO level to 1/4. The RXNE flag is not cleared and a new request is pending to read back next two packed data bytes, although the FIFO only contains half of them. Internal read and write pointers in the RXFIFO become misaligned and the data is corrupted.

The worst case is the continuous mode without clock interruption between data transfers.

The packing runs correctly when it is applied in full-duplex master non-continuous mode, in which the SPI peripheral always transfers an even number of data bytes, then stops the data transmission for the software to read out all the data received from the RxFIFO while SPI bus is idle before any further data transaction. Such safe read cannot be fully guaranteed if SCK signal is continuous, and in particular when 16-bit read from FIFO filled to 3/4 level cannot be prevented.

### Workaround

Continuous mode: none

Non-continuous mode: slow down the SPI communication clock frequency as much as required for the application to have time to read back data such as to prevent any 16-bit RXFIFO read operation while it is filled to its critical 3/4 level.

## 2.9.7 In I<sup>2</sup>S slave mode, enabling I2S while WS is active causes desynchronization

### Description

In I<sup>2</sup>S slave mode, the WS signal level is used to start the communication. If the I2S peripheral is enabled while the WS line is active (low for I<sup>2</sup>S protocol, high for LSB- or MSB-justified mode), and if the master is already sending the clock, the I2S peripheral (slave) starts communicating data from the instant of its enable, which causes desynchronization between the master and the slave throughout the whole communication.

### Workaround

Enable I2S peripheral while the WS line is at:

- high level, for I<sup>2</sup>S protocol.
- low level, for LSB- or MSB-justified mode.

## 2.9.8 Data flow corruption in master receiver TI half-duplex mode

### Description

Normally, a specific way of disabling the SPI controller in master receiver simplex mode, while receiving the last byte, ensures that the SPI clock stops after the receipt of the last data bit. The specific disabling sequence is described in the reference manual.

When the following conditions are all met:

- the device operates as SPI master receiver in TI half-duplex mode (BIDIMODE = 1 and BIDIOE = 0)
- the baud rate divider is set to 2
- the data size is set to 4-bit or 5-bit,

then upon disabling the SPI controller, even through the recommended sequence, the SPI clock may spuriously stop earlier than after the last data bit. As a consequence of missing SPI clock pulses, the last data is not sent by the external slave transmitter.

When enabled back again, the master receiver operates normally, including the generation of NSS pulse to start a new data transaction.

### Workaround

Instead of applying the specific disabling sequence during the transaction of the last data to receive, only disable the SPI controller (SPE = 0) upon the last data byte receipt is completed.

Depending on the timing, this may result in sending extra SPI clock pulses, unexpected by the SPI slave, and as a consequence, in de-synchronization between the master receiver and the slave transmitter. The workaround is therefore only guaranteed effective with slaves that are able to re-synchronize with the master receiver when a new data transaction, signaled with an NSS pulse, is initiated by the master receiver.

In normal receiving mode, any CEC message with destination address different from the own address should normally be ignored and have no effect to the CEC peripheral. Instead, such a message is unduly written into the reception buffer and sets the CEC peripheral to a state in which any subsequent message with the destination address equal to the own address is rejected (NACK), although it sets RXOVR flag (because the reception buffer is considered full) and generates (if enabled) an interrupt. This failure can only occur in a multi-node CEC framework where messages with addresses other than own address can appear on the CEC line.

The listen mode operates correctly.

## 2.10 Package

### 2.10.1 Inversed date code in the package marking versus datasheet

#### Description

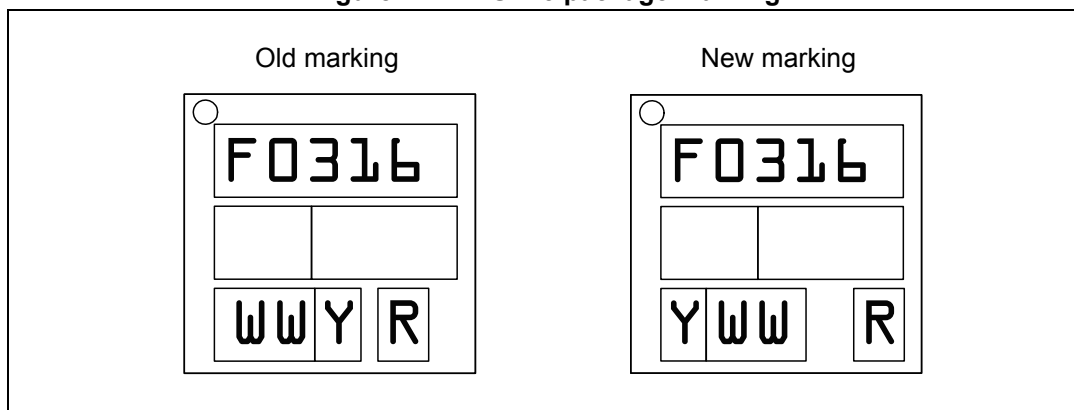
Until week 9 of 2017, the date code on the WLCSP25 package was marked in the format WWY (two digits for week followed by one digit for year), while the figure giving the example of package marking in the product datasheet was showing the inversed sequence YWW.

#### Workaround

From week 9, 2017 on, the marking on the WLCSP25 package is modified to show the date in the format YWW. [Figure 2](#) illustrates the difference between the old and the new product marking. The figure in the product datasheet was also updated to better reflect the dimensions and layout of the marking on the package.



Figure 2. WLCSP25 package marking



### 3 Revision history

**Table 7. Document revision history**

Date	Revision	Changes
12-Jun-2014	1	Initial release.
12-Oct-2016	2	<p><b>Added:</b></p> <p>USART:</p> <ul style="list-style-type: none"> <li>– Section 2.2.1: Start bit detected too soon when sampling for NACK signal from the smartcard</li> <li>– Section 2.2.2: Break request can prevent the Transmission Complete flag (TC) from being set</li> <li>– Section 2.2.3: RTS is active while RE or UE = 0</li> <li>– Section 2.2.4: Consistency not checked in mode 1 of automatic baud rate detection</li> <li>– Section 2.2.5: Framing error (FE) flag low upon automatic baud rate detection error</li> </ul> <p>GPIO:</p> <ul style="list-style-type: none"> <li>– Section 2.3.2: GPIOx locking mechanism not working properly for GPIOx_OTYPER register</li> </ul> <p>I2C:</p> <ul style="list-style-type: none"> <li>– Section 2.4.1: Wrong data sampling when data set-up time (tSU;DAT) is shorter than one I2CCLK period</li> <li>– Section 2.4.2: Spurious bus error detection in master mode</li> <li>– Section 2.4.8: 10-bit master mode: new transfer cannot be launched if first part of the address has not been acknowledged by the slave</li> </ul> <p>SPI:</p> <ul style="list-style-type: none"> <li>– Section 2.5.1: BSY bit may stay high when SPI is disabled</li> <li>– Section 2.5.2: BSY bit may stay high at the end of a data transfer in slave mode</li> <li>– Section 2.5.3: Wrong CRC transmitted in master mode with delayed SCK feedback</li> <li>– Section 2.5.4: CRC error in SPI slave mode if internal NSS changes before CRC transfer</li> <li>– Section 2.5.5: SPI CRC corrupted upon DMA transaction completion by another peripheral</li> <li>– Section 2.5.6: Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback</li> </ul> <p>RTC:</p> <ul style="list-style-type: none"> <li>– Section 2.6.1: Spurious tamper detection when disabling the tamper channel</li> <li>– Section 2.6.2: A tamper event preceding the tamper detect enable not detected</li> <li>– Section 2.6.3: RTC calendar registers are not locked properly</li> </ul>

Table 7. Document revision history (continued)

Date	Revision	Changes
12-Oct-2016	2	<p>ADC:</p> <ul style="list-style-type: none"> <li>– Section 2.7.1: Overrun flag not set if EOC reset coincides with new conversion end</li> <li>– Section 2.7.2: ADEN bit cannot be set immediately after the ADC calibration</li> </ul> <p>IWDG:</p> <ul style="list-style-type: none"> <li>– Section 2.8.1: RVU, PVU and WVU flags are not reset in STOP mode</li> <li>– Section 2.8.2: RVU, PVU and WVU flags are not reset with low-frequency APB</li> </ul> <p><b>Modified:</b></p> <ul style="list-style-type: none"> <li>– Document structure</li> <li>– Cover page and Table 3 organization</li> <li>– SPI: Section 2.5.7: Packing mode limitation at reception</li> </ul> <p><b>Removed:</b></p> <ul style="list-style-type: none"> <li>– Appendix A (package marking drawings are now available in the data sheet)</li> </ul>
15-May-2017	3	<p><b>Added:</b></p> <ul style="list-style-type: none"> <li>– <a href="#">Section 2.15.1: Inversed date code in the package marking versus datasheet</a></li> <li>– REV_ID on the cover page</li> </ul> <p><b>Modified:</b></p> <ul style="list-style-type: none"> <li>– Organization of the cover page</li> <li>– <a href="#">Section 2.10.3: RTS is active while RE or UE = 0</a> - “RTS” replaced “nRTS”</li> <li>– The order of functions in <a href="#">Table 3: Summary of device limitations OneStatusColumn</a> and <a href="#">Section 2: Description of device limitations</a> aligned with the corresponding reference manual</li> </ul>
09-May-2018	4	<p><b>Added:</b></p> <ul style="list-style-type: none"> <li>– REV_ID bitfield information on the cover page</li> <li>– <a href="#">Table 4: Summary of documentation errata</a></li> <li>– information on workaround qualifiers in <a href="#">Section 1: Summary of device errata</a></li> <li>– <a href="#">Section 2.3.1: DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear</a></li> <li>– <a href="#">Section 2.6.4: RTC interrupt can be masked by another RTC interrupt</a></li> <li>– <a href="#">Section 2.3.2: Byte and half-word accesses not supported</a></li> <li>– <a href="#">Section 2.7.9: Last-received byte loss in reload mode</a></li> <li>– <a href="#">Section 2.7.10: Spurious master transfer upon own slave address match</a></li> </ul>

Table 7. Document revision history (continued)

Date	Revision	Changes
09-May-2018	4	<p><b>Modified:</b></p> <ul style="list-style-type: none"> <li>– minor modifications in titles and/or text of existing limitation descriptors in <i>I2C</i>, <i>SPI/I2S</i> and <i>USART</i> sections</li> <li>– workaround of the limitation in <i>Section 2.9.7: In I<sup>2</sup>S slave mode, enabling I2S while WS is active causes desynchronization</i> re-qualified to “P”</li> <li>– workaround description in <i>Section 2.7.1: Wrong data sampling when data setup time (t<sub>SU, DAT</sub>) is shorter than one I2C kernel clock period</i></li> <li>– limitation in <i>Section 2.7.7: Wrong behavior in Stop mode when wakeup from Stop mode is disabled in I2C</i> qualified as documentation erratum and re-written</li> <li>– document ID in the footer of all pages to ES0236</li> <li>– renaming of introductory section on the cover page</li> </ul> <p><b>Removed:</b></p> <ul style="list-style-type: none"> <li>– redundant limitation “Wrong CRC transmitted in master mode with delay on SCK feedback” in <i>SPI/I2S</i> section, kept in previous versions for historical reasons.</li> </ul>

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2018 STMicroelectronics – All rights reserved