

---

**Device limitations for STM8AF6366 automotive MCUs featuring up to 32 Kbytes of Flash program memory**

---

**Applicability**

This document applies to the STM8AF6366 devices.

It gives a summary and a description of the device errata, with respect to the device datasheet and reference manual RM0016.

Deviation of the real device behavior from the intended device behavior is considered to be a device limitation. Deviation of the description in the reference manual or the datasheet from the intended device behavior is considered to be a documentation erratum. The term “errata” applies both to limitations and documentation errata.

**Table 1. Device variants**

Reference	silicon revision codes
	Device marking
STM8AF6366	V

# Contents

<b>1</b>	<b>Summary of device errata</b>	<b>6</b>
<b>2</b>	<b>Description of device errata</b>	<b>8</b>
2.1	STM8A core	8
2.1.1	Activation level (AL) bit not functional in Halt mode	8
2.1.2	JRIL and JRIH instructions not available	8
2.1.3	Main CPU execution is not resumed after an ISR resets the AL bit	8
2.1.4	Unexpected DIV/DIVW instruction result in ISR	9
2.1.5	Wait for event instruction (WFE) not available	9
2.1.6	Interrupt service routine (ISR) executed with priority of main process	10
2.2	System limitations	10
2.2.1	HSI RC oscillator cannot be switched off in run mode	10
2.2.2	LSI oscillator remains on in Active-halt mode when the AWU unit uses the HSE as input clock	10
2.2.3	RAM content at address 0x000 not preserved after RESET	10
2.2.4	RAM modified after reset by embedded bootloader	11
2.2.5	Flash/EEPROM memory is read incorrectly after wakeup from power down mode	11
2.2.6	$V_{DD}$ rise-time rate for $100\text{mV} < V_{DD} < 1\text{V}$	12
2.3	Timer peripheral limitations	12
2.3.1	Corruption of read sequence for the 16-bit counter registers	12
2.4	LINUART peripheral limitations	13
2.4.1	PE flag cannot be cleared during the reception of the first half of Stop bit	13
2.4.2	PE testing issue in UART mode	13
2.4.3	LIN mode: LIN header error when automatic resynchronization is enabled	13
2.4.4	LIN mode: framing error with data byte 0x00	14
2.4.5	LIN mode: framing error when receiving an identifier (ID)	14
2.4.6	LIN mode: parity error when receiving an identifier (ID)	14
2.4.7	LIN mode: OR flag not correctly set in LIN Master mode	14
2.5	I2C peripheral limitations	15
2.5.1	I2C event management	15
2.5.2	Corrupted last received data in I2C Master Receiver mode	15

---

2.5.3	Wrong behavior of I2C peripheral in Master mode after misplaced STOP .....	16
2.5.4	Violation of I2C “setup time for repeated START condition” parameter ..	16
2.5.5	In I2C slave “NOSTRETCH” mode, underrun errors may not be detected and may generate bus errors .....	17
2.5.6	I2C pulse missed .....	18
2.6	SPI peripheral limitations .....	18
2.6.1	Last bit too short if SPI is disabled during communication .....	18
2.6.2	Busy flag is not reliable when the SPI is a master simplex receive-only mode .....	19
2.6.3	CRC may be corrupted by SPI configuration or other bus transfers ...	19
2.6.4	Anticipated communication upon SPI transit from slave receiver to master .....	20
2.6.5	BSY bit may stay high at the end of data transfer in slave mode .....	20
2.7	ADC peripheral limitation .....	21
2.7.1	EOC interrupt triggered when AWDIE and EOCIE set to 1 .....	21
<b>3</b>	<b>Important security notice .....</b>	<b>22</b>
<b>4</b>	<b>Revision history .....</b>	<b>23</b>

## List of tables

Table 1.	Device variants . . . . .	1
Table 2.	Summary of device limitations . . . . .	6
Table 3.	V <sub>DD</sub> rise-time and fall-time rates . . . . .	12
Table 4.	Document revision history . . . . .	23

# List of figures

Figure 1. 16-bit read sequence for the counter (TIMx\_CNTR) . . . . . 12

# 1 Summary of device errata

The following table gives a quick reference to the STM8AF6366 device limitations and their status:

A = limitation present, workaround available

N = limitation present, no workaround available

P = limitation present, partial workaround available

“-” = limitation absent

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 2. Summary of device limitations**

Function	Section	Limitation	Rev V
STM8A core	2.1.1	Activation level (AL) bit not functional in Halt mode	N
	2.1.2	JRIL and JRJH instructions not available	N
	2.1.3	Main CPU execution is not resumed after an ISR resets the AL bit	A
	2.1.4	Unexpected DIV/DIVW instruction result in ISR	A
	2.1.5	Wait for event instruction (WFE) not available	N
	2.1.6	Interrupt service routine (ISR) executed with priority of main process	A
System limitations	2.2.1	HSI RC oscillator cannot be switched off in run mode	N
	2.2.2	LSI oscillator remains on in Active-halt mode when the AWU unit uses the HSE as input clock	N
	2.2.3	RAM content at address 0x000 not preserved after RESET	A
	2.2.4	RAM modified after reset by embedded bootloader	-
	2.2.5	Flash/EEPROM memory is read incorrectly after wakeup from power down mode	A
	2.2.6	$V_{DD}$ rise-time rate for $100\text{mV} < V_{DD} < 1\text{V}$	-
Timer peripheral limitations	2.3.1	Corruption of read sequence for the 16-bit counter registers	A
LINUART peripheral limitations	2.4.1	PE flag cannot be cleared during the reception of the first half of Stop bit	A
	2.4.2	PE testing issue in UART mode	N
	2.4.3	LIN mode: LIN header error when automatic resynchronization is enabled	-
	2.4.4	LIN mode: framing error with data byte 0x00	N
	2.4.5	LIN mode: framing error when receiving an identifier (ID)	N
	2.4.6	LIN mode: parity error when receiving an identifier (ID)	N
	2.4.7	LIN mode: OR flag not correctly set in LIN Master mode	N

Table 2. Summary of device limitations (continued)

Function	Section	Limitation	Rev V
<i>I2C peripheral limitations</i>	2.5.1	<i>I2C event management</i>	A
	2.5.2	<i>Corrupted last received data in I2C Master Receiver mode</i>	A
	2.5.3	<i>Wrong behavior of I2C peripheral in Master mode after misplaced STOP</i>	A
	2.5.4	<i>Violation of I2C "setup time for repeated START condition" parameter</i>	A
	2.5.5	<i>In I2C slave "NOSTRETCH" mode, underrun errors may not be detected and may generate bus errors</i>	A
	2.5.6	<i>I2C pulse missed</i>	-
<i>SPI peripheral limitations</i>	2.6.1	<i>Last bit too short if SPI is disabled during communication</i>	A
	2.6.2	<i>Busy flag is not reliable when the SPI is a master simplex receive-only mode</i>	N
	2.6.3	<i>CRC may be corrupted by SPI configuration or other bus transfers</i>	A
	2.6.4	<i>Anticipated communication upon SPI transit from slave receiver to master</i>	A
	2.6.5	<i>BSY bit may stay high at the end of data transfer in slave mode</i>	A
<i>ADC peripheral limitation</i>	2.7.1	<i>EOC interrupt triggered when AWDIE and EOCIE set to 1</i>	N

## 2 Description of device errata

### 2.1 STM8A core

#### 2.1.1 Activation level (AL) bit not functional in Halt mode

##### Description

The AL bit is not supported in Halt mode. In particular, when the AL bit of the CFG\_GCR register is set, the CPU does not return to Halt mode after exiting an interrupt service routine (ISR). It returns to the main program and executes the next instruction after the HALT instruction. The AL bit is supported correctly in WFI mode.

##### Workaround

No workaround available.

No fix is planned for this limitation.

#### 2.1.2 JRIL and JRIH instructions not available

##### Description

JRIL (jump if port INT pin = 0) and JRIH (jump if port INT pin = 1) are not supported by the devices covered by this datasheet. These instructions perform conditional jumps: JRIL and JRIH jump if one of the external interrupt lines is low and high, respectively. JRIL is equivalent to an unconditional jump and JRIH is equivalent to a NOP.

For further details on these instructions, refer to the *STM8 CPU programming manual* (PM0044).

##### Workaround

No workaround available.

No fix is planned for this limitation.

#### 2.1.3 Main CPU execution is not resumed after an ISR resets the AL bit

##### Description

If the CPU is in wait for interrupt state and the AL bit is set, the CPU returns to wait for interrupt state after executing an ISR. To continue executing the main program, the AL bit must be reset by the ISR. When AL is reset just before exiting the ISR, the CPU may remain stalled.

##### Workaround

Reset the AL bit at least two instructions before the IRET instruction.

No fix is planned for this limitation.



## 2.1.4 Unexpected DIV/DIVW instruction result in ISR

### Description

In very specific conditions, a DIV/DIVW instruction may return a false result when executed inside an interrupt service routine (ISR). This error occurs when the DIV/DIVW instruction is interrupted and a second interrupt is generated during the execution of the IRET instruction of the first ISR. Under these conditions, the DIV/DIVW instruction executed inside the second ISR, including function calls, may return an unexpected result.

The applications that do not use the DIV/DIVW instruction within ISRs are not impacted.

### Workaround 1

If an ISR or a function called by this routine contains a division operation, the following assembly code should be added inside the ISR before the DIV/DIVW instruction:

```
push cc
pop a
and a, # $BF
push a
pop cc
```

This sequence should be placed by C compilers at the beginning of the ISR using DIV/DIVW. Refer to the compiler documentation for details on the implementation and control of automatic or manual code insertion.

### Workaround 2

To optimize the number of cycles added by workaround 1, the user can use this workaround instead. Workaround 2 can be used in applications with fixed interrupt priorities, identified at the program compilation phase:

```
push #value
pop cc
```

where bits 5 and 3 of #value have to be configured according to interrupt priority given by I1 and I0, and bit 6 kept cleared.

In this case, compiler workaround 1 has to be disabled by using compiler directives.

No fix is planned for this limitation.

## 2.1.5 Wait for event instruction (WFE) not available

### Description

The WFE instruction is not implemented in the devices covered by this datasheet. This instruction is used to synchronize the device with external computing resources. For further details on this instruction, refer to the *STM8 CPU programming manual* (PM0044).

### Workaround

No workaround available.

No fix is planned for this limitation.

## 2.1.6 Interrupt service routine (ISR) executed with priority of main process

### Description

If an interrupt is cleared or masked when the context saving has already started, the corresponding ISR is executed with the priority of the main process. The next interrupt request can interrupt execution of the service routine

### Workaround

At the beginning of the interrupt routine, change the current priority level in the CCR register by software.

## 2.2 System limitations

### 2.2.1 HSI RC oscillator cannot be switched off in run mode

#### Description

The internal 16 MHz RC oscillator cannot be switched off in run mode, even if the HSIEN bit is programmed to 0.

#### Workaround

No workaround available.

No fix is planned for this limitation.

### 2.2.2 LSI oscillator remains on in Active-halt mode when the AWU unit uses the HSE as input clock

#### Description

When the auto-wakeup unit (AWU) uses the high speed external clock (HSE) divided by the prescaler (clock source enabled by setting the CKAWUSEL option bit), the LSI RC oscillator is not switched off when the device operates in Active-halt mode with the main voltage regulator (MVR) on. This causes negligible extra power consumption compared to the total consumption of the MCU in Active-halt mode with the MVR on.

#### Workaround

No workaround available.

No fix is planned for this limitation.

### 2.2.3 RAM content at address 0x000 not preserved after RESET

#### Description

When RESET is asserted during an ongoing write to any RAM location, memory content at address 0x000 could be corrupted.

#### Workaround

Do not use location 0x000 to store critical application data.

No fix is planned for this limitation.

## 2.2.4 RAM modified after reset by embedded bootloader

### Description

After each reset, the byte located at RAM address 0x95 is modified by the embedded bootloader even if the bootloader is disabled by option byte. So the RAM content at address 0x95 is not maintained after reset. The issue is present only in device revision Y.

### Workaround

No workaround available. Do not use byte in RAM at address 0x95 to store variables which should be unchanged after device reset.

## 2.2.5 Flash/EEPROM memory is read incorrectly after wakeup from power down mode

### Description

If Flash/EEPROM memory has been put in power down mode ( $I_{DDQ}$ ), the first read access after wakeup could return incorrect content.

By default, the Flash/EEPROM memory is put in  $I_{DDQ}$  mode when the MCU enters Halt mode and depending on the FLASH\_CR1 register settings made by software, the Flash/EEPROM may be forced to  $I_{DDQ}$  mode during active halt mode.

As a consequence, the following behavior may be seen on some devices:

- After wakeup from Low power mode, with Flash memory in  $I_{DDQ}$  mode, program execution gets lost due to an incorrect read of the vector table.
- Code reads an incorrect value from Flash/EEPROM memory, when forced in  $I_{DDQ}$  mode.
- Reset could be forced by an illegal opcode execution due to incorrect read of instruction.

*Note:* The use of the watchdog helps the application to recover in case of failure.

### Workaround

Keep the Flash/EEPROM in operating mode when MCU is put in Halt mode or Active-halt mode. This is done by configuring both the HALT and AHALT bits in the FLASH\_CR1 register before executing a HALT instruction to prevent the Flash/EEPROM entering  $I_{DDQ}$  mode.

Set HALT (bit 3) to 1:

0: Flash in power-down mode when MCU is in Halt mode

1: Flash in operating mode when MCU is in Halt mode

Keep AHALT (bit 2) at 0:

0: Flash in operating mode when MCU is in Active-halt mode

1: Flash in power-down when MCU is in Active-halt mode

Refer to the datasheet for details on the impact on current consumption and wakeup time.

## 2.2.6 $V_{DD}$ rise-time rate for $100\text{mV} < V_{DD} < 1\text{V}$

### Description

The product datasheet did not specify the  $V_{DD}$  rise-time initial conditions as the  $V_{DD}$  rise-time was implicitly specified for a  $V_{DD}$  starting from 0V. Nevertheless, we observed that some very specific applications could have a  $V_{DD}$  starting from a residual voltage already above 0V and thus require that we explicitly specify these conditions.

The  $t_{VDD}$  parameter must stay below  $50\mu\text{s/V}$  when  $V_{DD}$  is rising from 100mV to 1V.

**Table 3.  $V_{DD}$  rise-time and fall-time rates**

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$t_{VDD}$	$V_{DD}$ rise-time rate	$V_{DD} < 100\text{mV}$	$2^{(1)}$	-	$\infty$	$\mu\text{s/V}$
		$100\text{mV} < V_{DD} < 1\text{V}$	$2^{(1)}$	-	$50^{(1)}$	
		$V_{DD} > 1\text{V}$	$2^{(1)}$	-	$\infty$	
	$V_{DD}$ fall-time rate	-	$2^{(1)}$	-	$\infty$	

1. Guaranteed by design, not tested in production.

### Workaround

Not applicable. Limitation has been fixed on the products covered by this document.

## 2.3 Timer peripheral limitations

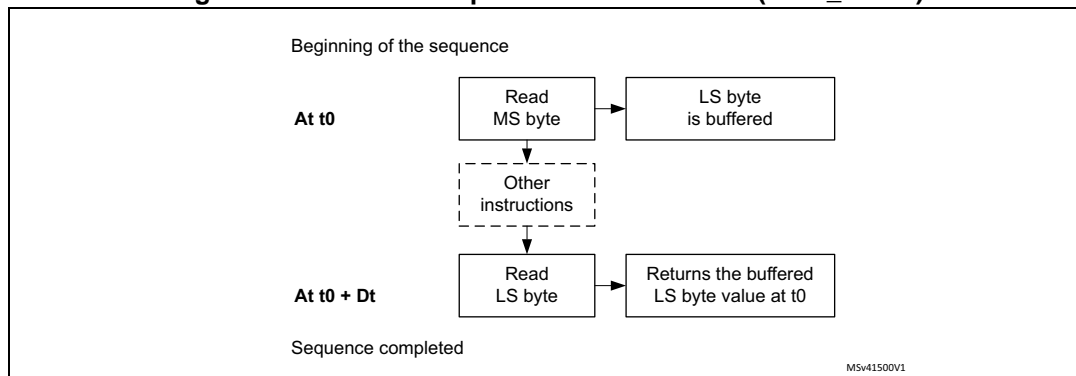
### 2.3.1 Corruption of read sequence for the 16-bit counter registers

#### Description

An 8-bit buffer is implemented for reading the 16-bit counter registers. Software must read the MS byte first, after which the LS byte value is buffered automatically (see the figure below). This buffered value remains unchanged until the 16-bit read sequence is completed.

When any multi-cycle instruction precedes the read of the LSB, the content of the buffer is lost and the second read returns the immediate content of the counter directly.

**Figure 1. 16-bit read sequence for the counter (TIMx\_CNTR)**



**Workaround**

Do not use multi-cycle instructions before reading the LSB.

No fix is planned for this limitation.

**2.4 LINUART peripheral limitations****2.4.1 PE flag cannot be cleared during the reception of the first half of Stop bit****Description**

The PE flag is set by hardware when the UART is in reception mode and a parity error (PE) occurs. This flag cannot be cleared during the first half of the Stop bit period. If the software attempts to clear the PE flag at this moment, the flag is set again by hardware, thus generating an unwanted interrupt (assuming the PIEN bit has been set in the UART\_CR1 register).

**Workaround**

1. Disable PE interrupts by setting PIEN to 0.
2. After the RXNE bit is set (received data ready to be read), poll the PE flag to check if it a parity error occurred. For example, this could be done in the RXNE interrupt service routine.

No fix is planned for this limitation.

**2.4.2 PE testing issue in UART mode****Description**

When the RXNE flag is not polled, the device is in overrun condition and the PE flag does not rise in case of a parity error. The flag rises only for the last data which have been correctly received.

**Workaround**

No workaround available.

No fix is planned for this limitation.

**2.4.3 LIN mode: LIN header error when automatic resynchronization is enabled****Description**

If UART2 is configured in LIN slave mode (LSLV bit set in UART2\_CR6 register) and the automatic resynchronization is enabled (LASE bit set in UART2\_CR6), the LHE flag may be set instead of LHDF flag when receiving a valid header.

This limitation is fixed in silicon revision X.

**Workaround**

No workaround available.

**2.4.4 LIN mode: framing error with data byte 0x00****Description**

If the UART2 interface is configured in LIN slave mode, and the active mode with break detection length is set to 11 (LBDL bit of UART2\_CR4 register set to 1), FE and RXNE flags are not set when receiving a 0x00 data byte with a framing error, followed by a recessive state. This occurs only if the dominant state length is between 9.56 and 10.56 times the baud rate.

**Workaround**

The LIN software driver can handle this exceptional case by implementing frame timeouts to comply with the LIN standard. This method has been implemented in ST LIN 2.1 driver package which passed the LIN compliance tests.

**2.4.5 LIN mode: framing error when receiving an identifier (ID)****Description**

If an ID framing error occurs when the UART2, configured in LIN mode, is in active mode, both the LHE and LHDF flags are set at the end of the LIN header with an ID framing error.

**Workaround**

The LIN software driver can handle this case by checking both LHE and LHDF flags upon header reception.

**2.4.6 LIN mode: parity error when receiving an identifier (ID)****Description**

If an ID parity error occurs, the UART2, configured in LIN mode, wakes up from mute mode and both LHE and LHDF are set at the end of the LIN header with parity error. The PE flag is also set.

**Workaround**

The LIN software driver can handle this case by checking all flags upon header reception.

No fix is planned for this limitation.

**2.4.7 LIN mode: OR flag not correctly set in LIN Master mode****Description**

When the UART operates in LIN Master mode, the OR flag is not set if an overrun condition occurs.

### Workaround

The LIN software driver can detect this case through a LIN protocol error. No fix is planned for this limitation.

## 2.5 I2C peripheral limitations

### 2.5.1 I2C event management

#### Description

As described in the I2C section of the microcontroller reference manual, the application firmware has to manage several software events before the current byte is transferred. If the EV7, EV7\_1, EV6\_1, EV6\_3, EV2, EV8, and EV3 events are not managed before the current byte is transferred, problems may occur such as receiving an extra byte, reading the same data twice, or missing data.

#### Workaround

When the EV7, EV7\_1, EV6\_1, EV6\_3, EV2, EV8, and EV3 events cannot be managed before the current byte transfer, and before the acknowledge pulse when the ACK control bit changes, it is recommended to use I<sup>2</sup>C interrupts in nested mode and to make them uninterruptible by increasing their priority to the highest priority in the application.

No fix is planned for this limitation.

### 2.5.2 Corrupted last received data in I2C Master Receiver mode

#### Conditions

In Master Receiver mode, when the communication is closed using method 2, the content of the last read data may be corrupted. The following two sequences are concerned by the limitation:

- Sequence 1: transfer sequence for master receiver when  $N = 2$ 
  - a) BTF = 1 (Data N-1 in DR and Data N in shift register)
  - b) Program STOP = 1
  - c) Read DR twice (Read Data N-1 and Data N) just after programming the STOP bit.
- Sequence 2: transfer sequence for master receiver when  $N > 2$ 
  - a) BTF = 1 (Data N-2 in DR and Data N-1 in shift register)
  - b) Program ACK = 0
  - c) Read Data N-2 in DR
  - d) Program STOP bit to 1
  - e) Read Data N-1.

#### Description

The content of the shift register (data N) is corrupted (data N is shifted 1 bit to the left) if the user software is not able to read data N-1 before the STOP condition is generated on the bus. In this case, reading data N returns a wrong value.

### Workarounds

- Workaround 1
  - Sequence 1  
When sequence 1 is used to close communication using method 2, mask all active interrupts between STOP bit programming and Read data N-1.
  - Sequence 2  
When sequence 2 is used to close communication using method 2, mask all active interrupts between Read data N-2, STOP bit programming and Read data N-1.
- Workaround 2  
Manage I2C RxNE and TxE events with interrupts of the highest priority level, so that the condition BTF = 1 never occurs.

## 2.5.3 Wrong behavior of I2C peripheral in Master mode after misplaced STOP

### Description

The I2C peripheral does not enter Master mode properly if a misplaced STOP is generated on the bus. This can happen in the following conditions:

- If a void message is received (START condition immediately followed by a STOP): the BERR (bus error) flag is not set, and the I2C peripheral is not able to send a START condition on the bus after writing to the START bit in the I2C\_CR2 register.
- In the other cases of a misplaced STOP, the BERR flag is set in the IC2\_CR2 register. If the START bit is already set in I2C\_CR2, the START condition is not correctly generated on the bus and can create bus errors.

### Workaround

In the I2C standard, it is not allowed to send a STOP before the full byte is transmitted (8 bits + acknowledge). Other derived protocols like CBUS allow it, but they are not supported by the I<sup>2</sup>C peripheral.

In case of noisy environment in which unwanted bus errors can occur, it is recommended to implement a timeout to ensure that the SB (start bit) flag is set after the START control bit is set. In case the timeout has elapsed, the peripheral must be reset by setting the SWRST bit in the I2C\_CR2 control register. The I<sup>2</sup>C peripheral should be reset in the same way if a BERR is detected while the START bit is set in I2C\_CR2.

No fix is planned for this limitation.

## 2.5.4 Violation of I2C “setup time for repeated START condition” parameter

### Description

In case of a repeated Start, the “setup time for repeated START condition” parameter (named  $t_{SU(STA)}$  in the datasheet and  $t_{SU(STA)}$  in the I2C specifications) may be slightly violated when the I2C operates in Master Standard mode at a frequency ranging from 88 to 100 kHz.  $t_{SU(STA)}$  minimum value may be 4  $\mu$ s instead of 4.7  $\mu$ s.



The issue occurs under the following conditions:

1. The I2C peripheral operates in Master Standard mode at a frequency ranging from 88 to 100 kHz (no issue in Fast mode)
2. and the SCL rise time meets one of the following conditions:
  - The slave does not stretch the clock and the SCL rise time is more than 300 ns (the issue cannot occur when the SCL rise time is less than 300 ns).
  - or the slave stretches the clock.

### Workaround

Reduce the frequency down to 88 kHz or use the I2C Fast mode if it is supported by the slave.

## 2.5.5 In I2C slave “NOSTRETCH” mode, underrun errors may not be detected and may generate bus errors

### Description

The data valid time ( $t_{VD;DAT}$ ,  $t_{VD;ACK}$ ) described by the I2C specifications may be violated as well as the maximum current data hold time ( $t_{HD;DAT}$ ) under the conditions described below. In addition, if the data register is written too late and close to the SCL rising edge, an error may be generated on the bus: SDA toggles while SCL is high. These violations cannot be detected because the OVR flag is not set (no transmit buffer underrun is detected).

This issue occurs under the following conditions:

1. The I2C peripheral operates In Slave transmit mode with clock stretching disabled (NOSTRETCH=1)
2. and the application is late to write the DR data register, but not late enough to set the OVR flag (the data register is written before the SCL rising edge).

### Workaround

If the master device supports it, use the clock stretching mechanism by programming the bit NOSTRETCH = 0 in the I2C\_CR1 register.

If the master device does not support it, ensure that the write operation to the data register is performed just after TXE or ADDR events. The user can use an interrupt on the TXE or ADDR flag and boost its priority to the higher level.

Using the “NOSTRETCH” mode with a slow I2C bus speed can prevent the application from being late to write the DR register (second condition).

*Note: The first data to be transmitted must be written into the data register after the ADDR flag is cleared, and before the next SCL rising edge, so that the time window to write the first data into the data register is less than  $t_{LOW}$ .*

*If this is not possible, a possible workaround can be the following:*

1. Clear the ADDR flag
2. Wait for the OVR flag to be set
3. Clear OVR and write the first data.

*The time window for writing the next data is then the time to transfer one byte. In that case, the master must discard the first received data.*

## 2.5.6 I2C pulse missed

### Description

When the I2C interface is used for long transmit/receive transactions, the MCU may return a NACK somewhere during the transaction instead of returning an ACK for all data. The received data may also be corrupted. In Master mode the I2C may not detect an incoming ACK. This is due to a weakness in the noise filter of the I/O pad which in certain conditions may cause the STM8 I2C to miss a pulse.

This limitation is fixed in revisions X and W.

### Workaround

Since data corruption is caused by noise generated by the CPU, CPU activity should be minimized during data reception and/or transmission. This is done by performing physical data transmission (Master mode) and reception (slave mode) in WFI state (wait for interrupt).

To allow the device to be woken up from WFI, I<sup>2</sup>C transmission and reception routines must be implemented through interrupt routines instead of polling mechanisms. Receive and transmit interrupts (received data processing) must be triggered only by the BTF bit flag (byte transfer finished) in the I2C\_SR1 register. This flag indicates that the I<sup>2</sup>C is in stretched state (data transfers are stretched on the bus).

Clock stretching must be enabled to allow data transfers from the slave to be stopped and to allow the CPU to be woken up to read the received byte.

To recover from possible errors, periodically check if the I2C does not remain in busy state for too long (BUSY bit set in I2C\_SR3 register). If so, it should be reinitialized.

Example of I<sup>2</sup>C slave code:

```
//...
//-----
void main()
{
    Init_I2C(); // init I2C to use interrupts: ITBUFEN=0, ITEVTEN=1,
    ITERREN=1
    while(1)
```

## 2.6 SPI peripheral limitations

### 2.6.1 Last bit too short if SPI is disabled during communication

#### Description

When the SPI interface operates in Master mode and the baud rate generator prescaler is equal to 2, the SPI is disabled during ongoing communications, and the data and clock output signals are switched off at the last strobing edge of the SPI clock.

As a consequence the length of the last bit is out of range and its reception on the bus is not ensured.

### Workaround

Check if a communication is ongoing before disabling the SPI interface. This can be done by monitoring the BSY bit in the SPI\_SR register.

## 2.6.2 Busy flag is not reliable when the SPI is a master simplex receive-only mode

### Description

When the master is simplex receive only mode, it provides the clock immediately after setting the SPE bit in the SPI\_CR1 register. In this case, the clock is provided until the SPE bit is cleared, meaning that the SPI is always busy because it is in receive-only mode only and continuously receives data. As a result, the BSY flag of the SPI\_SR register is unreliable to detect the SPI status when the SPI is in master receive-only mode.

### Workaround

None. The reference manual has been updated to explain how to handle this mode.

## 2.6.3 CRC may be corrupted by SPI configuration or other bus transfers

### Description

When the CRC is enabled (CRCEN bit set in the SPI\_CR2 register), the CRC calculation may be corrupted or unreliable if one of the following conditions is met:

- The CPHA bit, the CPOL bit, or the CRCPOLY bitfield is configured.
- The value of the polynomial programmed in the CRCPOLY bitfield of the SPI\_CRCPR register is even.
- A bus transfer is ongoing with another slave, or parasitic pulses are observed on the SCK output when the SPI is enabled in slave mode but not selected for communication.

### Workaround

Both the master and slave must reset and resynchronize their CRC calculation just before starting a new transfer secured by CRC.

Apply the following measures:

- Always configure the CPHA bit, the CPOL bit, and the CRCPOLY bitfield before setting the CRCEN bit.
- Always program an odd polynomial value in the CRCPOLY bitfield of the SPI\_CRCPR register (with bit 0 set).
- Before starting any transfer secured by CRC calculation, clear and set again the CRCEN bit while the SPI is disabled.

## 2.6.4 Anticipated communication upon SPI transit from slave receiver to master

### Description

The communication clock starts upon setting the MSTR bit even though the SPI is disabled, if transiting from the enabled slave receive only mode (RXONLY = 1) to whatever master mode.

### Workaround

Set the MSTR and SPE bits of the SPI\_CR1 register simultaneously. This forces the immediate start of the communication clock.

If the master is configured in transmitter mode (full-duplex or simplex), load the first data into the SPI\_DR data register before configuring the SPI\_CR1 register.

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

### Description

The BSY flag may sporadically remain high at the end of a data transfer in slave mode. The issue appears when an accidental synchronization happens between the internal CPU clock and the external SCK clock provided by the master.

This is related to the end of data transfer detection while the SPI is enabled in slave mode.

As a consequence, the end of the data transaction may be not recognized when the software needs to monitor it (for example at the end of a session before entering the low-power mode or before the direction of the data line has to be changed at half duplex bidirectional mode). The BSY flag is unreliable to detect the end of any data sequence transaction.

### 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 input.
- 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 the last data to the data register.
2. Poll TXE until it becomes high to ensure the data transfer has started.
3. Disable SPI by clearing SPE while the last data transfer is still ongoing.
4. Poll the BSY bit until it becomes low.
5. The BSY flag works correctly and can be used to recognize the end of the transaction.

*Note: This sequence can be used only when the CPU has enough performance to disable the SPI after a TXE event is detected, while the data frame transfer is still ongoing. It is impossible to achieve it when the ratio between CPU and SPI clock is low. In this specific case, the BSY check timeout can be measured by executing a fixed number of dummy instructions (such as NOP), corresponding to the time necessary to complete the data frame transaction.*

## **2.7 ADC peripheral limitation**

### **2.7.1 EOC interrupt triggered when AWDIE and EOCIE set to 1**

#### **Description**

When the analog watchdog is enabled and AWDIE and EOCIE are both set to 1, the ADC interrupt should only be triggered when the conversion result exceeds one of the analog watchdog thresholds.

However, for the devices covered by this datasheet, the interrupt is triggered after each conversion, thus leading to a high interrupt load.

#### **Workaround**

Set AWDIE to 1 and EOCIE to 0 instead and stop the conversions inside the ISR by resetting the CONT bit. However the latest conversion result having triggered the watchdog may be overwritten.

### 3 Important security notice

The STMicroelectronics group of companies (ST) places a high value on product security, which is why the ST product(s) identified in this documentation may be certified by various security certification bodies and/or may implement our own security measures as set forth herein. However, no level of security certification and/or built-in security measures can guarantee that ST products are resistant to all forms of attacks. As such, it is the responsibility of each of ST's customers to determine if the level of security provided in an ST product meets the customer needs both in relation to the ST product alone, as well as when combined with other components and/or software for the customer end product or application. In particular, take note that:

- ST products may have been certified by one or more security certification bodies, such as Platform Security Architecture ([www.psacertified.org](http://www.psacertified.org)) and/or Security Evaluation standard for IoT Platforms ([www.trustcb.com](http://www.trustcb.com)). For details concerning whether the ST product(s) referenced herein have received security certification along with the level and current status of such certification, either visit the relevant certification standards website or go to the relevant product page on [www.st.com](http://www.st.com) for the most up to date information. As the status and/or level of security certification for an ST product can change from time to time, customers should re-check security certification status/level as needed. If an ST product is not shown to be certified under a particular security standard, customers should not assume it is certified.
- Certification bodies have the right to evaluate, grant and revoke security certification in relation to ST products. These certification bodies are therefore independently responsible for granting or revoking security certification for an ST product, and ST does not take any responsibility for mistakes, evaluations, assessments, testing, or other activity carried out by the certification body with respect to any ST product.
- Industry-based cryptographic algorithms (such as AES, DES, or MD5) and other open standard technologies which may be used in conjunction with an ST product are based on standards which were not developed by ST. ST does not take responsibility for any flaws in such cryptographic algorithms or open technologies or for any methods which have been or may be developed to bypass, decrypt or crack such algorithms or technologies.
- While robust security testing may be done, no level of certification can absolutely guarantee protections against all attacks, including, for example, against advanced attacks which have not been tested for, against new or unidentified forms of attack, or against any form of attack when using an ST product outside of its specification or intended use, or in conjunction with other components or software which are used by customer to create their end product or application. ST is not responsible for resistance against such attacks. As such, regardless of the incorporated security features and/or any information or support that may be provided by ST, each customer is solely responsible for determining if the level of attacks tested for meets their needs, both in relation to the ST product alone and when incorporated into a customer end product or application.
- All security features of ST products (inclusive of any hardware, software, documentation, and the like), including but not limited to any enhanced security features added by ST, are provided on an "AS IS" BASIS. AS SUCH, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ST DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, unless the applicable written and signed contract terms specifically provide otherwise.

## 4 Revision history

**Table 4. Document revision history**

Date	Revision	Changes
24-Aug-2020	1	Initial release.
09-Sep-2021	2	Adjustment of the <i>Section 2.2.6: <math>V_{DD}</math> rise-time rate for <math>100mV &lt; V_{DD} &lt; 1V</math> workaround.</i>
05-Apr-2023	3	Updated: <i>Section 2.6.2: Busy flag is not reliable when the SPI is a master simplex receive-only mode</i> Added: <i>Section 3: Important security notice</i> <i>Section 2.6.3: CRC may be corrupted by SPI configuration or other bus transfers</i> <i>Section 2.6.4: Anticipated communication upon SPI transit from slave receiver to master</i> <i>Section 2.6.5: BSY bit may stay high at the end of data transfer in slave mode</i>

**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 acknowledgment.

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. For additional information about ST trademarks, please refer to [www.st.com/trademarks](http://www.st.com/trademarks). 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.

© 2023 STMicroelectronics – All rights reserved