STM8L05xx3 STM8L151x2/x3 Errata sheet

STM8L050J3, STM8L051F3, STM8L151x2 and STM8L151x3 device limitations

Applicability

This document applies to the part numbers of STM8L05xx3 and STM8L151x2/x3 devices listed in *Table 1* and their variants shown in *Table 2*.

Section 1 gives a summary and *Section 2* a description of / workaround for device limitations, with respect to the device datasheet and reference manual RM0031.

Table 1. Device summary

Reference	Part numbers
STM8L05xx3	STM8L051F3, STM8L050J3
STM8L151x2	STM8L151C2, STM8L151K2, STM8L151G2, STM8L151F2
STM8L151x3	STM8L151C3, STM8L151K3, STM8L151G3, STM8L151F3

Table 2. Device variants

Reference	Silicon revision codes
	Device marking(1)
STM8L051x3, STM8L151x2, STM8L151x3, STM8L050x3	Z

1. Refer to the device data sheet for how to identify this code on different types of package.

Contents

1 Summary of device limitations

The following table gives a quick references to all documented device limitations of STM8L05xx3 and STM8L151x2/x3 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. Z
Core	*2.1.1*	*Interrupt service routine (ISR) executed with priority of main process*	N
	2.1.2	*Main CPU execution is not resumed after an ISR resets the AL bit*	A
	2.1.3	*Unexpected DIV/DIVW instruction result in ISR*	A
	2.1.4	*Incorrect code execution when WFE execution is interrupted by ISR or event*	A
	2.1.5	*Core kept in stall mode when DMA transfer occurs during program/ erase operation to EEPROM*	A
	2.1.6	*Incorrect code execution when FLASH/EEPROM memory wakes up from power down mode*	A
System	*2.2.1*	*PA0, PB0 and PB4 configuration “at reset state” and “under reset”*	N
	2.2.2	*32.768 kHz LSE crystal accuracy may be disturbed by the use of adjacent I/Os*	N
I2C	*2.3.1*	*I2C event management*	A
	2.3.2	*Corrupted last received data in I2C Master Receiver mode*	A
	2.3.3	*Wrong behavior of the I2C peripheral in Master mode after misplaced STOP*	A
	2.3.4	*Violation of I2C “setup time for repeated START condition” parameter*	A
	2.3.5	*In I2C slave “NOSTRETCH” mode, underrun errors may not be detected and might generate bus errors*	A
	2.3.6	*SMBus standard not fully supported in I2C peripherals*	A
USART	*2.4.1*	*USART IDLE frame detection not supported in the case of a clock deviation*	N
	2.4.2	*PE flag can be cleared in USART Duplex mode by writing to the data register*	A
	2.4.3	*PE flag is not set in USART Mute mode using address mark detection*	N
	2.4.4	*IDLE flag is not set using address mark detection in the USART peripheral*	N

Table 3. Summary of device limitations (continued)

Function	Section	Limitation	Status
			Rev. Z
SPI	2.5.1	CRC may be corrupted by SPI configuration or other bus transfers	A
	2.5.2	Anticipated communication upon SPI transit from slave receiver to master	A
	2.5.3	BSY bit may stay high at the end of data transfer in slave mode	A

2 Description of device limitations

The following sections describe device limitations and provide workarounds if available. They are grouped by device functions.

2.1 Core

2.1.1 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.

Workaround

None.

No fix is planned for this limitation.

2.1.2 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.

2.1.3 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

- Option 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 your compiler documentation for details on the implementation and control of automatic or manual code insertion.

- Option 2

To optimize the number of cycles added by workaround 1, you 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.

2.1.4 Incorrect code execution when WFE execution is interrupted by ISR or event

Description

Two types of failures can occur:

Case 1:

In case WFE instruction is placed in the two MSB of the 32-bit word within the memory, an event which occurs during the WFE execution cycle or re-execution cycle (when returning from ISR handler) will cause an incorrect code execution.

Case 2:

An interrupt request, which occurs during the WFE execution cycle will lead to incorrect code execution. This is also valid for the WFE re-execution cycle, while returning from an ISR handler.

The above failures have no impact on the core behavior when the ISR request or events occur in Wait for Event mode itself, out of the critical single cycle of WFE instruction execution.

Workaround

General solution is to ensure no interrupt request or event occurs during WFE instruction execution or re-execution cycle by proper application timing.

Dedicated workarounds:

Case 1:

Replace the WFE instruction with

```
WFE
```

```
JRA next
next:
```

Case 2:

It is recommended to avoid any interrupts before WFE mode is entered. This can be done by disabling all interrupts before the device enters Wait for event mode.

```
SIM
WFE
RIM
```

This workaround also prevents WFE re-execution in case 1.

No fix is planned for this limitation.

2.1.5 Core kept in stall mode when DMA transfer occurs during program/ erase operation to EEPROM

Description

When the MCU performs EEPROM program/erase operation, the core is stalled during data transfer to the memory controller, which occurs at the beginning of the program/erase operation. If a DMA request servicing starts while the core is stalled, the core does not return from stall mode to program execution.

The core is stalled for 11 cycles during byte program/erase, 8 cycles during word program/erase and 3 cycles during each word transfer in block programming mode. For block erase, the core is stalled for 127 cycles.

When a DMA request arises, it is only served if the DMA priority is higher than the core access priority.

If the current DMA priority is lower than the core one, the DMA service is delayed until the core access becomes idle.

The DMA also includes a programmable timeout function, configurable by DMA_GCSR register. If the core does not release the bus during this timeout, the DMA automatically increases its own priority and forces the core to release the bus for DMA service.

No fix is planned for this limitation.

Several workarounds are available for this limitation.

Workaround

- Option 1

Disable all DMA requests during data transfer to the EEPROM.

This workaround is applicable for all program/erase operations.

- Option 2

Configure DMA programmable timeout in the DMA_GCSR register to exceed the number of stall cycles required during the transfer. DMA priority must never be configured to a very high level.

This workaround is applicable for all program/erase operations except block erase.

In order to apply this workaround to block erase, use block programming to 0x00 instead of block erase. This takes ~6 ms instead of ~3 ms.

- Option 3

This workaround can be used if block erase cannot be replaced by block programming.

In this workaround, DMA is used to transfer data to the EEPROM instead of the core. All other DMA transfers are delayed once the core is stalled due to data transfer to memory controller.

```
/* start of the workaround in user code, using FW Library */
#ifdef USE_EVENT_MODE
   DMA1_Channel3->CCR= DMA_CCR_MEM | DMA_CCR_IDM | DMA_CCR_TCIE; /*
Config DMA Chn3 Mem, incr, disable, interrupt) */
#else
   DMA1_Channel3->CCR= DMA_CCR_MEM | DMA_CCR_IDM; /* Config DMA
Chn3 (Mem, incr,disable) */
#endif
   DMA1_Channel3->CM0ARH= (uint8_t)0; /* Source address */
   DMA1_Channel3->CM0ARL= (uint8_t)0;
   DMA1_Channel3->CPARH= (uint8_t)(addr_begin >> 8); /* Destination
address */
   DMA1_Channel3->CPARL= (uint8_t)(addr_begin);
   DMA1_Channel3->CNBTR= 2; /* Number of data to be transferred */
   DMA1_Channel3->CSPR= 8; /* Low priority, 16 bit mode */
   DMA1_Channel3->CSPR &= ~DMA_CSPR_TCIF;/* Clear TCIF */
   DMA1->GCSR|= 1; /* Global DMA enable */

#ifdef USE_EVENT_MODE
   WFE->CR3 = WFE_CR3_DMA1CH23_EV; /* Enable event generation on
   DMA */
#endif
   FLASH->DUKR = 0xAE; /* Unprotect data memory */
   FLASH->DUKR = 0x56;
   while((FLASH->IAPSR & FLASH_IAPSR_DUL) == 0)
   {} /* Polling DUL */
   FLASH_Block_Load();
/* end of the workaround in user code */

/* following routine has to be placed in the RAM */
void FLASH_Block_Load(){
   __asm("sim\n"); /* Disable interrupts */
   FLASH->CR2 |= FLASH_CR2_ERASE; /* Enable erase block mode */
   DMA1_Channel3->CCR|= DMA_CCR_CE; /* Enable DMA MEM transfer */
#ifdef USE_EVENT_MODE
   __asm("wfe"); /* Wait for end of DMA operation */
#else
   while((DMA1_Channel3->CSPR & DMA_CSPR_TCIF) == 0)
```

```
   {} /* Polling for end of DMA operation */
#endif
   __asm("rim\n"); /* Enable interrupts */
}
```

2.1.6 Incorrect code execution when FLASH/EEPROM memory wakes up from power down mode

Description

In case FLASH/EEPROM memory is put in power down mode (I_{DDQ}), first read after wakeup could return an incorrect content when F_{CPU} is above 8 MHz + 5%.

FLASH/EEPROM memory is put in I_{DDQ} mode by default during Halt mode and could be forced to I_{DDQ} mode by software for wait mode and during RAM execution.

As a consequence, 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 incorrect read of vector table.
- Code running from RAM read an incorrect value from FLASH/EEPROM memory, when forced in I_{DDQ} mode.
- Program execution gets corrupted when returning from RAM execution to FLASH memory execution in case FLASH memory is forced in I_{DDQ} mode.

Workaround

Slow down F_{SYSCLK} before entering Low power mode to ensure correct FLASH memory wakeup. This could be done using clock divider (CLK_CKDIVR) or by activation of fast wakeup feature by setting FHWU bit in CLK_ICKCR register. Original clock setting can be reconfigured back by software after wakeup.

Code example, assuming no divider is used in application by default.

```
CLK_CKDIVR = 0x01;
_asm(“HALT”);
CLK_CKDIVR = 0x00;
```

The interrupt service routine executed after wakeup could either stay at slower clock speed, or reconfigure clock setting. Care has to be taken to restore previous clock divider at the end of interrupt routines when modifying clock divider.

No fix planned for this limitation.

2.2 System

2.2.1 PA0, PB0 and PB4 configuration “at reset state” and “under reset”

Description

When a reset occurs, PA0, PB0 and PB4 configurations differ from the other pins:

- PA0 is configured as input with pull-up under reset (i.e. during the reset phase) and at reset state (i.e. after internal reset release).
- A pull-up is applied to PB0 and PB4 under reset (i.e. during the reset phase). These two pins are input floating at reset state (i.e. after internal reset release).

Workaround

None.

No fix is planned for this limitation.

2.2.2 32.768 kHz LSE crystal accuracy may be disturbed by the use of adjacent I/Os

Description

The activity on the PC4 and PC7 I/Os (input or output) can lead to missing pulses on the low speed external oscillator (32.768 kHz external crystal).

Workaround

None.

If a high LSE accuracy is required, PC4 and PC7 must be tied to V_{DD} or V_{SS}.

No fix is planned for this limitation.

2.3 I2C

2.3.1 I2C event management

Description

As described in the I2C section of the STM8L05/15x microcontroller family reference manual (RM0031), 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:

1. Use the I2C with DMA in general, except when the Master is receiving a single byte.
2. Use I2C interrupts in nested mode and boost their priorities to the highest one in the application to make them uninterruptible.

2.3.2 Corrupted last received data in I2C Master Receiver mode

Description

- 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 the data N-1 before the STOP condition is generated on the bus. In this case, reading data N returns a wrong value.

Workaround

- Option 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.
- Option 2

  Manage I2C RxNE and TxE events with DMA or interrupts of the highest priority level, so that the condition BTF = 1 never occurs.

No fix is planned for this limitation.

2.3.3 Wrong behavior of the 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 allowed to send a Stop only at the end of the full byte (8 bits + acknowledge), so this scenario is not allowed. Other derived protocols like CBUS allow it, but they are not supported by the I2C 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 I2C 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.

2.3.4 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 tSU(STA) in the datasheet and Tsu: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. tSU(STA) minimum value may be 4 ìs instead of 4.7 ìs.

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.

No fix is planned for this limitation.

2.3.5 In I2C slave “NOSTRETCH” mode, underrun errors may not be detected and might 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. You can use an interrupt on the TXE or ADDR flag and boost its priority to the higher level or use DMA.

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 tLOW.

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.

No fix is planned for this limitation.

2.3.6 SMBus standard not fully supported in I2C peripherals

Description

The I2C peripheral is not fully compliant with the SMBus v2.0 standard since it does not support the capability to NACK an invalid byte/command.

Workaround

A higher-level mechanism should be used to verify that a write operation is being performed correctly at the target device, such as:

- The use of the SMBA pin if supported by the host
- The alert response address (ARA) protocol
- The Host notify protocol.

No fix is planned for this limitation.

2.4 USART

2.4.1 USART IDLE frame detection not supported in the case of a clock deviation

Description

An idle frame cannot be detected if the receiver clock is deviated.

If a valid idle frame of a minimum length (depending on the M and Stop bit numbers) is followed without any delay by a start bit, the IDLE flag is not set if the receiver clock is deviated from the RX line (only if the RX line switches before the receiver clock).

Consequently, the IDLE flag is not set even if a valid idle frame occurred.

Workaround

None.

No fix is planned for this limitation.

2.4.2 PE flag can be cleared in USART Duplex mode by writing to the data register

Description

The PE flag can be cleared by a read to the USART_SR register followed by a read or a write to the USART_DR register.

When working in duplex mode, the following event can occur: the PE flag set by the receiver at the end of a reception is cleared by the software transmitter reading the USART_SR (to check TXE or TC flags) and writing a new data into the USART_DR.

The software receiver can also read a PE flag at ‘0’ if a parity error occurred.

Workaround

The PE flag should be checked before writing to the USART_DR.

No fix is planned for this limitation.

2.4.3 PE flag is not set in USART Mute mode using address mark detection

Description

If, when using address mark detection, the receiver recognizes in Mute mode a valid address frame but the parity check fails, it exits from the Mute mode without setting the PE flag.

Workaround

None.

No fix is planned for this limitation.

2.4.4 IDLE flag is not set using address mark detection in the USART peripheral

Description

The IDLE flag is not set when the address mark detection is enabled, even when the USART is in Run mode (not only in Mute mode).

Workaround

None.

No fix is planned for this limitation.

2.5 SPI

2.5.1 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 (bit 0 set).
- Before starting any transfer secured by CRC calculation, clear and set again the CRCEN bit while the SPI is disabled.

2.5.2 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, which 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.

2.5.3 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.

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.

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) and/or Security Evaluation standard for IoT Platforms (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 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
30-Jun-2011	1	Initial release.
01-Aug-2011	2	**Added:** *Section 2.1.5: Core kept in stall mode when DMA transfer occurs during program/ erase operation to EEPROM*
18-Feb-2013	3	**Added:** STM8L051F3 and STM8L151C2 part numbers **Modified:** *Section 2.1.4: Incorrect code execution when WFE execution is interrupted by ISR or event* Cover page
21-Jun-2013	4	**Added:** *Section 2.1.6: Incorrect code execution when FLASH/EEPROM memory wakes up from power down mode*
03-Oct-2017	5	**Added:** – STM8L050J3 part number **Modified:** – Cover page **Removed:** – Appendix A Revision code on device marking
08-Feb-2023	6	**Added:** – *Section 2.5: SPI* – *Section 2.5.1: CRC may be corrupted by SPI configuration or other bus transfers* – *Section 2.5.2: Anticipated communication upon SPI transit from slave receiver to master* – *Section 2.5.3: BSY bit may stay high at the end of data transfer in slave mode* **Modified:** *Section Table 3.: Summary of device limitations*
07-Mar-2023	7	**Updated:** *Section 2.5.1: CRC may be corrupted by SPI configuration or other bus transfers* **Added:** *Section 3: Important security notice*