Applicability

This document applies to the part numbers of STM32G041x6/x8 devices and the device variants as stated in this page. It gives a summary and a description of the device errata, with respect to the device datasheet and reference manual RM0444. 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 summary

<table>
<thead>
<tr>
<th>Reference</th>
<th>Part numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32G041x6</td>
<td>STM32G041C6, STM32G041K6, STM32G041F6, STM32G041G6, STM32G041J6</td>
</tr>
<tr>
<td>STM32G041x8</td>
<td>STM32G041C8, STM32G041K8, STM32G041F8, STM32G041G8, STM32G041Y8</td>
</tr>
</tbody>
</table>

Table 2. Device variants

<table>
<thead>
<tr>
<th>Reference</th>
<th>Silicon revision codes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Device marking(1)</td>
</tr>
<tr>
<td>STM32G041xx</td>
<td>Z</td>
</tr>
</tbody>
</table>

1. Refer to the device data sheet for how to identify this code on different types of package.
2. REV_ID[15:0] bitfield of DBGMCU_IDCODE register.
1 Summary of device errata

The following table gives a quick reference to the STM32G041x6/x8 device limitations 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Section</th>
<th>Limitation</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>2.2.1</td>
<td>Unstable LSI when it clocks RTC or CSS on LSE</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>2.2.2</td>
<td>WUFx wakeup flag wrongly set during configuration</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>2.2.3</td>
<td>Under Level 1 read protection, booting from Main Flash memory selected through PA14-BOOT0 pin is not functional</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>2.2.4</td>
<td>Overwriting with all zeros a Flash memory location previously programmed with all ones fails</td>
<td>N</td>
</tr>
<tr>
<td>DMA</td>
<td>2.3.1</td>
<td>DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear</td>
<td>A</td>
</tr>
<tr>
<td>DMAMUX</td>
<td>2.4.1</td>
<td>SOFx not asserted when writing into DMAMUX_CFR register</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>2.4.2</td>
<td>OFx not asserted for trigger event coinciding with last DMAMUX request</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>2.4.3</td>
<td>OFx not asserted when writing into DMAMUX_RGCFR register</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>2.4.4</td>
<td>Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event</td>
<td>A</td>
</tr>
<tr>
<td>TIM</td>
<td>2.5.1</td>
<td>One-pulse mode trigger not detected in master-slave reset + trigger configuration</td>
<td>P</td>
</tr>
<tr>
<td>LPTIM</td>
<td>2.6.1</td>
<td>MCU may remain stuck in LPTIM interrupt when entering Stop mode</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>2.6.2</td>
<td>MCU may remain stuck in LPTIM interrupt when clearing event flag</td>
<td>P</td>
</tr>
<tr>
<td>RTC and TAMP</td>
<td>2.7.1</td>
<td>Calendar initialization may fail in case of consecutive INIT mode entry</td>
<td>A</td>
</tr>
<tr>
<td>I2C</td>
<td>2.8.1</td>
<td>Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>2.8.2</td>
<td>Spurious bus error detection in master mode</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>2.8.3</td>
<td>Spurious master transfer upon own slave address match</td>
<td>P</td>
</tr>
<tr>
<td>SPI</td>
<td>2.9.1</td>
<td>BSY bit may stay high when SPI is disabled</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>2.9.2</td>
<td>BSY bit may stay high at the end of data transfer in slave mode</td>
<td>A</td>
</tr>
</tbody>
</table>
2 Description of device errata

The following sections describe limitations of the applicable devices with Arm® core and provide workarounds if available. They are grouped by device functions.

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

2.1 Core

Errata notice for the Arm® Cortex®-M0+ core revision r0p1 is available from http://infocenter.arm.com.

2.2 System

2.2.1 Unstable LSI when it clocks RTC or CSS on LSE

Description

The LSI clock can become unstable (duty cycle different from 50 %) and its maximum frequency can become significantly higher than 32 kHz, when:
- LSI clocks the RTC, or it clocks the clock security system (CSS) on LSE (which holds when the LSECSSON bit set), and
- the V_DD power domain is reset while the backup domain is not reset, which happens:
  - upon exiting Shutdown mode
  - if V_BAT is separate from V_DD and V_DD goes off then on
  - if V_BAT is tied to V_DD (internally in the package for products not featuring the VBAT pin, or externally) and a short (< 1 ms) V_DD drop under V_DD(min) occurs

Workaround

Apply one of the following measures:
- Clock the RTC with LSE or HSE/32, without using the CSS on LSE
- If LSI clocks the RTC or when the LSECSSON bit is set, reset the backup domain upon each V_DD power up (when the BORRSTF flag is set). If V_BAT is separate from V_DD, also restore the RTC configuration, backup registers and anti-tampering configuration.

2.2.2 WUFx wakeup flag wrongly set during configuration

Description

Upon configuring a wakeup pin (WKUPx), the corresponding wakeup flag (WUFx) might spuriously go high depending on the state and configuration of the wakeup pin.

Workaround

After configuring a wakeup pin, clear its corresponding WUFx flag.

2.2.3 Under Level 1 read protection, booting from Main Flash memory selected through PA14-BOOT0 pin is not functional

Description

With the Flash memory read protection set to Level 1 and the boot mode selected through the PA14-BOOT0 pin (BOOT0 function of the pin), an attempt to boot from Main Flash memory can wrongly be interpreted by the read protection mechanism as an unauthorized access, preventing the user code execution. Booting from Main Flash memory operates correctly if selected through option bits (nBOOT_SEL and nBOOT0 both set).
2.2.4 Overwriting with all zeros a Flash memory location previously programmed with all ones fails

**Description**

Any attempt to re-program with all zeros (0x0000 0000 0000 0000) a Flash memory location previously programmed with 0xFFFF FFFF FFFF FFFF fails and the PROGERR flag of the FLASH_SR register is set.

**Workaround**

None.

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 DMA_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**

Do not clear GIFx flags. Instead, use HTIFx, TCIFx, and TEIFx specific event flags and their corresponding clear bits.

2.4 DMAMUX

2.4.1 SOFx not asserted when writing into DMAMUX_CFR register

**Description**

The SOFx flag of the DMAMUX_CSR status register is not asserted if overrun from another DMAMUX channel occurs when the software writes into the DMAMUX_CFR register.

This can happen when multiple DMA channels operate in synchronization mode, and when overrun can occur from more than one channel. As the SOFx flag clear requires a write into the DMAMUX_CFR register (to set the corresponding CSOFx bit), overrun occurring from another DMAMUX channel operating during that write operation fails to raise its corresponding SOFx flag.

**Workaround**

None. Avoid the use of synchronization mode for concurrent DMAMUX channels, if at least two of them potentially generate synchronization overrun.

2.4.2 OFx not asserted for trigger event coinciding with last DMAMUX request

**Description**

In the DMAMUX request generator, a trigger event detected in a critical instant of the last-generated DMAMUX request being served by the DMA controller does not assert the corresponding trigger overrun flag OFx. The critical instant is the clock cycle at the very end of the trigger overrun condition.
Additionally, upon the following trigger event, one single DMA request is issued by the DMAMUX request generator, regardless of the programmed number of DMA requests to generate.

The failure only occurs if the number of requests to generate is set to more than two (GNBREQ[4:0] > 00001).

**Workaround**

Make the trigger period longer than the duration required for serving the programmed number of DMA requests, so as to avoid the trigger overrun condition from occurring on the very last DMA data transfer.

---

**2.4.3 OFx not asserted when writing into DMAMUX_RGCFR register**

**Description**

The OFx flag of the DMAMUX_RGSR status register is not asserted if an overrun from another DMAMUX request generator channel occurs when the software writes into the DMAMUX_RGCFR register. This can happen when multiple DMA channels operate with the DMAMUX request generator, and when an overrun can occur from more than one request generator channel. As the OFx flag clear requires a write into the DMAMUX_RGCFR register (to set the corresponding COFx bit), an overrun occurring in another DMAMUX channel operating with another request generator channel during that write operation fails to raise the corresponding OFx flag.

**Workaround**

None. Avoid the use of request generator mode for concurrent DMAMUX channels, if at least two channels are potentially generating a request generator overrun.

---

**2.4.4 Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event**

**Description**

If a write access into the DMAMUX_CxCR register having the SE bit at zero and SPOL[1:0] bitfield at a value other than 00:

- sets the SE bit (enables synchronization),
- modifies the values of the DMAREQ_ID[5:0] and SYNC_ID[4:0] bitfields, and
- does not modify the SPOL[1:0] bitfield,

and if a synchronization event occurs on the previously selected synchronization input exactly two AHB clock cycles before this DMAMUX_CxCR write, then the input DMA request selected by the DMAREQ_ID[5:0] value before that write is routed.

**Workaround**

Ensure that the SPOL[1:0] bitfield is at 00 whenever the SE bit is 0. When enabling synchronization by setting the SE bit, always set the SPOL[1:0] bitfield to a value other than 00 with the same write operation into the DMAMUX_CxCR register.

---

**2.5 TIM**

**2.5.1 One-pulse mode trigger not detected in master-slave reset + trigger configuration**

**Description**

The failure occurs when several timers configured in one-pulse mode are cascaded, and the master timer is configured in combined reset + trigger mode with the MSM bit set:

- OPM=1 in TIMx_CR1,
- SMS[3:0]=1000 and MSM=1 in TIMx_SMCR.

The MSM delays the reaction of the master timer to the trigger event, so as to have the slave timers cycle-accurately synchronized.

If the trigger arrives when the counter value is equal to the period value set in the TIMx_ARR register, the one-pulse mode of the master timer does not work and no pulse is generated on the output.
Workaround

None. However, unless a cycle-level synchronization is mandatory, it is advised to keep the MSM bit reset, in which case the problem is not present. The MSM=0 configuration also allows to decrease the timer latency to external trigger events.

2.6 LPTIM

2.6.1 MCU may remain stuck in LPTIM interrupt when entering Stop mode

Description

This limitation occurs when disabling the low-power timer (LPTIM).

When the user application clears the ENABLE bit in the LPTIM_CR register within a small time window around one LPTIM interrupt occurrence, then the LPTIM interrupt signal used to wake up the MCU from Stop mode may be frozen in active state. Consequently, when trying to enter Stop mode, this limitation prevents the MCU from entering low-power mode and the firmware remains stuck in the LPTIM interrupt routine.

This limitation applies to all Stop modes and to all instances of the LPTIM. Note that the occurrence of this issue is very low.

Workaround

In order to disable a low power timer (LPTIMx) peripheral, do not clear its ENABLE bit in its respective LPTIM_CR register. Instead, reset the whole LPTIMx peripheral via the RCC controller by setting and resetting its respective LPTIMxRST bit in RCC_APBxRSTRz register.

2.6.2 MCU may remain stuck in LPTIM interrupt when clearing event flag

Description

This limitation occurs when the LPTIM is configured in interrupt mode (at least one interrupt is enabled) and the software clears any flag by writing the LPTIM_ICR bit in the LPTIM_ISR register. If the interrupt status flag corresponding to a disabled interrupt is cleared simultaneously with a new event detection, the set and clear commands might reach the APB domain at the same time, leading to an asynchronous interrupt signal permanently stuck at ‘1’.

This issue can occur either during an interrupt subroutine execution (where the flag clearing is usually done), or outside an interrupt subroutine.

Consequently, the firmware remains stuck in the LPTIM interrupt routine, and the MCU cannot enter Stop mode.

Workaround

To avoid this issue, it is strongly advised to follow the recommendations listed below:

- Clear the flag only when its corresponding interrupt is enabled in the interrupt enable register.
- If for specific reasons, it is required to clear some flags that have corresponding interrupt lines disabled in the interrupt enable register, it is recommended to clear them during the current subroutine prior to those which have corresponding interrupt line enabled in the interrupt enable register.
- Flags must not be cleared outside the interrupt subroutine.

Note: The proper clear sequence is already implemented in the HAL_LPTIM_IRQHandler in the.

2.7 RTC and TAMP
2.7.1 Calendar initialization may fail in case of consecutive INIT mode entry

**Description**
If the INIT bit of the RTC_ICSR register is set between one and two RTCCLOCK cycles after being cleared, the INITF flag is set immediately instead of waiting for synchronization delay (which should be between one and two RTCCLOCK cycles), and the initialization of registers may fail. Depending on the INIT bit clearing and setting instants versus the RTCCLOCK edges, it can happen that, after being immediately set, the INITF flag is cleared during one RTCCLOCK period then set again. As writes to calendar registers are ignored when INITF is low, a write occurring during this critical period might result in the corruption of one or more calendar registers.

**Workaround**
After exiting the initialization mode, clear the BYPSHAD bit (if set) then wait for RSF to rise, before entering the initialization mode again.

*Note:* It is recommended to write all registers in a single initialization session to avoid accumulating synchronization delays.

2.8 I2C

2.8.1 Wrong data sampling when data setup time (t_{SU;DAT}) is shorter than one I2C kernel clock period

**Description**
The I²C-bus specification and user manual specify a minimum data setup time (t_{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 I²C-bus SDA line when t_{SU;DAT} is smaller than one I2C kernel clock (I²C-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 I²C-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.

2.8.2 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 I²C-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.
2.8.3 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 I2C-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 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 ADDRCF 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 ADDRCF 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.

2.9 SPI

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.
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.
Revision history

Table 4. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>21-Jun-2018</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Summary of device errata</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Description of device errata</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>Core</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>System</td>
<td></td>
</tr>
<tr>
<td>2.2.1</td>
<td>Unstable LSI when it clocks RTC or CSS on LSE</td>
<td></td>
</tr>
<tr>
<td>2.2.2</td>
<td>WUFx wakeup flag wrongly set during configuration</td>
<td></td>
</tr>
<tr>
<td>2.2.3</td>
<td>Under Level 1 read protection, booting from Main Flash memory selected through PA14-BOOT0 pin is not functional</td>
<td></td>
</tr>
<tr>
<td>2.2.4</td>
<td>Overwriting with all zeros a Flash memory location previously programmed with all ones fails</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>DMA</td>
<td></td>
</tr>
<tr>
<td>2.3.1</td>
<td>DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear</td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td>DMAMUX</td>
<td></td>
</tr>
<tr>
<td>2.4.1</td>
<td>SOFx not asserted when writing into DMAMUX_CFR register</td>
<td></td>
</tr>
<tr>
<td>2.4.2</td>
<td>OFFx not asserted for trigger event coinciding with last DMAMUX request</td>
<td></td>
</tr>
<tr>
<td>2.4.3</td>
<td>OFFx not asserted when writing into DMAMUX_RGCFR register</td>
<td></td>
</tr>
<tr>
<td>2.4.4</td>
<td>Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>TIM</td>
<td></td>
</tr>
<tr>
<td>2.5.1</td>
<td>One-pulse mode trigger not detected in master-slave reset + trigger configuration</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>LPTIM</td>
<td></td>
</tr>
<tr>
<td>2.6.1</td>
<td>MCU may remain stuck in LPTIM interrupt when entering Stop mode</td>
<td></td>
</tr>
<tr>
<td>2.6.2</td>
<td>MCU may remain stuck in LPTIM interrupt when clearing event flag</td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>RTC and TAMP</td>
<td></td>
</tr>
<tr>
<td>2.7.1</td>
<td>Calendar initialization may fail in case of consecutive INIT mode entry</td>
<td></td>
</tr>
<tr>
<td>2.8</td>
<td>I2C</td>
<td></td>
</tr>
<tr>
<td>2.8.1</td>
<td>Wrong data sampling when data setup time ( t_{SU;DAT} ) is shorter than one I2C kernel clock period</td>
<td></td>
</tr>
<tr>
<td>2.8.2</td>
<td>Spurious bus error detection in master mode</td>
<td></td>
</tr>
<tr>
<td>2.8.3</td>
<td>Spurious master transfer upon own slave address match</td>
<td></td>
</tr>
<tr>
<td>2.9</td>
<td>SPI</td>
<td></td>
</tr>
</tbody>
</table>
2.9.1 BSY bit may stay high when SPI is disabled ........................................ 8
2.9.2 BSY bit may stay high at the end of data transfer in slave mode. ............. 8

Revision history .......................................................................................... 10