STM32L1xxx6/8/B-A Errata sheet

STM32L100x6/8/B-A STM32L151x6/8/B-A and STM32L152x6/8/B-A ultra-low-power device limitations

Silicon identification

This errata sheet applies to revision Z and Y of the STMicroelectronics STM32L100x6/8/B-A, STM32L151x6/8/B-A and STM32L152x6/8/B-A ultra-low-power products. This family features an ARM® 32-bit Cortex®-M3 core, for which an errata notice is also available (see *Section 1* for details). A full list of root part numbers is shown in *Table 1*.

The products can be identified (see *Table 2*) by:

- The revision code marked below the sales type on the device package
- The last three digits of the internal sales type printed on the box label

Table 1. Device summary

Reference	Part number
STM32L100C6-A, STM32L100R8-A, STM32L100RB-A	STM32L100C6xxA, STM32L100R8xxA, STM32L100RBxxA
STM32L151CB-A, STM32L151C8-A, STM32L151C6-A, STM32L151RB-A, STM32L151R8-A, STM32L151R6-A, STM32L151VB-A, STM32L151V8-A	STM32L151CBxxA, STM32L151C8xxA, STM32L151C6xxA, STM32L151RBxxA, STM32L151R8xxA, STM32L151R6xxA, STM32L151VBxxA, STM32L151V8xxA
STM32L152CB-A, STM32L152C8-A, STM32L152C6-A, STM32L152RB-A, STM32L152R8-A, STM32L152R6-A, STM32L152VB-A, STM32L152V8-A	STM32L152CBxxA, STM32L152C8xxA, STM32L152C6xxA, STM32L152RBxxA, STM32L152R8xxA, STM32L152R6xxA, STM32L152VBxxA, STM32L152V8xxA

Table 2. Device identification(1)

Sales type	Revision code(2) marked on device
STM32L100x6/8/B-A, STM32L151x6/8/B-A and STM32L152x6/8/B-A	“Z” or “Y”

1. The REV_ID bits in the DBGMCU_IDCODE register show the revision code of the device (see the *STM32L100xx, STM32L151xx, STM32L152xx and STM32L162xx advanced ARM®-based 32-bit MCUs* reference manual (RM0038) for details on how to find the revision code).
2. Refer to the device datasheet for details on how to identify the revision code on the different packages.

Contents

List of tables

1 ARM® 32-bit Cortex®-M3 limitations

An ARM errata notice of the STM32L1xxxx core is available searching for “Cortex®-M3 errata” at the following web address: www.arm.com.

All the described limitations are minor and relate to revision r2p0-00rel0 of the Cortex®-M3 core. *Table 3* summarizes these limitations and their implications on the behavior of the STM32L1xxxx ultra-low-power devices.

Table 3. Cortex®-M3 core limitations and impact on microcontroller behavior

ARM ID	ARM category	ARM summary of errata	Impact on STM32L1xxxx ultra-low-power devices
602117	Cat 2	LDRD with base in list may result in incorrect base register when interrupted or faulted	Minor
563915	Cat 2	Event register is not set by interrupts and debug	Minor
752419	Cat 2	Interrupted loads to SP can cause erroneous behavior	Minor
740455	Cat 2	SVC and BusFault/MemManage may occur out of order	Minor

1.1 Cortex®-M3 limitation description for the STM32L1xxxx ultra-low-power devices

Only the limitations described below have an impact, even though minor, on the implementation of STM32L1xxxx ultra-low-power devices.

All other limitations described in the ARM errata notice (and summarized in *Table 3* above) have no impact and are not related to the implementation of the STM32L1xxxx ultra-low-power devices (Cortex-M3 r2p0-00rel0).

1.1.1 Cortex®-M3 LDRD with base in list may result in incorrect base register when interrupted or faulted

Description

The Cortex®-M3 Core has a limitation when executing an LDRD instruction from the system-bus area, with the base register in a list of the form LDRD Ra, Rb, [Ra, #imm]. The execution may not complete after loading the first destination register due to an interrupt before the second loading completes or due to the second loading getting a bus fault.

Workaround

1. This limitation does not impact the STM32L1xxxx code execution when executing from the embedded Flash memory, which is the standard use of the microcontroller.
2. Use the latest compiler releases. As of today, the compilers no longer generate this particular sequence. Moreover, a scanning tool is provided to detect this sequence on previous releases (refer to your preferred compiler provider).

1.1.2 Cortex®-M3 event register is not set by interrupts and debug

Description

When interrupts related to a wake from event (WFE) occur before the WFE is executed, the event register used for WFE wakeup events is not set and the event is missed. Therefore, when the WFE is executed, the core does not wake up from a WFE if no other event or interrupt occurs.

Workaround

1. For the following interrupt sources:
   - All external interrupts/events lines (EXTI)
   - PVD output on EXTI line 16 (if VREFINT is enabled only)
   - RTC alarm on EXTI line 17
   - USB wakeup on EXTI line 18
   - RTC tamper and timestamp on EXTI line 19
   - RTC wakeup on EXTI line 20
   - Comparator 1 wake-up on EXTI line 21 (if VREFINT is enabled only)
   - Comparator 2 wake-up on EXTI line 22 (if VREFINT is enabled only)
   - Channel acquisition on EXTI line 23

   Use STM32L1xxxx external events instead of interrupts to wake up the core from a WFE by configuring an external or internal EXTI line in event mode.
2. For all other interrupt sources, a timer must be programmed to provide a timeout event and wake up the core if the event is likely to arrive before the WFE instruction is executed.

1.1.3 Cortex®-M3 Interrupted loads to the stack-pointer can cause erroneous behavior

Description

If an interrupt occurs during the data-phase of a single word load to the stack-pointer (SP/R13), an erroneous behavior can occur. In all cases, returning from the interrupt will result in the load instruction being executed in an additional time. For all instructions performing an update to the base register, the base register will be erroneously updated on each execution, resulting in the stack-pointer being loaded from an incorrect memory location.

The instructions affected by this limitation are the following:

- LDR SP, [Rn],#imm
- LDR SP, [Rn,#imm]!
- LDR SP, [Rn,#imm]
- LDR SP, [Rn]
- LDR SP, [Rn,Rm]

Workaround

As of today, no compiler generates these particular instructions. This limitation can only occur with hand-written assembly code.

Both issues can be solved by replacing the direct load to the stack pointer by an intermediate load to a general-purpose register followed by a move to the stack pointer.

Example:

Replace LDR SP, [R0] by

LDR R2,[R0]

MOV SP,R2

1.1.4 SVC and BusFault/MemManage may occur out of order

Description

If an SVC exception is generated by executing the SVC instruction while the following instruction fetch is faulted, then the MemManage or BusFault handler may be entered even though the faulted instruction which followed the SVC should not have been executed.

Workaround

A workaround is only required if the SVC handler will not return to the return address that has been stacked for the SVC exception and the instruction access after the SVC will fault. If this is the case then padding can be inserted between the SVC and the faulting area of code, for example, by inserting NOP instructions.

2 STM32L100x6/8/B-A, STM32L151x6/8/B-A and STM32L152x6/8/B-A silicon limitations

Table 4 summarizes the fix status for products listed in *Table 2: Device identification*.

The legend for *Table 4* is as follows:

A = workaround available,

N = no workaround available,

P = partial workaround available,

'-' and grayed = fixed.

Table 4. Summary of silicon limitations

Links to silicon limitations		Rev Z	Rev Y
Section 2.1: System limitations	*Section 2.1.1: Number of RTC backup registers is 20 bytes instead of 80 bytes*	N	N
	Section 2.1.2: If debugger is connected in JTAG mode and JNRST (PB4) pin configuration is changed, the connection is lost	A	A
	Section 2.1.3: Data EEPROM cycling limited to 100 kcycles	A	A
	Section 2.1.4: Wakeup sequence from Standby mode when using more than one wakeup source	A	A
	Section 2.1.5: Flash memory wakeup issue when waking up from Stop or Sleep with Flash memory in power-down mode	A	-
	Section 2.1.6: Unexpected system reset when waking up from Stop mode with regulator in low-power mode	A	-
Section 2.2: IWDG peripheral limitation	*Section 2.2.1: RVU and PVU flags are not reset in Stop mode*	A	A
Section 2.3: I2C peripheral limitations	*Section 2.3.1: SMBus standard not fully supported*	A	A
	Section 2.3.2: Wrong behavior of I2C peripheral in Master mode after misplaced STOP	A	A
	Section 2.3.3: Violation of I2C "setup time for repeated START condition" parameter	A	A
	Section 2.3.4: In I2C slave "NOSTRETCH" mode, underrun errors may not be detected and may generate bus errors	A	A
	Section 2.3.5: 10-bit Master mode: new transfer cannot be launched if first part of the address has not been acknowledged by the slave	A	A

Table 4. Summary of silicon limitations (continued)

Links to silicon limitations		Rev Z	Rev Y
Section 2.4: SPI/I2S peripheral limitations	*Section 2.4.1: In I2S slave mode, WS level must to be set by the external master when enabling the I2S*	A	A
	Section 2.4.2: BSY bit may stay high at the end of a data transfer in Slave mode	A	A
	Section 2.4.3: Wrong CRC calculation when the polynomial is even	A	A
	Section 2.4.4: Wrong CRC transmitted in Master mode with delayed SCK feedback	A	A
	Section 2.4.5: SPI CRC may be corrupted when a peripheral connected to the same DMA channel of the SPI finishes its DMA transaction	N	N
	Section 2.4.6: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback	A	A
Section 2.5: USART peripheral limitations	*Section 2.5.1: Idle frame is not detected if receiver clock speed is deviated*	N	N
	Section 2.5.2: In full duplex mode, the Parity Error (PE) flag can be cleared by writing the data register	A	A
	Section 2.5.3: Parity Error (PE) flag is not set when receiving in Mute mode using address mark detection	N	N
	Section 2.5.4: Break frame is transmitted regardless of nCTS input line status	N	N
	Section 2.5.5: nRTS signal abnormally driven low after a protocol violation	A	A
	Section 2.5.6: Start bit detected too soon when sampling for NACK signal from the smartcard	N	N
	Section 2.5.7: Break request can prevent the Transmission Complete flag (TC) from being set	A	A
	Section 2.5.8: Guard time is not respected when data are sent on TXE events	A	A
	Section 2.5.9: nRTS is active while RE or UE = 0	A	A
Section 2.6: DAC peripheral limitations	*Section 2.6.1: Spurious activation of DAC output buffer (PA4 and PA5)*	A	A
Section 2.7: RCC peripheral limitation	*Section 2.7.1: Delay after an RCC peripheral clock enabling*	A	A
Section 2.8: RTC limitation	*Section 2.8.1: Spurious tamper detection when disabling the tamper channel*	N	N
	Section 2.8.2: RTC calendar registers are not locked properly	A	A

2.1 System limitations

2.1.1 Number of RTC backup registers is 20 bytes instead of 80 bytes

Description

The number of RTC backup registers is 20 bytes (five 32-bit registers) instead of 80 bytes.

Workaround

None.

2.1.2 If debugger is connected in JTAG mode and JNRST (PB4) pin configuration is changed, the connection is lost

Description

PB4 is configured by default in alternate function mode after reset.

When the configuration bit changes from an alternate function to an input, analog or GPIO, the reset signal connected to the CPU is tied to '0', and forces a reset on the CPU TAP that stops the debugger connection, even if the pin itself is pulled up.

Only JTAG mode with 4 wires is impacted. Serial Wire Debug (SWD) mode is not impacted.

Workaround

During the debug phase, when the debugger is connected in JTAG mode is used, I/O port PB4 should not be used by the application. If the application needs to use PB4 even during debug phase, the debugger should use Serial Wire Debug (SWD) mode to connect.

2.1.3 Data EEPROM cycling limited to 100 kcycles

Description

The Data EEPROM, which usually supports 300 kcycles, supports only 100 kcycles. All the retention parameter are the same (replacing in the electrical characteristics 300 kcycles with 100 kcycles).

Workaround

None.

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

Description

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

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

Workaround

To avoid this problem, the following sequence should be applied before entering the Standby mode:

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

Note: When applying this workaround, if one of the wakeup sources is still kept high, be aware that the MCU will enter the Standby mode but then wakes up immediately, generating the power reset.

2.1.5 Flash memory wakeup issue when waking up from Stop or Sleep with Flash memory in power-down mode

Description

When an external wakeup event (EXTI) occurs in a narrow time window around low-power mode entry (Stop or Sleep mode with the Flash memory in power-down state), the Flash wakeup time may be increased. As a result, the first data read or the instruction fetch from the Flash memory may be incorrect.

The probability that this issue occurs is very low since it may happen only during a very narrow time window.

Workarounds

Two workarounds are available:

- Do not put the Flash memory module in power-down mode when entering Sleep or Low-power sleep modes.
- Before entering Stop mode by executing a WFI instruction from the RAM, set the RUN_PD bit in the FLASH_ACR register. After exiting from Stop mode, the Flash memory is automatically powered ON and the user can resume program execution from the Flash memory. After wakeup, clear the RUN_PD bit.

2.1.6 Unexpected system reset when waking up from Stop mode with regulator in low-power mode

Description

If the device returns to Run mode after waking up from Stop mode while the internal voltage regulator is configured to switch to low-power mode in Stop mode (LPSDSR=1 in PWR_CR register), an unexpected system reset may occur if the following conditions are met:

- The internal regulator is set to Range 2 or Range 3 before entering Stop mode.
- V_{DD} power supply is below 2.7 V.

The probability that this issue occurs is very low since it may happen only for very narrow supply voltage windows which vary from one device to another.

This reset is internal only and does not affect the NRST pin state and the flags in the control/status register (RCC_CSR).

Workarounds

Two workarounds are possible:

- Enter Stop mode with the internal voltage regulator set to main mode (LPSDSR=0 in PWR_CR).
- Set the internal voltage regulator to Range1 before entering Stop mode.

2.2 IWDG peripheral limitation

2.2.1 RVU and PVU flags are not reset in Stop mode

Description

The RVU and PVU flags of the IWDG_SR register are set by hardware after a write access to the IWDG_RLR and the IWDG_PR registers, respectively. If the Stop mode is entered immediately after the write access, the RVU and PVU flags are not reset by hardware.

Before performing a second write operation to the IWDG_RLR or the IWDG_PR register, the application software must wait for the RVU or PVU flag to be reset. However, since the RVU/PVU bit is not reset after exiting Stop mode, the software goes into an infinite loop and the independent watchdog (IWDG) generates a reset after the programmed timeout period.

Workaround

Wait until the RVU or PVU flag of the IWDG_SR register are reset before entering Stop mode.

2.3 I^2C peripheral limitations

2.3.1 SMBus standard not fully supported

Description

The I^2C 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

The following higher-level mechanisms should be used to verify that a write operation is being performed correctly at the target device:

1. The SMBA pin if supported by the host
2. The alert response address (ARA) protocol
3. The host notify protocol

2.3.2 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 and the Start bit is already set in the I2C_CR2 register. In this case 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 I²C peripheral.

In case of noisy environment in which unwanted bus errors can occur, it is recommended to reset the I2C peripheral by setting the SWRST bit in the I2C_CR2 control register if a BERR is detected while the Start bit is set in the I2C_CR2 register.

No fix is planned for this limitation.

2.3.3 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 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. $t_{SU(STA)}$ minimum value may be 4 µs instead of 4.7 µs.

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.

2.3.4 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 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 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.

2.3.5 10-bit Master mode: new transfer cannot be launched if first part of the address has not been acknowledged by the slave

Description

In Master mode, the master automatically sends a STOP bit when the slave has not acknowledged a byte during the address transmission.

In the 10-bit addressing mode, if the first part of the 10-bit address (corresponding to 10-bit header + 2 MSB) has not been acknowledged by the slave, the STOP bit is sent but the Start bit is not cleared and the master cannot launch a new transfer.

Workaround

When the I2C is configured in 10-bit addressing Master mode and the NACKF status flag is set in the I2C_ISR register while the Start bit is still set in the I2C_CR2 register, then proceed as follows:

1. Wait for the STOP condition detection (STOPF = 1 in I2C_ISR register).
2. Disable the I2C peripheral.
3. Wait for a minimum of 3 APB cycles.
4. Enable the I2C peripheral again.

2.4 SPI/I2S peripheral limitations

2.4.1 In I2S slave mode, WS level must to be set by the external master when enabling the I2S

Description

In slave mode the WS signal level is used only to start the communication. If the I2S (in slave mode) is enabled while the master is already sending the clock and the WS signal level is low (for I2S protocol) or is high (for the LSB or MSB-justified mode), the slave starts communicating data immediately. In this case the master and the slave will be desynchronized throughout the whole communication.

Workaround

The I2S peripheral must be enabled when the external master sets the WS line at:

- High level when the I2S protocol is selected.
- Low level when the LSB or MSB-justified mode is selected.

2.4.2 BSY bit may stay high at the end of a 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 data transaction may be not recognized when the software needs to monitor it (e.g. at the end of session before entering the low-power mode or before direction of 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

When the NSS hardware management is applied and the NSS signal is provided by the master, the end of a transaction can be detected by the NSS polling by the slave.

- If the SPI receiving mode is enabled, the end of a transaction with master can be detected by the corresponding RXNE event signalizing the last data transfer completion.
- In SPI transmit mode, the user can check the BSY under timeout corresponding to the time necessary to complete the last data frame transaction. The timeout should be measured from TXE event signalizing the last data frame transaction start (it is raised once the second bit transaction is ongoing). Either BSY becomes low normally or the timeout expires when the synchronization issue happens.

When upper workarounds are not applicable, the following sequence can be used to prevent the synchronization issue at SPI transmit mode.

1. Write last data to 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 workaround can be used only when CPU has enough performance to disable SPI after the 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 and the data frame is short especially. In this specific case the timeout can be measured from TXE, while calculating a fixed number of CPU clock periods corresponding to the time necessary to complete the data frame transaction.

2.4.3 Wrong CRC calculation when the polynomial is even

Description

When the CRC is enabled, the CRC calculation will be wrong if the polynomial is even.

Workaround

Use odd polynomial.

2.4.4 Wrong CRC transmitted in Master mode with delayed SCK feedback

Description

In transmit transaction of the SPI interface in Master mode with the CRC enabled, the CRC data transmission may be corrupted if the delay of an internal feedback signal derived from the SCK output (further feedback clock) is greater than two APB clock periods. While the data and the CRC bit shifting and the transfer is based on an internal clock, the CRC progressive calculation uses the feedback clock. If the delay of the feedback clock is greater than two APB periods, the transmitted CRC value may get wrong.

The main factors contributing to the delay increase are low V_{DD} level, high temperature, high SCK pin capacitive load and low SCK I/O output speed. The SPI communication speed has no impact.

Workaround

The following workaround can be adopted, jointly or individually:

- Decrease the APB clock speed.
- Configure the IO pad of the SCK pin to be faster.

Table 5 gives the maximum allowable APB frequency versus the GPIOx_OSPEEDR output speed control field setting for the SCK pin, at 30 pF of capacitive load.

2.4.5 SPI CRC may be corrupted when a peripheral connected to the same DMA channel of the SPI finishes its DMA transaction

Description

When the SPI is running with the CRC feature enabled at all the modes, the CRC calculation may be frozen and checked corrupted before the CRCNEXT bit is written. It can happen if a peripheral, mapped on the same DMA channel than the SPI, is doing a DMA transfer whereas the SPI is configured to manage data transfers by software (IT or polling). As a consequence the CRC error flag is unduly raised.

Workaround

There is no known workaround of this conflict. If possible use the DMA for SPI transfers to avoid this conflict.

2.4.6 Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback

Description

In receive transaction, in both I^2S and SPI Master modes, the last bit of the transacted frame is not captured when the signal provided by internal feedback loop from the SCK pin exceeds a critical delay. The lastly transacted bit of the stored data then keeps the value from the pattern received previously. As a consequence, the last receive data bit may be wrong and/or the CRCERR flag can be unduly asserted in the SPI mode if any data under check sum and/or just the CRC pattern is wrongly captured.

In SPI mode, data are synchronous with the APB clock. A delay of up to two APB clock periods can thus be tolerated for the internal feedback delay. The I2S mode is more sensitive than the SPI mode since the SCK clock is not synchronized with the APB clock. In this case, the margin of the internal feedback delay is lower than one APB clock period.

The main factors contributing to the delay increase are low V_{DD} level, high temperature, high SCK pin capacitive load and low SCK I/O output speed. The SPI communication speed has no impact.

Workaround

The following workaround can be adopted, jointly or individually:

- Decrease the APB clock speed.
- Configure the IO pad of the SCK pin to be faster.

Table 5 gives the maximum APB frequency versus the GPIOx_OSPEEDR output speed control field setting for the SCK pin, at 30 pF of capacitive load.

Table 5. Maximum allowable APB frequency at 30 pF load

Setting of OSPEEDR bits [1:0] for the SCK pin	Maximum APB frequency for SPI [MHz]	Maximum APB frequency for I2S [MHz]
Very high (11)	32	32
High (10)	28 (30 if VDD > 1.8 V)	28 (30 if VDD > 1.8 V)
Medium (01)	11 (13 if VDD > 1.8 V)	8 (9 if VDD > 1.8 V)
Low (00)	2	1.5

2.5 USART peripheral limitations

2.5.1 Idle frame is not detected if receiver clock speed is deviated

Description

If the USART receives an idle frame followed by a character, and the clock of the transmitter device is faster than the USART receiver clock, the USART receive signal falls too early when receiving the character start bit, with the result that the idle frame is not detected (IDLE flag is not set).

Workaround

None.

2.5.2 In full duplex mode, the Parity Error (PE) flag can be cleared by writing the data register

Description

In full duplex mode, when the Parity Error flag is set by the receiver at the end of a reception, it may be cleared while transmitting by reading the USART_SR register to check the TXE or TC flags and writing data in the data register.

Consequently, the software receiver can read the PE flag as '0' even if a parity error occurred.

Workaround

The Parity Error flag should be checked after the end of reception and before transmission.

2.5.3 Parity Error (PE) flag is not set when receiving in Mute mode using address mark detection

Description

The USART receiver is in Mute mode and is configured to exit the Mute mode using the address mark detection. When the USART receiver recognizes a valid address with a parity error, it exits the Mute mode without setting the Parity Error flag.

Workaround

None.

2.5.4 Break frame is transmitted regardless of nCTS input line status

Description

When the CTS hardware flow control is enabled (CTSE = 1) and the Send Break bit (SBK) is set, the transmitter sends a break frame at the end of current transmission regardless of nCTS input line status.

Consequently, if an external receiver device is not ready to accept a frame, the transmitted break frame is lost.

Workaround

None.

2.5.5 nRTS signal abnormally driven low after a protocol violation

Description

When the RTS hardware flow control is enabled, the nRTS signal goes high when a data is received. If this data was not read and a new data is sent to the USART (protocol violation), the nRTS signal goes back to low level at the end of this new data.

Consequently, the sender gets the wrong information that the USART is ready to receive further data.

On the USART side, an overrun is detected which indicates that some data has been lost.

Workarounds

Workarounds are required only if the other USART device violates the communication protocol which is not the case in most applications.

Two workarounds can be used:

- After data reception and before reading in the data in the data register, the software takes over the control of the nRTS signal as a GPIO and holds it high as long as needed. If the USART device is not ready, the software holds the nRTS pin high, and releases it when the device is ready to receive new data.
- The time required by the software to read the received data must always be lower than the duration of the second data reception. For example, this can be ensured by treating all the receptions by DMA mode.

2.5.6 Start bit detected too soon when sampling for NACK signal from the smartcard

Description

In the ISO7816, when a character parity error is incorrect, the smartcard receiver shall transmit a NACK error signal at (10.5 +/- 0.2) etu after the character Start bit falling edge. In this case, the USART transmitter should be able to detect correctly the NACK signal by sampling at (11.0 +/-0.2) etu after the character Start bit falling edge.

The USART peripheral used in smartcard mode doesn't respect the (11 +/-0.2) etu timing, and when the NACK falling edge arrives at 10.68 etu or later, the USART might misinterpret this transition as a Start bit even if the NACK is correctly detected.

Workaround

None.

2.5.7 Break request can prevent the Transmission Complete flag (TC) from being set

Description

After the end of transmission of a data (D1), the Transmission Complete (TC) flag will not be set in the following conditions:

- CTS hardware flow control is enabled.
- D1 is being transmitted.
- A break transfer is requested before the end of D1 transfer.
- nCTS is de-asserted before the end of transfer of D1.

Workaround

If the application needs to detect the end of transfer of the data, the break request should be done after making sure that the TC flag is set.

2.5.8 Guard time is not respected when data are sent on TXE events

Description

In smartcard mode, when sending a data on TXE event, the programmed guard time is not respected. The data written in the data register are transferred on the bus without waiting for the completion of the guardtime duration corresponding to the previous transmitted data.

Workaround

Write the data after TC is set because in smartcard mode, the TC flag is set at the end of the guard time duration.

2.5.9 nRTS is active while RE or UE = 0

Description

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

Workaround

Configure the I/O used for nRTS as alternate function after setting the UE and RE bits.

2.6 DAC peripheral limitations

2.6.1 Spurious activation of DAC output buffer (PA4 and PA5)

Description

The high speed signal with a falling edge and a slope higher than 7V/us can cause a spurious activation of the DAC buffer if applied on the pins PA4 and PA5. Such spurious activation can happen regardless of the GPIO configuration and it may pull a current from the signal source up to 1 mA in the typical conditions. As a result of this unintended DAC buffer pulling, a shape of the falling edge may change and cause a shifted timing of the digital signal applied on the pin.

Workarounds

- Use another IO with an equivalent function.
- Reduce the slope of the signal applied on the affected pins.

2.7 RCC peripheral limitation

2.7.1 Delay after an RCC peripheral clock enabling

Description

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write to registers.

This delay depends on the peripheral's mapping:

- If the peripheral is mapped on AHB: the delay should be equal to 2 AHB cycles.
- If the peripheral is mapped on APB: the delay should be equal to 1 + (AHB/APB prescaler) cycles.

Workarounds

1. Use the DSB instruction to stall the Cortex-M CPU pipeline until the instruction is completed.
2. Insert "n" NOPs between the RCC enable bit write and the peripheral register writes (n = 2 for AHB peripherals, n = 1 + AHB/APB prescaler in case of APB peripherals).

2.8 RTC limitation

2.8.1 Spurious tamper detection when disabling the tamper channel

Description

If the tamper detection is configured for detection on the falling edge event (TAMPFLT=00 and TAMPxTRG=1) and if the tamper event detection is disabled when the tamper pin is at high level, a false tamper event is detected.

Workaround

None.

2.8.2 RTC calendar registers are not locked properly

Description

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

Workaround

1. Use BYPSHAD = 1 mode (Bypass shadow registers), or
2. If BYPSHAD = 0, read SSR again after reading SSR/TR/DR to confirm that SSR is still the same, otherwise read the values again.

3 Revision history

Table 6. Document revision history

Date	Revision	Changes
07-Jan-2014	1	Initial release.
27-Jan-2014	2	Update Table 4 adding link to *Section 2.1.4: Wakeup sequence from Standby mode when using more than one wakeup source*.
18-Feb-2014	3	Modified Title to STM32L15xx6/8/B-A. Updated *Table 1: Device summary*.
13-Mar-2014	4	Updated document (Title, Tables) adding STM32L100x6/8/B-A Devices.
15-Oct-2014	5	Added *Section 2.4.2: BSY bit may stay high at the end of a data transfer in Slave mode*.
18-May-2015	6	Removed appendix A section: all package markings put in the corresponding datasheets. Added USART limitations: - *Section 2.5.6: Start bit detected too soon when sampling for NACK signal from the smartcard*. - *Section 2.5.7: Break request can prevent the Transmission Complete flag (TC) from being set*. - *Section 2.5.8: Guard time is not respected when data are sent on TXE events*. - *Section 2.5.9: nRTS is active while RE or UE = 0*. Added SPI limitation: - *Section 2.4.3: Wrong CRC calculation when the polynomial is even*. Added RTC limitation: - *Section 2.8.1: Spurious tamper detection when disabling the tamper channel*.

Table 6. Document revision history (continued)

Date	Revision	Changes
16-Nov-2015	7	Updated cover adding revision Y. Updated *Table 4: Summary of silicon limitations* adding revision Y with same limitations as revision Z. Added 2 system limitations fixed by revision Y: - *Section 2.1.5: Flash memory wakeup issue when waking up from Stop or Sleep with Flash memory in power-down mode*. - *Section 2.1.6: Unexpected system reset when waking up from Stop mode with regulator in low-power mode*.
24-Nov-2016	8	Added I^2C limitation: – *Section 2.3.5: 10-bit Master mode: new transfer cannot be launched if first part of the address has not been acknowledged by the slave*. Added RTC limitation: – *Section 2.8.2: RTC calendar registers are not locked properly*. Added DAC limitation: – *Section 2.6.1: Spurious activation of DAC output buffer (PA4 and PA5)*. SPI/I2S limitations: – Updated *Section 2.4.2: BSY bit may stay high at the end of a data transfer in Slave mode*. – Added *Section 2.4.4: Wrong CRC transmitted in Master mode with delayed SCK feedback*. – Added *Section 2.4.5: SPI CRC may be corrupted when a peripheral connected to the same DMA channel of the SPI finishes its DMA transaction*. – Added *Section 2.4.6: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback*.