STM32L151x6/8/B-A STM32L152x6/8/B-A

Errata sheet

STM32L151x6/8/B-A and STM32L152x6/8/B-A device errata

Applicability

This document applies to the part numbers of STM32L151x6/8/B-A and STM32L152x6/8/B-A 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 RM0038.

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**

Reference	Part numbers
STM32L151x6/8/B-A	STM32L151VB-A, STM32L151RB-A, STM32L151R8-A, STM32L151C8-A, STM32L151R6-A, STM32L151C6-A, STM32L151CB-A, STM32L151V8-A
STM32L152x6/8/B-A	STM32L152VB-A, STM32L152RB-A, STM32L152R8-A, STM32L152C8-A, STM32L152R6-A, STM32L152C6-A, STM32L152CB-A, STM32L152V8-A

Table 2. **Device variants**

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32L151x6/8/B-A, STM32L152x6/8/B-A	A	0x1000
	Z	0x1018
	1, Y	0x1038

1. *Refer to the device datasheet for how to identify this code on different types of package.*
2. *REV_ID[15:0] bitfield of DBG_IDCODE register.*

1 Summary of device errata

The following table gives a quick reference to the STM32L151x6/8/B-A and STM32L152x6/8/B-A device limitations and their status:

A = limitation present, workaround available

N = limitation present, no workaround available

P = limitation present, partial workaround available

“-” = limitation absent

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

Table 3. **Summary of device limitations**

Function	Section	Limitation	Status	
			Rev. A,Z	Rev. 1,Y
Core	2.1.1	Cortex®-M3 LDRD with base in list may result in incorrect base register when interrupted or faulted	A	A
	2.1.2	Cortex®-M3 event register is not set by interrupts and debug	A	A
	2.1.3	Interrupted loads to SP can cause erroneous behavior	A	A
	2.1.4	SVC and BusFault/MemManage may occur out of order	A	A
System	2.2.1	Unexpected system reset when waking up from Stop mode with regulator in low-power mode	A	-
	2.2.2	Wake-up sequence from Standby mode fails when using more than one wake-up source	A	A
	2.2.3	Flash memory wake-up issue when waking up from Stop or Sleep with flash memory in power-down mode	A	-
	2.2.4	Delay after enabling an RCC peripheral clock	A	A
	2.2.5	Data EEPROM cycling limited to 100 kcycles	N	N
	2.2.6	LSE crystal oscillator may be disturbed by transitions on PC13	N	N
GPIO	2.3.1	If debugger is connected in JTAG mode and JNTRST (PB4) pin configuration is changed, the connection is lost	A	A
DMA	2.4.1	DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	A	A
DAC	2.5.1	DMA request not automatically cleared by clearing DMAEN	A	A
	2.5.2	DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge	N	N
	2.5.3	Spurious activation of DAC output buffer (PA4 and PA5)	A	A
TIM	2.6.1	PWM re-enabled in automatic output enable mode despite of system break	P	P
	2.6.3	Consecutive compare event missed in specific conditions	N	N
	2.6.4	Output compare clear not working with external counter reset	P	P
IWDG	2.7.1	RVU flag not reset in Stop	A	A
	2.7.2	PVU flag not reset in Stop	A	A
	2.7.3	RVU flag not cleared at low APB clock frequency	A	A
	2.7.4	PVU flag not cleared at low APB clock frequency	A	A
RTC	2.8.1	Spurious tamper detection when disabling the tamper channel	N	N

Function	Section	Limitation	Status	
			Rev. A,Z	Rev. 1,Y
RTC	2.8.2	RTC calendar registers are not locked properly	A	A
	2.8.3	RTC interrupt can be masked by another RTC interrupt	A	A
	2.8.4	Calendar initialization may fail in case of consecutive INIT mode entry	A	A
	2.8.5	Alarm flag may be repeatedly set when the core is stopped in debug	N	N
	2.8.6	A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF	N	N
	2.8.7	Number of RTC backup registers is 20 bytes instead of 80 bytes	N	N
I2C	2.9.1	SMBus standard not fully supported	A	A
	2.9.2	Wrong behavior of I2C peripheral in master mode after misplaced STOP	A	A
	2.9.3	Violation of I2C "setup time for repeated START condition" parameter	A	A
	2.9.4	In I2C slave “NOSTRETCH” mode, underrun errors may not be detected and may generate bus errors	A	A
	2.9.5	In 10-bit master mode, new transfer cannot be launched if first part of the address has not been acknowledged by the slave	A	A
USART	2.10.1	Communication parameters reprogramming after ATR in Smartcard mode when SCLK is used to clock the card	A	A
	2.10.2	Non-compliant sampling for NACK signal from smartcard	N	N
	2.10.3	Break request preventing TC flag from being set	A	A
	2.10.4	RTS is active while RE = 0 or UE = 0	A	A
	2.10.5	Idle frame not detected if the receiver clock speed is deviated	N	N
	2.10.6	Parity error flag (PE) cleared by writing the data register in full-duplex mode	A	A
	2.10.7	Parity Error (PE) flag not set when receiving in mute mode using address mark detection	N	N
	2.10.8	Break frame is transmitted regardless of nCTS input line status	N	N
	2.10.9	nRTS signal abnormally driven low after a protocol violation	A	A
	2.10.10	Start bit detected too soon when sampling for NACK signal from the smartcard	N	N
	2.10.11	Guard time is not respected when data are sent on TXE events	A	A
SPI/I2S	2.11.1	BSY bit may stay high when SPI is disabled	A	A
	2.11.2	BSY bit may stay high at the end of data transfer in slave mode	A	A
	2.11.3	Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback	A	A
	2.11.4	SPI CRC corruption upon DMA transaction completion by another peripheral	P	P
	2.11.5	In I^{2}S slave mode, enabling I2S while WS is active causes desynchronization	A	A
	2.11.6	Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters	A	A
	2.11.8	Anticipated communication upon SPI transit from slave receiver to master	A	A
	2.11.9	Wrong CRC calculation when the polynomial is even	A	A
	2.11.10	Wrong CRC transmitted in master mode with delayed SCK feedback	A	A
	2.11.11	In I2S slave mode, WS level must be set by the external master when enabling the I2S	A	A

Function	Section	Limitation	Status	
			Rev. A,Z	Rev. 1,Y
USB	2.12.2	False wake-up detection for last K-state not terminated by EOP or reset before suspend	A	A
	2.12.3	ESOF interrupt timing desynchronized after resume signaling	A	A
	2.12.4	Incorrect CRC16 in the memory buffer	N	N
	2.12.5	Buffer description table update completes after CTR interrupt triggers	A	A

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

Table 4. **Summary of device documentation errata**

Function	Section	Documentation erratum
DMA	2.4.2	Byte and half-word accesses not supported
TIM	2.6.2	TRGO and TRGO2 trigger output failure
SPI/I2S	2.11.7	CRC error in SPI slave mode if internal NSS changes before CRC transfer
USB	2.12.1	Possible packet memory overrun/underrun at low APB frequency

2 Description of device errata

The following sections describe the errata 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

Reference manual and errata notice for the Arm® Cortex®-M3 core is available from http://infocenter.arm.com. Only applicable information from the Arm errata notice is replicated in this document.

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

Description

This limitation is registered under Arm® ID number 602117 and classified into “Category 2”.

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 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, they 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).

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

Description

This limitation is registered under Arm® ID number 563915 and classified into “Category 2”.

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

Workaround

Use external events instead of interrupts to wake up the core from WFE by configuring an external or internal EXTI line in event mode.

2.1.3 Interrupted loads to SP can cause erroneous behavior

Description

This limitation is registered under Arm® ID number 752419 and classified into “Category 2”.

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

The affected instructions are:

1. LDR SP,[Rn],#imm
2. LDR SP,[Rn,#imm]!

3. LDR SP,[Rn,#imm]
4. LDR SP,[Rn]
5. LDR SP,[Rn,Rm]

Workaround

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

Both issues may be worked around by replacing the direct load to the stack-pointer, with an intermediate load to a general-purpose register followed by a move to the stack-pointer.

Example: the following instruction "LDR SP, [R0]" can be replaced by

“LDR R2,[R0]

MOV SP,R2 "

2.1.4 SVC and BusFault/MemManage may occur out of order

Description

This limitation is registered under Arm® ID number 740455 and classified into “Category 2”.

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 does not return to the return address that has been stacked for the SVC exception and the instruction access after the SVC faults. 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.

2.2 System

2.2.1 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).

Workaround

Apply one of the following measures:

- 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.

2.2.2 Wake-up sequence from Standby mode fails when using more than one wake-up source

Description

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

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

Workaround

Apply the following sequence before entering Standby mode:

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

Note: When applying this measure, if one of the wake-up sources is still kept high, be aware that the microcontroller enters Standby mode but then wakes up immediately, generating the power reset.

2.2.3 Flash memory wake-up issue when waking up from Stop or Sleep with flash memory in power-down mode

Description

When an external wake-up 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 memory wake-up time may be increased. As a result, the first data read or 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.

Workaround

Apply one of the following measures:

- 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 RAM, set the RUN_PD bit of the FLASH_ACR register. After exiting from Stop mode, the flash memory is automatically powered on and the user can resume program execution from flash memory. After wake-up, clear the RUN_PD bit.

2.2.4 Delay after enabling an RCC peripheral clock

Description

When enabling an RCC peripheral clock, there is a delay between the clock enable and the effective peripheral enabling. The user should take this into account to manage the peripheral read/write to registers.

This delay depends on the peripheral 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.

Workaround

Apply one of the following measures:

- Use the DSB instruction to stall the Cortex-M CPU pipeline until the instruction is completed.
- 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).

2.2.5 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.

2.2.6 LSE crystal oscillator may be disturbed by transitions on PC13

Description

On LQFP and UFQFPN packages, the LSE crystal oscillator clock frequency can be incorrect when PC13 is toggling in input or output (for example when used for RTC_OUT1). The external clock input (LSE bypass) is not impacted by this limitation. The WLCSP and UFBGA packages are not impacted by this limitation.

Avoid toggling PC13 when LSE is used on LQFP and UFQFPN packages.

Workaround

None.

2.3 GPIO

2.3.1 If debugger is connected in JTAG mode and JNTRST (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 four wires is impacted. The serial wire debug (SWD) mode is not impacted.

Workaround

Do not use the PB4 I/O port during the debug phase and when the debugger connected in JTAG mode is used. If the application needs to use PB4 even during the debug phase, use the serial wire debug (SWD) mode to connect the debugger.

2.4 DMA

2.4.1 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 when the channel is active.

Workaround

Do not clear GIFx flags when the channel is active. Instead, use HTIFx, TCIFx, and TEIFx specific event flags and their corresponding clear bits.

2.4.2 Byte and half-word accesses not supported

Description

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

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

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

Workaround

No application workaround is required.

2.5 DAC

2.5.1 DMA request not automatically cleared by clearing DMAEN

Description

Upon an attempt to stop a DMA-to-DAC transfer, the DMA request is not automatically cleared by clearing the DAC channel bit of the DAC_CR register (DMAEN) or by disabling the DAC clock.

If the application stops the DAC operation while the DMA request is pending, the request remains pending while the DAC is reinitialized and restarted, with the risk that a spurious DMA request is serviced as soon as the DAC is enabled again.

Workaround

Apply the following sequence to stop the current DMA-to-DAC transfer and restart the DAC:

1. Check if DMAUDR bit is set in DAC_CR.
2. Clear the DAC channel DMAEN bit.
3. Disable the DAC clock.
4. Reconfigure the DAC, DMA and the triggers.
5. Restart the application.

2.5.2 DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge

Description

When the DAC channel operates in DMA mode (DMAEN of DAC_CR register set), the DMA channel underrun flag (DMAUDR of DAC_SR register) fails to rise upon an internal trigger detection if that detection occurs during the same clock cycle as a DMA request acknowledge. As a result, the user application is not informed that an underrun error occurred.

This issue occurs when software and hardware triggers are used concurrently to trigger DMA transfers.

Workaround

None.

2.5.3 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 shifted timing of the digital signal applied on the pin.

Workaround

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

2.6 TIM

2.6.1 PWM re-enabled in automatic output enable mode despite of system break

Description

In automatic output enable mode (AOE bit set in TIMx_BDTR register), the break input can be used to do a cycle-by-cycle PWM control for a current mode regulation. A break signal (typically a comparator with a current threshold) disables the PWM output(s) and the PWM is re-armed on the next counter period.

However, a system break (typically coming from the CSS Clock security System) is supposed to stop definitively the PWM to avoid abnormal operation (for example with PWM frequency deviation).

In the current implementation, the timer system break input is not latched. As a consequence, a system break indeed disables the PWM output(s) when it occurs, but PWM output(s) is (are) re-armed on the following counter period.

Workaround

Preferably, implement control loops with the output clear enable function (OCxCE bit in the TIMx_CCMR1/CCMR2 register), leaving the use of break circuitry solely for internal and/or external fault protection (AOE bit reset).

2.6.2 TRGO and TRGO2 trigger output failure

Description

Some reference manual revisions may omit the following information.

The timers can be linked using ITRx inputs and TRGOx outputs. Additionally, the TRGOx outputs can be used as triggers for other peripherals (for example ADC). Since this circuitry is based on pulse generation, care must be taken when initializing master and slave peripherals or when using different master/slave clock frequencies:

- If the master timer generates a trigger output pulse on TRGOx prior to have the destination peripheral clock enabled, the triggering system may fail.
- If the frequency of the destination peripheral is modified on-the-fly (clock prescaler modification), the triggering system may fail.

As a conclusion, the clock of the slave timer or slave peripheral must be enabled prior to receiving events from the master timer, and must not be changed on-the-fly while triggers are being received from the master timer.

This is a documentation issue rather than a product limitation.

Workaround

No application workaround is required or applicable as long as the application handles the clock as indicated.

2.6.3 Consecutive compare event missed in specific conditions

Description

Every match of the counter (CNT) value with the compare register (CCR) value is expected to trigger a compare event. However, if such matches occur in two consecutive counter clock cycles (as consequence of the CCR value change between the two cycles), the second compare event is missed for the following CCR value changes:

- in edge-aligned mode, from ARR to 0:
  - first compare event: CNT = CCR = ARR
  - second (missed) compare event: CNT = CCR = 0

- in center-aligned mode while up-counting, from ARR-1 to ARR (possibly a new ARR value if the period is also changed) at the crest (that is, when TIMx_RCR = 0):
  - first compare event: CNT = CCR = (ARR-1)
  - second (missed) compare event: CNT = CCR = ARR
- in center-aligned mode while down-counting, from 1 to 0 at the valley (that is, when TIMx_RCR = 0):
  - first compare event: CNT = CCR = 1
  - second (missed) compare event: CNT = CCR = 0

This typically corresponds to an abrupt change of compare value aiming at creating a timer clock single-cycle-wide pulse in toggle mode.

As a consequence:

- In toggle mode, the output only toggles once per counter period (squared waveform), whereas it is expected to toggle twice within two consecutive counter cycles (and so exhibit a short pulse per counter period).
- In center mode, the compare interrupt flag does note rise and the interrupt is not generated.

Note: The timer output operates as expected in modes other than the toggle mode.

Workaround

None.

2.6.4 Output compare clear not working with external counter reset

Description

The output compare clear event (ocref_clr) is not correctly generated when the timer is configured in the following slave modes: Reset mode, Combined reset + trigger mode, and Combined gated + reset mode.

The PWM output remains inactive during one extra PWM cycle if the following sequence occurs:

1. The output is cleared by the ocref_clr event.
2. The timer reset occurs before the programmed compare event.

Workaround

Apply one of the following measures:

- Use BKIN (or BKIN2 if available) input for clearing the output, selecting the Automatic output enable mode (AOE = 1).
- Mask the timer reset during the PWM ON time to prevent it from occurring before the compare event (for example with a spare timer compare channel open-drain output connected with the reset signal, pulling the timer reset line down).

2.7 IWDG

2.7.1 RVU flag not reset in Stop

Description

Successful write to the IWDG_RLR register raises the RVU flag and prevents further write accesses to the register until the RVU flag is automatically cleared by hardware. However, if the device enters Stop mode while the RVU flag is set, the hardware never clears that flag, and writing to the IWDG_RLR register is no longer possible.

Workaround

Ensure that the RVU flag is cleared before entering Stop mode.

2.7.2 PVU flag not reset in Stop

Description

Successful write to the IWDG_PR register raises the PVU flag and prevents further write accesses to the register until the PVU flag is automatically cleared by hardware. However, if the device enters Stop mode while the PVU flag is set, the hardware never clears that flag, and writing to the IWDG_PR register is no longer possible.

Workaround

Ensure that the PVU flag is cleared before entering Stop mode.

2.7.3 RVU flag not cleared at low APB clock frequency

Description

Successful write to the IWDG_RLR register raises the RVU flag and prevents further write accesses to the register until the RVU flag is automatically cleared by hardware. However, at APB clock frequency lower than twice the IWDG clock frequency, the hardware never clears that flag, and writing to the IWDG_RLR register is no longer possible.

Workaround

Set the APB clock frequency higher than twice the IWDG clock frequency.

2.7.4 PVU flag not cleared at low APB clock frequency

Description

Successful write to the IWDG_PR register raises the PVU flag and prevents further write accesses to the register until the PVU flag is automatically cleared by hardware. However, at APB clock frequency lower than twice the IWDG clock frequency, the hardware never clears that flag, and writing to the IWDG_PR register is no longer possible.

Workaround

Set the APB clock frequency higher than twice the IWDG clock frequency.

2.8 RTC

2.8.1 Spurious tamper detection when disabling the tamper channel

Description

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

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

Apply one of the following measures:

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

2.8.3 RTC interrupt can be masked by another RTC interrupt

Description

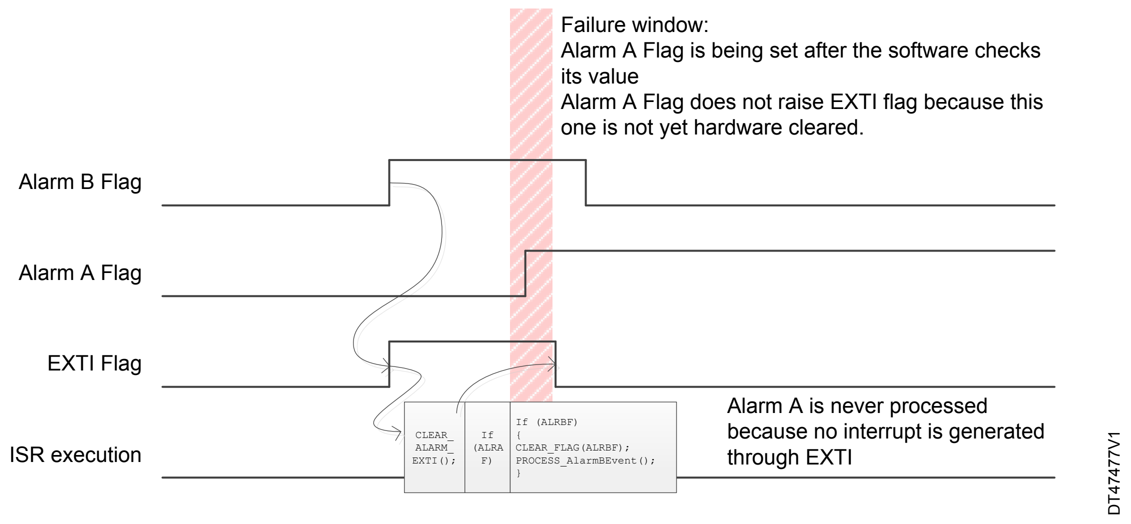
One RTC interrupt request can mask another RTC interrupt request if they share the same EXTI configurable line. For example, interrupt requests from Alarm A and Alarm B or those from tamper and timestamp events are OR-ed to the same EXTI line (refer to the *EXTI line connections* table in the *Extended interrupt and event controller (EXTI)* section of the reference manual).

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

Alarm interrupt service routine:

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

Figure 1. Masked RTC interrupt


Failure window:
Alarm A Flag is being set after the software checks its value
Alarm A Flag does not raise EXTI flag because this one is not yet hardware cleared.
Alarm B Flag
Alarm A Flag
EXTI Flag
ISR execution
Alarm A is never processed because no interrupt is generated through EXTI
DT47477V1

```
CLEAR_
ALARM_
EXTI();
```

```
If
(ALRA
F)
```

```
If (ALRBF)
{
CLEAR_FLAG(ALRBF);
PROCESS_AlarmBEvent();
}
```

Workaround

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

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

2.8.4 Calendar initialization may fail in case of consecutive INIT mode entry

Description

If the INIT bit of the RTC_ISR register is set between one and two RTCCLK cycles after being cleared, the INITF flag is set immediately instead of waiting for synchronization delay (which should be between one and two RTCCLK cycles), and the initialization of registers may fail.

Depending on the INIT bit clearing and setting instants versus the RTCCLK edges, it can happen that, after being immediately set, the INITF flag is cleared during one RTCCLK period then set again. As writes to calendar registers are ignored when INITF is low, a write during this critical period might result in the corruption of one or more calendar registers.

Workaround

After existing 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.*

2.8.5 Alarm flag may be repeatedly set when the core is stopped in debug

Description

When the core is stopped in debug mode, the clock is supplied to subsecond RTC alarm downcounter even when the device is configured to stop the RTC in debug.

As a consequence, when the subsecond counter is used for alarm condition (the MASKSS[3:0] bitfield of the RTC_ALRMASSR and/or RTC_ALRMBSSR register set to a non-zero value) and the alarm condition is met just before entering a breakpoint or printf, the ALRAF and/or ALRBF flag of the RTC_SR register is repeatedly set by hardware during the breakpoint or printf, which makes any attempt to clear the flag(s) ineffective.

Workaround

None.

2.8.6 A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF

Description

With the timestamp on tamper event enabled (TAMPTS bit of the RTC_CR register set), a tamper event is ignored if it occurs:

- within four APB clock cycles after setting the CTSF bit of the RTC_SCR register to clear the TSF flag, while the TSF flag is not yet effectively cleared (it fails to set the TSOVF flag)
- within two ck_apre cycles after setting the CTSF bit of the RTC_SCR register to clear the TSF flag, when the TSF flag is effectively cleared (it fails to set the TSF flag and timestamp the calendar registers)

Workaround

None.

2.8.7 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.

2.9 I2C

2.9.1 SMBus standard not fully supported

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

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

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

2.9.2 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. This can happen in the following conditions:

- If a void message is received (START condition immediately followed by a STOP): the BERR (bus error) flag is not set, and the I2C peripheral is not able to send a START condition on the bus after writing to the START bit in the I2C_CR2 register.
- In the other cases of a misplaced STOP, the BERR flag is set in the IC2_CR2 register. If the START bit is already set in I2C_CR2, the START condition is not correctly generated on the bus and can create bus errors.

Workaround

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

2.9.3 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 $T_{su:sta}$ in the I2C specifications) may be slightly violated when the I2C operates in Master Standard mode at a frequency ranging from 88 to 100 kHz. $t_{SU(STA)}$ minimum value may be 4 µs instead of 4.7 µs.

The issue occurs under the following conditions:

- The I2C peripheral operates in master standard mode at a frequency ranging from 88 to 100 kHz (no issue in fast mode), 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.

2.9.4 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:

- The I2C peripheral operates In Slave transmit mode with clock stretching disabled (NOSTRETCH = 1), 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.

2.9.5 In 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 three APB cycles.
4. Enable the I2C peripheral again.

2.10 USART

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

Description

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

Workaround

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

2.10.2 Non-compliant sampling for NACK signal from smartcard

Description

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

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

Workaround

None.

2.10.3 Break request preventing TC flag from being set

Description

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

- CTS hardware flow control is enabled
- D1 transmission is in progress

- a break transfer is requested before the end of D1 transfer
- CTS is de-asserted before the end of D1 transfer

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

Workaround

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

2.10.4 RTS is active while RE = 0 or UE = 0

Description

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

Workaround

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

2.10.5 Idle frame not detected if the 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.

2.10.6 Parity error flag (PE) cleared by writing the data register in full-duplex mode

Description

When a parity error occurs in reception mode, the PE flag of USART_ISR register is set to 1 at the end of the reception. It can then be cleared by the application by setting to 1 PECF bit of USART_ICR register.
However, if a parity error occurs during data reception in full-duplex mode, PE may be cleared during transmission by reading the USART_ISR register to check the TXE or TC flags and writing data in the data register. As a result, the receiver software may read the PE flag at 0 even if a parity error occurred.

Workaround

Check the Parity Error flag after the end of reception and before transmission.

2.10.7 Parity Error (PE) flag not set when receiving in mute mode using address mark detection

Description

When the USART receiver is in mute mode, configured to exit mute mode using the address mark detection, and it recognizes a valid address with a parity error, the receiver exits mute mode without setting the Parity Error flag (PE) in the USART_ISR register.

Workaround

None.

2.10.8 Break frame is transmitted regardless of nCTS input line status

Description

When CTS hardware flow control is enabled (CTSE = 1) and the send break request bit (SBKRQ) is set in USART_RQR register, the transmitter sends a break frame at the end of the current transmission regardless of nCTS input line status. As a result, the transmitted break frame is lost if the external receiver device is not ready to accept a frame.

Workaround

None.

2.10.9 nRTS signal abnormally driven low after a protocol violation

Description

When 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.

The sender consequently gets the wrong information that the USART is ready to receive further data.

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

Workaround

Two workarounds are possible. They are required only if the other USART device has violated the protocol. In most systems (no limitation on the other device), the USART works fine and no workaround is needed.

Workaround 1

After data reception and before reading the data in the data register, take control of the nRTS pin using the GPIO registers and keeps it high as long as needed.

If the USART is not ready and that further received data from the other device may be discarded, keep the nRTS pin high.

Release it when the USART is ready to continue reception.

Workaround 2

Ensure that the received data is always read in a time window less than the duration of the second data reception. One solution is to handle all data reception by DMA.

2.10.10 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 does not 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

Apply one of the following measures:

- Use the DSB instruction to stall the Cortex-M CPU pipeline until the instruction is completed.
- 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).

2.10.11 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 i.e. the data written in the data register is 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.

2.11 SPI/I2S

2.11.1 BSY bit may stay high when SPI is disabled

Description

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

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

Workaround

When the SPI operates in:

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

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

Description

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

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

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

Workaround

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

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

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

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

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

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

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

Description

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

In SPI mode, data are synchronous with the APB clock. A delay of up to two APB clock periods can thus be tolerated for the internal feedback delay. The I^{2}S mode is more sensitive than the SPI mode since the SCK clock is not synchronized with the APB. In this case, margin of the internal feedback delay is lower than one APB clock period.

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

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

Table 5. Maximum allowable APB frequency at 30 pF load

GPIOx_OSPEEDR[1:0] for SCK pin	Max. APB frequency for SPI mode [MHz]	Max. APB frequency for I^{2}S mode [MHz]
11 (very high)	32	32
10 (high)	28 (30 at V_{DD} > 1.8 V)	28 (30 at V_{DD} > 1.8 V)
01 (medium)	11 (13 at V_{DD} > 1.8 V)	8 (9 at V_{DD} > 1.8 V)
00 (low)	2	1.5

Workaround

The following measures can be adopted, jointly or individually:

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

2.11.4 SPI CRC corruption upon DMA transaction completion by another peripheral

Description

When the following conditions are all met:

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

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

Workaround

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

2.11.5 In I^{2}S slave mode, enabling I2S while WS is active causes desynchronization

Description

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

Workaround

Enable I2S peripheral while the WS line is at:

- high level, for I^{2}S protocol.
- low level, for LSB- or MSB-justified mode.

2.11.6 Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters

Description

When SPI is handled by DMA in full-duplex master or slave mode with CRC enabled, the CRC computation may temporarily freeze for the ongoing frame, which results in corrupted CRC.

This happens when the receive counter reaches zero upon the receipt of the CRC pattern (as the receive counter was set to a value greater, by CRC length, than the transmit counter). An internal signal dedicated to receive-only mode is left unduly pending. Consequently, the signal can cause the CRC computation to freeze during a next transaction in which DMA TXE event service is accidentally delayed (for example, due to DMA servicing a request from another channel).

Workaround

Apply one of the following measures prior to each full-duplex SPI transaction:

- Set the DMA transmission and reception data counters to equal values. Upon the transaction completion, read the CRC pattern out from RxFIFO separately by software.
- Reset the SPI peripheral via peripheral reset register.

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

Description

Some reference manual revisions may omit the information that the device operating as SPI slave must be configured in software NSS control if the SPI master pulses the NSS (for (for example in NSS pulse mode).

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

This is a documentation error rather than a product limitation.

Workaround

No application workaround is required as long as the device operating as SPI slave is duly configured in software NSS control.

2.11.8 Anticipated communication upon SPI transit from slave receiver to master

Description

Regardless of the master mode configured, the communication clock starts upon setting the MSTR bit even though the SPI is disabled, if transiting from receive-only (RXONLY = 1) or half-duplex receive (BIDIMODE = 1 and BIDIOE = 0) slave mode to master mode.

Workaround

Apply one of the following measures:
- Before transiting to master mode, hardware-reset the SPI via the reset controller.
- Set the MSTR and SPE bits of the SPI configuration register simultaneously, which forces the immediate start of the communication clock. In transmitter configuration, load the data register in advance with the data to send.

2.11.9 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.

2.11.10 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 workarounds can be used, jointly or individually:
- Decrease the APB clock speed.
- Configure the I/O pad of the SCK pin to be faster.

Section 2.11.3: Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback 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.

2.11.11 In I2S slave mode, WS level must 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.

2.12 USB

2.12.1 Possible packet memory overrun/underrun at low APB frequency

Description

Some data sheet and/or reference manual revisions may omit the information that 10 MHz minimum APB clock frequency is required to avoid USB data overrun/underrun issues.

Operating the USB peripheral with lower APB clock frequency may lead to:

- Overrun for *out* transactions - the USB peripheral fails to store the received data into the PBM before the next byte is received on the USB (PBM overrun). The USB cell detects an internal error condition, discards the last received byte, stops writing into the PBM, sends no acknowledge (forcing the host to retry the transaction), and informs the application by setting the PMAOVR flag/interrupt.
- Underrun for *in* transactions - the USB peripheral fails to read from the PBM the next byte to transmit before the transmission of the previous one is completed on the USB. The USB cell detects an internal error condition, stops reading from PBM, generates a bit stuffing error on the USB (forcing the host to retry the transaction), and informs the application by setting the PMAOVR flag/interrupt.

This is a documentation issue rather than a device limitation.

Workaround

No application workaround is required if the minimum APB clock frequency of 10 MHz is respected.

2.12.2 False wake-up detection for last K-state not terminated by EOP or reset before suspend

Description

If a USB K-state is detected outside of a suspend state, the device generates *RESUME RECEIVED* signal and keeps it pending until:

- an EOP is detected on the USB, or
- a USB reset is detected on the USB, or
- a software reset is generated by setting the FRES bit.

It may happen that none of the above events occurs before the USB peripheral sets the suspend interrupt (after 3 ms idling), and the software decides to enter Suspend mode (by setting the FSUSP bit). The USB peripheral then immediately generates a wake-up interrupt (WKUP = 1) because it still detects the pending *RESUME RECEIVED* signal. This has a negative impact to the device power consumption.

A workaround is available for applications requiring to minimize the power consumption.

Workaround

Apply software reset prior to setting the FSUSP bit. This clears any spurious *RESUME RECEIVED* condition.

2.12.3 ESOF interrupt timing desynchronized after resume signaling

Description

Upon signaling resume, the device is expected to allow full 3 ms of time to the host or hub for sending the initial SOF (start of frame) packet, without triggering SUSP interrupt. However, the device only allows two full milliseconds and unduly triggers SUSP interrupt if it receives the initial packet within the third millisecond.

Workaround

When the device initiates resume (remote wake-up), mask the SUSP interrupt by setting the SUSPM bit for 3 ms, then unmask it by clearing SUSPM.

2.12.4 Incorrect CRC16 in the memory buffer

Description

Memory buffer locations are written starting from the address contained in the ADDRn_RX for a number of bytes corresponding to the received data packet length, CRC16 inclusive (that is, data payload length plus two bytes), or up to the last allocated memory location defined by BL_SIZE and NUM_BLOCK, whichever comes first. In the former case, the CRC16 checksum is written wrongly, with its least significant byte going to both memory buffer byte locations expected to receive the least and the most significant bytes of the checksum.

Although the checksum written in the memory buffer is wrong, the underlying CRC checking mechanism in the USB peripheral is fully functional.

Workaround

Ignore the CRC16 data in the memory buffer.

2.12.5 Buffer description table update completes after CTR interrupt triggers

Description

During OUT transfers, the correct transfer interrupt (CTR) is triggered a little before the last USB SRAM accesses have completed. If the software responds quickly to the interrupt, the full buffer contents may not be correct.

Workaround

Software should ensure that a small delay is included before accessing the SRAM contents. This delay should be 800 ns in Full Speed mode and 6.4 µs in Low Speed mode.

Important security notice

Revision history

Table 6. **Document revision history**

Date	Version	Changes
07-Jan-2014	1	Initial release.
27_Jan-2014	2	Updated 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.
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 I2C 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.
18-Mar-2022	9	Removed STM32L100xx-A part numbers. Added Silicon revisions A and 1. Document restructured.

Date	Version	Changes
		Added DMA errata: • DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear (2.4.1) • Byte and half-word accesses not supported (2.4.2) Added DAC errata: • DMA request not automatically cleared by clearing DMAEN (2.5.1) • DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge (2.5.2) Added TIM errata: • PWM re-enabled in automatic output enable mode despite of system break (2.6.1) • TRGO and TRGO2 trigger output failure (2.6.2) • Consecutive compare event missed in specific conditions (2.6.3) • Output compare clear not working with external counter reset (2.6.4) Added IWDG errata: • RVU flag not cleared at low APB clock frequency (2.7.3) • PVU flag not cleared at low APB clock frequency (2.7.4) Added RTC errata: • RTC interrupt can be masked by another RTC interrupt (2.8.3) • Calendar initialization may fail in case of consecutive INIT mode entry (2.8.4) • Alarm flag may be repeatedly set when the core is stopped in debug (2.8.5) USART errata: • Added Communication parameters reprogramming after ATR in Smartcard mode when SCLK is used to clock the card (2.10.1) and Non-compliant sampling for NACK signal from smartcard (2.10.2) • Updated Start bit detected too soon when sampling for NACK signal from the smartcard. SPI/I2S errata: • Added BSY bit may stay high when SPI is disabled (2.11.1), Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters (2.11.6), CRC error in SPI slave mode if internal NSS changes before CRC transfer (2.11.7), Anticipated communication upon SPI transit from slave receiver to master (2.11.8). • Updated BSY bit may stay high at the end of data transfer in slave mode (2.11.2) and SPI CRC corruption upon DMA transaction completion by another peripheral (2.11.4) Added USB errata: • Possible packet memory overrun/underrun at low APB frequency (2.12.1) • False wakeup detection for last K-state not terminated by EOP or reset before suspend (2.12.2)
12-Aug-2024	10	Added errata: • LSE crystal oscillator may be disturbed by transitions on PC13 • A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF • Buffer description table update completes after CTR interrupt triggers

Contents