STM32F446xC/xE Errata sheet

STM32F446xC/xE device limitations

Silicon identification

This errata sheet applies to the revision A of STM32F446xx microcontroller families.

The STM32F446xx devices feature an ARM® 32-bit Cortex®-M4 core with FPU, for which an errata notice is also available (see *Section 1* for details).

The full list of part numbers is shown in *Table 2*. The products are identifiable as shown in *Table 1*:

- by the revision code marked below the order code on the device package
- by the last three digits of the Internal order code printed on the box label

Table 1. Device identification(1)

Order code	Revision code marked on device(2)
STM32F446xx	“A”

1. The REV_ID bits in the DBGMCU_IDCODE register show the revision code of the device (see the RM0390 STM32F446xx reference manual for details on how to find the revision code).
2. Refer to the device datasheet for details on how to identify the revision code and the date code on the different packages.

Table 2. Device summary

Reference	Part number
STM32F446xx	STM32F446MC, STM32F446ME, STM32F446RC, STM32F446RE, STM32F446VC, STM32F446VE, STM32F446ZC, STM32F446ZE.

Contents

List of tables

1 ARM® 32-bit Cortex®-M4 with FPU limitations

An errata notice of the STM32F446xx core is available from http://infocenter.arm.com.

All the described limitations are minor and related to the revision r0p1-v1 of the Cortex®-M4 core. *Table 3* summarizes these limitations and their implications on the behavior of STM32F446xx devices.

Table 3. Cortex®-M4 core limitations and impact on microcontroller behavior

ARM ID	ARM category	ARM summary of errata	Impact on STM32F446xx
752770	Cat B	Interrupted loads to SP can cause erroneous behavior	Minor
776924	Cat B	VDIV or VSQRT instructions might not complete correctly when very short ISRs are used	Minor

1.1 Cortex®-M4 interrupted loads to stack pointer can cause erroneous behavior

Description

An interrupt occurring during the data-phase of a single word load to the stack pointer (SP/R13) can cause an erroneous behavior of the device. In addition, returning from the interrupt results in the load instruction being executed an additional time.

For all the instructions performing an update of 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 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 limitations 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

1.2 VDIV or VSQRT instructions might not complete correctly when very short ISRs are used

Description

On Cortex®-M4 with FPU core, 14 cycles are required to execute a VDIV or VSQRT instruction.

This limitation is present when the following conditions are met:

- A VDIV or VSQRT is executed
- The destination register for VDIV or VSQRT is one of s0 - s15
- An interrupt occurs and is taken
- The ISR being executed does not contain a floating point instruction
- 14 cycles after the VDIV or VSQRT is executed, an interrupt return is executed

In this case, if there are only one or two instructions inside the interrupt service routine, then the VDIV or VQSRT instruction does not complete correctly and the register bank and FPSCR are not updated, meaning that these registers hold incorrect out-of-date data.

Workaround

Two workarounds are applicable:

- Disable lazy context save of floating point state by clearing LSPEN to 0 (bit 30 of the FPCCR at address 0xE000EF34).
- Ensure that every ISR contains more than 2 instructions in addition to the exception return instruction.

2 STM32F446xx silicon limitations

Table 4 gives quick references to all documented limitations.

Legend for *Table 4*: A = workaround available; N = no workaround available; P = partial workaround available, '-' and grayed = fixed.

Table 4. Summary of silicon limitations

Links to silicon limitations		Revision A
Section 2.1: System limitations	*Section 2.1.1: Debugging Sleep/Stop mode with WFE/WFI entry*	A
	Section 2.1.2: Wakeup sequence from Standby mode when using more than one wakeup source	A
	Section 2.1.3: Full JTAG configuration without NJTRST pin cannot be used	A
	Section 2.1.4: MPU attribute to RTC and IWDG registers could be managed incorrectly	A
	Section 2.1.5: Delay after an RCC peripheral clock enabling	A
	Section 2.1.6: Internal noise impacting the ADC accuracy	A
	Section 2.1.7: Data cache might be corrupted during Flash read-while-write operation	A
Section 2.2: IWDG peripheral limitations	*Section 2.2.1: RVU and PVU flags are not reset in Stop mode*	A
Section 2.3: RTC limitations	*Section 2.3.1: Spurious tamper detection when disabling the tamper channel*	N
	Section 2.3.2: Detection of a tamper event occurring before enabling the tamper detection is not supported in edge detection mode	A
	Section 2.3.3: RTC calendar registers are not locked properly	A
Section 2.4: I2C peripheral limitations	*Section 2.4.1: SMBus standard not fully supported*	A
	Section 2.4.2: Start cannot be generated after a misplaced Stop	A
	Section 2.4.3: Mismatch on the "Setup time for a repeated Start condition" timing parameter	A
	Section 2.4.4: Data valid time ($t_{VD;DAT}$) violated without the OVR flag being set	A
	Section 2.4.5: Both SDA and SCL maximum rise time (t_r) violated when VDD_I2C bus higher than ((VDD+0.3) / 0.7) V	A
	Section 2.4.6: Spurious Bus Error detection in Master mode	A
	Section 2.4.7: Wrong data sampling when data set-up time (tSU;DAT) is smaller than one FMPI2CCLK period	A

Table 4. Summary of silicon limitations (continued)

Links to silicon limitations		Revision A
Section 2.5: SPI peripheral limitations	*Section 2.5.1: Wrong CRC calculation when the polynomial is even*	A
	Section 2.5.2: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback	A
	Section 2.5.3: Wrong CRC transmitted in Master mode with delayed SCK feedback	A
	Section 2.5.4: BSY bit may stay high at the end of a data transfer in Slave mode	A
Section 2.6: I2S peripheral limitations	*Section 2.6.1: In I2S Slave mode, with Bit ASTRTEN=1, WS level must be set by the external masterwhen enabling the I2S*	A
	Section 2.6.2: In I2S slave PCM short pulse mode when WS hold time is not respected, it results in data corruption	A
	Section 2.6.3: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback	A
Section 2.7: USART peripheral limitations	*Section 2.7.1: Idle frame is not detected if receiver clock speed is deviated*	N
	Section 2.7.2: In full-duplex mode, the Parity Error (PE) flag can be cleared by writing to the data register	A
	Section 2.7.3: Parity Error (PE) flag is not set when receiving in Mute mode using address mark detection	N
	Section 2.7.4: Break frame is transmitted regardless of nCTS input line status	N
	Section 2.7.5: nRTS signal abnormally driven low after a protocol violation	A
	Section 2.7.6: Start bit detected too soon when sampling for NACK signal from the smartcard	A
	Section 2.7.7: Break request can prevent the Transmission Complete flag (TC) from being set	A
	Section 2.7.8: Guard time is not respected when data are sent on TXE events	A
	Section 2.7.9: nRTS is active while RE or UE = 0	A
Section 2.8: bxCAN limitations	*Section 2.8.1: bxCAN time triggered communication mode not supported*	A
Section 2.9: FSMC peripheral limitations	*Section 2.9.1: Dummy read cycles inserted when reading synchronous memories*	N
Section 2.10: SDIO peripheral limitations	*Section 2.10.1: Wrong CCRCFAIL status after a response without CRC is received*	A
	Section 2.10.2: No underrun detection with wrong data transmission	A
Section 2.11: ADC peripheral limitations	*Section 2.11.1: ADC sequencer modification during conversion*	A

Table 4. Summary of silicon limitations (continued)

Links to silicon limitations		Revision A
Section 2.12: DAC peripheral limitations	*Section 2.12.1: DMA underrun flag management*	A
	Section 2.12.2: DMA request not automatically cleared by DMAEN=0	A
Section 2.13: QuadSPI limitations	*Section 2.13.1: Extra data written in the FIFO at the end of a read transfer*	A
	Section 2.13.2: First nibble of data is not written after dummy phase	A
	Section 2.13.3: Wrong data can be read in memory-mapped after an indirect mode operation	A

2.1 System limitations

2.1.1 Debugging Sleep/Stop mode with WFE/WFI entry

Description

When the Sleep debug or Stop debug mode is enabled (DBG_SLEEP bit or DBG_STOP bit are set in the DBGMCU_CR register), this allows software debugging during Sleep or Stop mode. After wakeup some unreachable instructions could be executed if the following condition are met:

- If the application software disables the Prefetch queue
- The number of wait state configured on Flash interface is higher than 0
- And Linker place WFE or WFI instructions on 4-bytes aligned addresses (0x080xx_xxx4)

Workaround

- Add three NOPs after WFI/WFE instruction
- Keep one AHB master active during sleep (example keep DMA1 or DMA2 RCC clock enable bit set)
- Execute WFI/WFE instruction from routines inside the SRAM

2.1.2 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 needs to be cleared prior to Standby mode entry, otherwise the MCU wakes up immediately.

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

Workaround

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

- Disable all used wakeup sources,
- Clear all related wakeup flags,
- Re-enable all used wakeup sources,
- Enter Standby mode

Note: Be aware that, when applying this workaround, if one of the wakeup sources is still kept high, the MCU enters Standby mode but then it wakes up immediately generating a power reset.

2.1.3 Full JTAG configuration without NJTRST pin cannot be used

Description

When using the JTAG debug port in debug mode, the connection with the debugger is lost if the NJTRST pin (PB4) is used as a GPIO. Only the 4-wire JTAG port configuration is impacted.

Workaround

Use the SWD debug port instead of the full 4-wire JTAG port.

2.1.4 MPU attribute to RTC and IWDG registers could be managed incorrectly

Description

If the MPU is used and the non bufferable attribute is set to the RTC or IWDG memory map region, the CPU access to the RTC or IWDG registers could be treated as bufferable, provided that there is no APB prescaler configured (AHB/APB prescaler is equal to 1).

Workaround

If the non bufferable attribute is required for these registers, the software could perform a read after the write to guaranty the completion of the write access.

2.1.5 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 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®-M4 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).
3. Or simply insert a dummy read operation from the corresponding register just after enabling the peripheral clock.

2.1.6 Internal noise impacting the ADC accuracy

Description

An internal noise generated on V_{DD} supplies and propagated internally may impact the ADC accuracy.

This noise is always active whatever the power mode of the MCU (RUN or Sleep).

Workarounds

To adapt the accuracy level to the application requirements, set one of the following options:

- Option1
  Set the ADCDC1 bit in the PWR_CR register.
- Option2
  Set the corresponding ADCxDC2 bit in the SYSCFG_PMC register.

Only one option can be set at a time.

For more details on option 1 and option2 mechanisms, refer to AN4073.

2.1.7 Data cache might be corrupted during Flash read-while-write operation

Description

When a write operation to the internal Flash memory is done, the data cache is updated to reflect the data update. If a read operation to the other memory bank occurs during the data cache update, the data cache content may be corrupted. In this case, subsequent read operations from the same address (Cache hits) will be corrupted.

This issue only occurs in dual bank mode when reading (data access or code execution) from one Flash bank while writing to the other Flash bank with data cache enabled.

Workaround

When the application is performing data accesses in both Flash memory banks, the data cache must be disabled by resetting the DCEN bit in FLASH_ACR register before performing any write operation to Flash memory. Before enabling the data cache again, the cache must be reset by setting and then resetting the DCRST bit in FLASH_ACR register.

Example of code

```
  /* Disable data cache  */
  __HAL_FLASH_DATA_CACHE_DISABLE();

  /* Set PG bit */
  SET_BIT(FLASH->CR, FLASH_CR_PG);

  /* Program the Flash word */
WriteFlash(Address, Data);

  /* Reset data cache */
  __HAL_FLASH_DATA_CACHE_RESET();
  /* Enable data cache */
  __HAL_FLASH_DATA_CACHE_ENABLE();
```

2.2 IWDG peripheral limitations

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 the 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 is reset before entering the Stop mode.

2.3 RTC limitations

2.3.1 Spurious tamper detection when disabling the tamper channel

Description

If the tamper detection is configured for detection on 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

2.3.2 Detection of a tamper event occurring before enabling the tamper detection is not supported in edge detection mode

Description

When the tamper detection is enabled in edge detection mode (TAMPFLT=00):

- When TAMPxTRG=0 (rising edge detection): if the tamper input is already high before enabling the tamper detection, the tamper event may or may not be detected when enabling the tamper detection. The probability to detect it increases with the APB frequency.
- When TAMPxTRG=1 (falling edge detection): if the tamper input is already low before enabling the tamper detection, the tamper event is not detected when enabling the tamper detection.

Workaround

The I/O state should be checked by software in the GPIO registers, just after enabling the tamper detection and before writing sensitive values in the backup registers, in order to ensure that no active edge occurred before enabling the tamper event detection.

2.3.3 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, 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.

2.4 I2C peripheral limitations

2.4.1 SMBus standard not fully supported

Description

The I^{2}C peripheral is not fully compliant with the SMBus v2.0 standard since It does not support the capability to NACK an invalid byte/command.

Workarounds

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

1. Using the SMBAL pin if supported by the host
2. the alert response address (ARA) protocol
3. the Host notify protocol

2.4.2 Start cannot be generated after a misplaced Stop

Description

If a master generates a misplaced Stop on the bus (bus error) while the microcontroller I2C peripheral attempts to switch to Master mode by setting the START bit, the Start condition is not properly generated.

Workaround

In the I^{2}C standard, it is allowed to send a Stop only at the end of the full byte (8 bits + acknowledge), so this scenario is not allowed. Other derived protocols like CBUS allow it, but they are not supported by the I^{2}C peripheral.

A software workaround consists in asserting the software reset using the SWRST bit in the I2C_CR1 control register.

2.4.3 Mismatch on the “Setup time for a repeated Start condition” timing parameter

Description

In case of a repeated Start, the “Setup time for a repeated Start condition” (named Tsu;sta in the I²C specification) can be slightly violated when the I²C operates in Master Standard mode at a frequency between 88 kHz and 100 kHz.

The limitation can occur only in the following configuration:

- in Master mode
- in Standard mode at a frequency between 88 kHz and 100 kHz (no limitation in Fast-mode)
- SCL rise time:
  - If the slave does not stretch the clock and the SCL rise time is more than 300 ns (if the SCL rise time is less than 300 ns, the limitation cannot occur)
  - If the slave stretches the clock

The setup time can be violated independently of the APB peripheral frequency.

Workaround

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

2.4.4 Data valid time ($t_{VD;DAT}$) violated without the OVR flag being set

Description

The data valid time ($t_{VD;DAT}$, $t_{VD;ACK}$) described by the I²C standard can be violated (as well as the maximum data hold time of the current data ($t_{HD;DAT}$)) under the conditions described below. This violation cannot be detected because the OVR flag is not set (no transmit buffer underrun is detected).

This limitation can occur only under the following conditions:

- in Slave transmit mode
- with clock stretching disabled (NOSTRETCH=1)
- if the software is late to write the DR data register, but not late enough to set the OVR flag (the data register is written before)

Workaround

If the master device allows it, use the clock stretching mechanism by programming the bit NOSTRETCH=0 in the I2C_CR1 register.

If the master device does not allow it, ensure that the software is fast enough when polling the TXE or ADDR flag to immediately write to the DR data register. For instance, use an interrupt on the TXE or ADDR flag and boost its priority to the higher level.

2.4.5 Both SDA and SCL maximum rise time (t_r) violated when VDD_I2C bus higher than ((VDD+0.3) / 0.7) V

Description

When an external legacy I^2C bus voltage (VDD_I2C) is set to 5 V while the MCU is powered from V_{DD}, the internal 5-Volt tolerant circuitry is activated as soon the input voltage (V_{IN}) reaches the V_{DD} + diode threshold level. An additional internal large capacitance then prevents the external pull-up resistor (R_P) from rising the SDA and SCL signals within the maximum timing (t_r) which is 300 ns in fast mode and 1000 ns in Standard mode.

The rise time (t_r) is measured from V_{IL} and V_{IH} with levels set at 0.3VDD_I2C and 0.7VDD_I2C.

Workaround

The external VDD_I2C bus voltage should be limited to a maximum value of ((VDD+0.3) / 0.7) V. As a result, when the MCU is powered from V_{DD}=3.3 V, VDD_I2C should not exceed 5.14 V to be compliant with I^2C specifications.

2.4.6 Spurious Bus Error detection in Master mode

Description

In Master mode, a bus error can be detected by mistake, so the BERR flag can be wrongly raised in the status register. This will generate a spurious Bus Error interrupt if the interrupt is enabled. A bus error detection has no effect on the transfer in Master mode, therefore the I2C transfer can continue normally.

Workaround

If a bus error interrupt is generated in Master mode, the BERR flag must be cleared by software. No other action is required and the on-going transfer can be handled normally.

2.4.7 Wrong data sampling when data set-up time (tSU;DAT) is smaller than one FMPI2CCLK period

Description

The I2C bus specification and user manual specifies a minimum data set-up time (tSU;DAT) at:

- 250ns in Standard-mode,
- 100 ns in Fast-mode,
- 50 ns in Fast-mode Plus.

The I2C SDA line is not correctly sampled when tSU;DAT is smaller than one FMPI2CCLK (FMPI2C clock) period: the previous SDA value is sampled instead of the current one. This can result in a wrong Slave address reception, a wrong received data byte, or a wrong received acknowledge bit.

Workaround

Increase the I2CCLK frequency to get I2CCLK period smaller than the transmitter minimum data set-up time. Or, if it is possible, increase the transmitter minimum data set-up time.

2.5 SPI peripheral limitations

2.5.1 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.

Work-around

Use odd polynomial.

2.5.2 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 I^2S mode is more sensitive than the SPI mode since the SCK clock is not synchronized with the APB. 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.

Workarounds

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.

The following table gives the maximum allowable APB frequency versus 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

OSPEEDR [1:0] for SCK pin	Max. APB frequency for SPI mode [MHz]	Max. APB frequency for I^2S mode [MHz]
11 (very high)	90	84 (if V_{DD} > 2.7 V)
10 (high)	90	60
01 (medium)	80	35
00 (low)	26	14

2.5.3 Wrong CRC transmitted in Master mode with delayed SCK feedback

Description

In transmit transaction of the SPI/I^{2}S interface in SPI Master mode with 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 data and CRC bit shifting and 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.

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

Table 6. Maximum allowable APB frequency at 30 pF load

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)	90	84 (if V_{DD} > 2.7 V)
10 (high)	90	60
01 (medium)	80	35
00 (low)	26	14

2.5.4 BSY bit may stay high at the end of a data transfer in Slave mode

Description

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 internal CPU clock and external SCK clock provided by 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 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 NSS hardware management is applied and NSS signal is provided by master, the end of a transaction can be detected by the NSS polling by slave.

- If 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, 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 TXE event is detected while the data frame transfer is still ongoing. It is impossible to achieve it when ratio between CPU and SPI clock is low and data frame is short especially. In this specific case timeout can be measured from TXE, while calculating fixed number of CPU clock periods corresponding to the time necessary to complete the data frame transaction.

2.6 I2S peripheral limitations

2.6.1 In I2S Slave mode, with Bit ASTRTEN=1, WS level must be set by the external masterwhen enabling the I2S

Description

In Slave mode, and when Bit ASTRTEN=1: I2S Asynchronous Start is enabled, 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 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.

2.6.2 In I2S slave PCM short pulse mode when WS hold time is not respected, it results in data corruption

Description

If I2S peripheral is configured in

- Slave mode (I2SCFG = '00' or '01')
- I2S PCM standard selected (I2SSTD = '11')
- I2S Asynchronous Start is disabled (ASTRTEN = '0')

and the WS signal does not respect hold time versus SCK. In these conditions the data transmitted and received by the slave are corrupted, because the master and slave will be de-synchronized throughout the whole communication.

Workaround

Two work around are possible:

- The THOLD (NSS_WS vs SCK rising edge) time must be respected by the I2S Master.
- Enable I2S Asynchronous Start (ASTRTEN bit is equal to '1') when using I2S PCM standard.

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

The limitation described in *Section 2.5.2: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback* also applies to I^{2}S interface.

2.7 USART peripheral limitations

2.7.1 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.

2.7.2 In full-duplex mode, the Parity Error (PE) flag can be cleared by writing to 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 to 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.

2.7.3 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.

2.7.4 Break frame is transmitted regardless of nCTS input line status

Description

When 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 the 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.

2.7.5 nRTS signal abnormally driven low after a protocol violation

Description

When RTS hardware flow control is enabled, the nRTS signal goes high when data is received. If this data was not read and 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 USART side, an overrun is detected, which indicates that data has been lost.

Workaround

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 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.

2.7.6 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

2.7.7 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 if the following conditions are met:

- 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 D1 data transfer.

Workaround

If the application needs to detect the end of a data transfer, the break request should be issued after checking that the TC flag is set.

2.7.8 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 i.e. the data written in the data register is transferred on the bus without waiting 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.

2.7.9 nRTS is active while RE or UE = 0

Description

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

Workaround

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

2.8 bxCAN limitations

2.8.1 bxCAN time triggered communication mode not supported

Description

The time triggered communication mode described in the reference manual is not supported. As a result timestamp values are not available. TTCM bit must be kept cleared in the CAN_MCR register (time triggered communication mode disabled).

Workaround

None

2.9 FSMC peripheral limitations

2.9.1 Dummy read cycles inserted when reading synchronous memories

Description

When performing a burst read access to a synchronous memory, two dummy read accesses are performed at the end of the burst cycle whatever the type of AHB burst access. However, the extra data values which are read are not used by the FSMC and there is no functional failure.

Workaround

None.

2.10 SDIO peripheral limitations

2.10.1 Wrong CCRCFAIL status after a response without CRC is received

Description

The CRC is calculated even if the response to a command does not contain any CRC field. As a consequence, after the SDIO command IO_SEND_OP_COND (CMD5) is sent, the CCRCFAIL bit of the SDIO_STA register is set.

Workaround

The CCRCFAIL bit in the SDIO_STA register shall be ignored by the software. CCRCFAIL must be cleared by setting CCRCFAILC bit of the SDIO_ICR register after reception of the response to the CMD5 command.

2.10.2 No underrun detection with wrong data transmission

Description

In case there is an ongoing data transfer from the SDIO host to the SD card and the hardware flow control is disabled (bit 14 of the SDIO_CLKCR is not set), if an underrun condition occurs, the controller may transmit a corrupted data block (with wrong data word) without detecting the underrun condition when the clock frequencies have the following relationship:

[3 x period(PCLK2) + 3 x period(SDIOCLK)] >= (32 / (BusWidth)) x period(SDIO_CK)

Workaround

Avoid the above-mentioned clock frequency relationship, by:

- Incrementing the APB frequency
- or decreasing the transfer bandwidth
- or reducing SDIO_CK frequency

2.11 ADC peripheral limitations

2.11.1 ADC sequencer modification during conversion

Description

If an ADC conversion is started by software (writing the SWSTART bit), and if the ADC_SQRx or ADC_JSQRx registers are modified during the conversion, the current conversion is reset and the ADC does not restart a new conversion sequence automatically.

If an ADC conversion is started by hardware trigger, this limitation does not apply. The ADC restarts a new conversion sequence automatically.

Workaround

When an ADC conversion sequence is started by software, a new conversion sequence can be restarted only by setting the SWSTART bit in the ADC_CR2 register.

2.12 DAC peripheral limitations

2.12.1 DMA underrun flag management

Description

If the DMA is not fast enough to input the next digital data to the DAC, as a consequence, the same digital data is converted twice. In these conditions, the DMAUDR flag is set, which usually leads to disable the DMA data transfers. This is not the case: the DMA is not disabled by DMAUDR=1, and it keeps servicing the DAC.

Workaround

To disable the DAC DMA stream, reset the EN bit (corresponding to the DAC DMA stream) in the DMA_SxCR register.

2.12.2 DMA request not automatically cleared by DMAEN=0

Description

if the application wants to stop the current DMA-to-DAC transfer, the DMA request is not automatically cleared by DMAEN=0, or by DACEN=0.

If the application stops the DAC operation while the DMA request is high, the DMA request will be pending while the DAC is reinitialized and restarted; with the risk that a spurious unwanted DMA request is serviced as soon as the DAC is re-enabled.

Workaround

To stop the current DMA-to-DAC transfer and restart, the following sequence should be applied:

1. Check if DMAUDR is set.
2. Clear the DAC/DMAEN bit.
3. Clear the EN bit of the DAC DMA/Stream
4. Reconfigure by software the DAC, DMA, triggers etc.
5. Restart the application.

2.13 QuadSPI limitations

2.13.1 Extra data written in the FIFO at the end of a read transfer

Description

When all the conditions listed below are gathered:

- QUADSPI is used in indirect mode
- QUADSPI clock is AHB/2 (PRESCALER = 0x01 in the QUADSPI_CR)
- QUADSPI is in quad mode (DMO
  - DE = 0b11 in the QUADSPI_CCR)
- QUADSPI is in DDR mode (DDR
  - M = 0b1 in the QUADSPI_CCR)

An extra data is incorrectly written in the FIFO when a data is read at the same time that the FIFO gets full at the end of a read transfer.

Workaround

One of the two workarounds listed below can be done:

- Read out the extra data until the BUSY flag goes low and discard it.
- Request an abort after reading out all the correct received data from FIFO in order to flush FIFO and have the busy low. Abort will keep the last register configuration (set the ABORT bit in the QUADSPI_CR).

2.13.2 First nibble of data is not written after dummy phase

Description:

The first nibble of data to be written to the external flash is lost in the following conditions:

- QUADSPI is used in indirect write mode and
- at least one dummy cycle is used

Workaround

Do not use dummy cycles for creating latency between address phase and data phase, in indirect write mode. Instead, use alternate bytes to substitute the dummy cycles. The same latency can be achieved if the number of dummy cycles to substitute with alternate-byte cycles is an integer multiple of the number of cycles required for transferring one alternate byte, as shown in the table:

Table 7. QUADSPI mode

QUADSPI mode	Number of cycles per alternate byte
4-data-line DDR	1
4-data-line SDR	2
2-data-line SDR	4
1-data-line SDR	8

For example, the latency corresponding to eight dummy cycles can be exactly substituted with one single alternate byte in 1-data-line SDR mode, but two alternate bytes are required in 2-data-line SDR mode. One single dummy cycle can only exactly be substituted in 4-data-line DDR mode, using one alternate byte.

Note: This is also applicable to dual-flash memory mode.

2.13.3 Wrong data can be read in memory-mapped after an indirect mode operation

Description

Wrong data can be read with the first memory-mapped read request in the following condition:

Quad-SPI peripheral entered memory-mapped mode with both LSB bits in the address register QUADSPI_AR[1:0] not reset.

Workaround

QUADSPI_AR register must be reset just before entering memory-mapped mode.

Depending on the current Quad-SPI operating mode, one of the two workarounds listed below can be used:

1. Indirect read mode: reset address register then do an abort request to stop reading and clear busy bit. Then enter to memory-mapped mode.
2. Indirect write mode: reset the address register then enter to memory-mapped mode.

Note: User should take care to not write to QUADSPI_DR register after resetting address register.

3 Revision history

Table 8. Document revision history

Date	Revision	Changes
16-Mar-2015	1	Initial release.
03-Jan-2017	2	Updated: – *Section 2.1.5: Delay after an RCC peripheral clock enabling*. – *Section 2.4.2: Start cannot be generated after a misplaced Stop* Added – *Section 2.1.7: Data cache might be corrupted during Flash read-while-write operation*. – *Section 2.3.1: Spurious tamper detection when disabling the tamper channel* – *Section 2.3.2: Detection of a tamper event occurring before enabling the tamper detection is not supported in edge detection mode* – *Section 2.3.3: RTC calendar registers are not locked properly*. – *Section 2.4.6: Spurious Bus Error detection in Master mode*. – *Section 2.5.1: Wrong CRC calculation when the polynomial is even* – *Section 2.5.2: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback* – *Section 2.5.3: Wrong CRC transmitted in Master mode with delayed SCK feedback* – *Section 2.5.4: BSY bit may stay high at the end of a data transfer in Slave mode* – *Section 2.6.3: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback*