STM32F76xxx/77xxx device errata

Applicability

This document applies to the part numbers of STM32F76xxx/77xxx 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 RM0341.

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
STM32F76xxx	STM32F765BG, STM32F765BI, STM32F765IG, STM32F765II, STM32F765NG, STM32F765NI, STM32F765VG, STM32F765VI, STM32F765ZG, STM32F765ZI, STM32F767BG, STM32F767BI, STM32F767IG, STM32F767II, STM32F767NG, STM32F767NI, STM32F767VG, STM32F767VI, STM32F767ZG, STM32F767ZI, STM32F768AI, STM32F769AG, STM32F769AI, STM32F769BG, STM32F769BI, STM32F769IG, STM32F769II, STM32F769NG, STM32F769NI
STM32F77xxx	STM32F777BI, STM32F777II, STM32F777NI, STM32F777VI, STM32F777ZI, STM32F778AI, STM32F779AI, STM32F779BI, STM32F779II, STM32F779NI

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32F76xxx, STM32F77xxx	A	0x1000
	Z, 1	0x1001

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

1 Summary of device errata

The following table gives a quick reference to the STM32F76xxx/77xxx 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	Rev. Z	Rev. 1
Core	2.1.1	Cortex-M7 data corruption when using Data cache configured in write-through	N	N	N
System	2.2.1	Internal noise impacting the ADC accuracy	A	A	A
	2.2.2	Wakeup from Standby mode when the back-up SRAM regulator is enabled	A	A	A
	2.2.3	LSE high driving and low driving capability is not usable for TFBGA216 package under certain conditions	A	A	A
	2.2.4	DTCM-RAM not accessible in read when the MCU is in Sleep mode (WFI/WFE)	A	A	A
	2.2.5	Full JTAG configuration without NJTRST pin cannot be used	A	A	A
	2.2.6	PC13 signal transitions disturb LSE	N	N	N
FMC	2.3.1	Dummy read cycles inserted when reading synchronous memories	N	N	N
	2.3.3	Wrong data read from a busy NAND memory	A	A	A
	2.3.4	Spurious clock stoppage with continuous clock feature enabled	A	A	A
	2.3.5	Data read might be corrupted when the write FIFO is disabled	A	A	A
QUADSPI	2.4.1	First nibble of data not written after dummy phase	A	A	A
	2.4.2	Wrong data from memory-mapped read after an indirect mode operation	A	A	A
	2.4.3	Memory-mapped read operations may fail when timeout counter is enabled	P	P	P
	2.4.4	Memory-mapped access in indirect mode clearing QUADSPI_AR register	P	P	P
ADC	2.5.1	ADC sequencer modification during conversion	A	A	A
DAC	2.6.1	DMA request not automatically cleared by clearing DMAEN	A	A	A
	2.6.2	DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge	N	N	N
	2.6.3	DMA underrun flag management	A	A	A
DSI	2.7.1	Tearing effect parasitic detection	P	P	P
	2.7.2	Incorrect calculation of the time to activate the clock between HS transmissions	P	P	P
	2.7.3	The immediate update procedure may fail	A	A	A

Function	Section	Limitation	Status		
			Rev. A	Rev. Z	Rev. 1
DSI	2.7.4	When used over the DSI link, the tearing effect interrupt flag is set when an acknowledge trigger is received from the display	A	A	A
JPEG	2.8.1	False EOI marker is inserted after clearing the HDR bit	A	A	A
	2.8.2	No DMA transfer complete generated at the end of the encoding process after clearing the HDR bit	A	A	A
	2.8.3	JPEG FIFO might be corrupted	A	A	A
TIM	2.10.1	One-pulse mode trigger not detected in master-slave reset + trigger configuration	P	P	P
	2.10.2	Consecutive compare event missed in specific conditions	N	N	N
	2.10.3	Output compare clear not working with external counter reset	P	P	P
LPTIM	2.11.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	A	A	A
	2.11.2	Device may remain stuck in LPTIM interrupt when clearing event flag	P	P	P
	2.11.3	LPTIM events and PWM output are delayed by 1 kernel clock cycle	P	P	P
IWDG	2.12.1	RVU flag not reset in Stop	A	A	A
	2.12.2	PVU flag not reset in Stop	A	A	A
	2.12.3	WVU flag not reset in Stop	A	A	A
	2.12.4	RVU flag not cleared at low APB clock frequency	A	A	A
	2.12.5	PVU flag not cleared at low APB clock frequency	A	A	A
	2.12.6	WVU flag not cleared at low APB clock frequency	A	A	A
RTC and TAMP	2.13.1	RTC calendar registers are not locked properly	A	A	A
	2.13.2	RTC interrupt can be masked by another RTC interrupt	A	A	A
	2.13.3	Calendar initialization may fail in case of consecutive INIT mode entry	A	A	A
	2.13.4	Alarm flag may be repeatedly set when the core is stopped in debug	N	N	N
I2C	2.14.1	10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave	A	A	A
	2.14.3	Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period	P	P	P
	2.14.4	Spurious bus error detection in master mode	A	A	A
	2.14.5	Last-received byte loss in reload mode	P	P	P
	2.14.6	Spurious master transfer upon own slave address match	P	P	P
	2.14.7	OVR flag not set in underrun condition	N	N	N
	2.14.8	Transmission stalled after first byte transfer	A	A	A
USART	2.15.1	RTS is active while RE = 0 or UE = 0	A	A	A
	2.15.2	Receiver timeout counter wrong start in two-stop-bit configuration	A	A	A
	2.15.3	Data corruption due to noisy receive line	N	N	N
SPI	2.16.2	BSY bit may stay high at the end of data transfer in slave mode	A	A	A
	2.16.3	Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters	A	A	A
	2.16.4	CRC error in SPI slave mode if internal NSS changes before CRC transfer	D	D	D
	2.16.1	BSY bit may stay high when SPI is disabled	A	A	A
SAI	2.17.2	Last SAI_SCK clock pulse truncated upon disabling SAI master	N	N	N

Function	Section	Limitation	Status		
			Rev. A	Rev. Z	Rev. 1
SAI	2.17.3	Last SAI_MCLK clock pulse truncated upon disabling SAI master	A	A	A
	2.17.4	SAI_MCLK clock absent in a specific configuration	A	A	A
SDMMC	2.18.1	Wrong CCRCFAIL status after a response without CRC is received	A	A	A
	2.18.2	MMC stream write of less than seven bytes does not work correctly	A	A	A
bxCAN	2.19.1	bxCAN time-triggered communication mode not supported	N	N	N
OTG_FS	2.20.1	Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_FS registers	A	A	A
	2.20.2	Host packet transmission may hang when connecting through a hub to a low-speed device	N	N	N
OTG_HS	2.21.1	Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_HS registers	A	A	A
	2.21.2	Host packet transmission may hang when connecting through a hub to a low-speed device	N	N	N
ETH	2.22.1	Incorrect L3 checksum is inserted in transmitted IPv6 packets without TCP, UDP or ICMP payloads	A	A	A
	2.22.2	The ethernet MAC processes invalid extension headers in the received IPv6 frames	N	N	N
	2.22.3	MAC stuck in the idle state on receiving the TxFIFO flush command exactly one clock cycle after a transmission completes	P	P	P
	2.22.4	Transmit frame data corruption	A	A	A
	2.22.5	Incorrect status and corrupted frames when RxFIFO overflow occurs on the penultimate word of Rx frames	A	A	A
	2.22.6	Ethernet erroneous data received in RMII configuration	A	A	A

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

Table 4. Summary of device documentation errata

Function	Section	Documentation erratum
FMC	2.3.2	Missing information on prohibited 0xFF value of NAND transaction wait timing
HASH	2.9.1	Superseded suspend sequence for data loaded by DMA
	2.9.2	Superseded suspend sequence for data loaded by the CPU
TIM	2.10.4	TIM12 input XOR function not available
RTC and TAMP	2.13.5	Setting GPIO properties of PC13 used as RTC_ALARM open-drain output
I2C	2.14.2	Wrong behavior in Stop mode
SPI	2.16.4	CRC error in SPI slave mode if internal NSS changes before CRC transfer
SAI	2.17.1	Automatic restart upon late or anticipated frame error in I2S slave mode not supported

2 Description of device errata

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

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

2.1 Core

Reference manual and errata notice for the Arm® Cortex®-M7 core is available from http://infocenter.arm.com.

2.1.1 Cortex-M7 data corruption when using Data cache configured in write-through

Description

This limitation is registered under Arm® ID number 1259864 and classified into “Category A”.

If a particular sequence of stores and loads is performed to write-through memory, and some timing-based internal conditions are met, then a load might not get the last data stored to that address.

This erratum can only occur if the loads and stores are to write-through memory. This could be due to any of the following:

- The MPU has been programmed to set this address as write-through .
- The default memory map is being used and this address is write-through in that map.
- The memory is cacheable, and the CM7_CACR.FORCEWT bit is set.
- The memory is cacheable, shared, and the CM7_CACR.SIWT bit is set.

The following sequence is required for this erratum to occur:

1. The address of interest must be in the cache.
2. A write-through store to the same doubleword as the address of interest.
3. One of the following:
   – A linefill is started (to a different cacheline to the address of interest) that allocates to the same set as the address of interest.
   – An ECC error.
   – A cache maintenance operation without a following DSB.
4. A store to the address of interest.
5. A load from the address of interest.

If certain specific timing conditions are met, the load will get the data from the first store, or from what was in the cache at the start of the sequence instead of the data from the second store.

The effect of this erratum is that load operations can return incorrect data.

Workaround

There is no direct workaround for this erratum.

Where possible, Arm® recommends that you use the MPU to change the attributes on any write-through memory to write-back memory. If this is not possible, it might be necessary to disable the cache for sections of code that access write-through memory.

2.2 System

2.2.1 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).

Workaround

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

- Option 1: Set the ADCDC1 bit in the PWR_CR register.
- Option 2: Set the corresponding ADCxDC2 bit in the SYSCFG_PMC register.

Only one option can be set at a time. For more details on option1 and option 2 mechanisms, refer to AN4073.

2.2.2 Wakeup from Standby mode when the back-up SRAM regulator is enabled

Description

When writing to the PWR_CSR1 register to enable or disable the back-up SRAM regulator, if the EIWUP bit is overwritten 0, the RTC wakeup event (alarm, RTC Tamper, RTC TimeStamp or RTC wakeup time) does not wake up the system from Standby mode.

Workaround

For each write access on the PWR_CSR1 register to enable or disable the back-up SRAM regulator, the EIWKUP bit must be set to 1 in order to enable a wakeup from Standby mode using RTC events.

2.2.3 LSE high driving and low driving capability is not usable for TFBGA216 package under certain conditions

Description

On the TFBGA216 package when the LSE low driving capability or LSE high driving capability is selected (LSEDRV[1:0]=00 or LSEDRV[1:0]=11 in the RCC_BDCR register, respectively) for the LSE oscillator, the oscillation stability is impacted by toggling the MCU pins near the LSE input pin at relatively high-frequency.

The TFBGA216 pins impacting the LSE stability are: PF0, PF1, PI11 and PI12.

Under the above described conditions, intermittent LSE clock pulse losses (in low driving capability) or intermittent LSE clock pulse add-ons (in high driving capability) are possible.

Workaround

On the TFBGA216 package do not select the LSE high driving capability or the LSE low driving capability, and:

- Use the LSE medium high driving capability (LSEDRV[1:0]=01 in the RCC_BDCR register)
- Or the LSE medium low driving capability (LSEDRV[1:0]=10 in the RCC_BDCR register)

2.2.4 DTCM-RAM not accessible in read when the MCU is in Sleep mode (WFI/WFE)

Description

The DTCM-RAM is not accessible in read during Sleep mode (when the CPU clock is gated). When a read access to the DTCM-RAM is performed by an AHB bus master (that are the DMAs) while the CPU is in sleep mode (CPU clock is gated), the data is not transmitted to the AHB bus and the AHB master reads 0x0000_0000.

There is no issue when a write is performed to the DTCM-RAM while the CPU is in sleep mode, the data is correctly written in the DTCM-RAM.

Workaround

Use the AXI SRAM1 or SRAM2 for DMA data read transfers and use the AXI DTCM-RAM for DMA data write transfers in Sleep mode.

2.2.5 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 or for an alternate function other than NJTRST. Only the 4-wire JTAG port configuration is impacted.

Workaround

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

2.2.6 PC13 signal transitions disturb LSE

Description

The PC13 port toggling disturbs the LSE clock.

Workaround

None.

2.3 FMC

2.3.1 Dummy read cycles inserted when reading synchronous memories

Description

When performing a burst read access from a synchronous memory, two dummy read accesses are performed at the end of the burst cycle whatever the type of burst access.

The extra data values read are not used by the FMC and there is no functional failure.

Workaround

None.

2.3.2 Missing information on prohibited 0xFF value of NAND transaction wait timing

Description

Some reference manual revisions may omit the information that the value 0xFF is prohibited for the wait timing of NAND transactions in their corresponding memory space (common or attribute).

Whatever the setting of the PWAITEN bit of the FMC_PCRx register, the wait timing set to 0xFF would cause a NAND transaction to stall the system with no fault generated.

This is a documentation error rather than a device limitation.

Workaround

No application workaround required provided that the 0xFF wait timing value is duly avoided.

2.3.3 Wrong data read from a busy NAND memory

Description

When a read command is issued to the NAND memory, the R/B signal gets activated upon the de-assertion of the chip select. If a read transaction is pending, the NAND controller might not detect the R/B signal (connected to NWAIT) previously asserted and sample a wrong data. This problem occurs only when the MEMSET timing is configured to 0x00 or when ATTHOLD timing is configured to 0x00 or 0x01.

Workaround

Either configure MEMSET timing to a value greater than 0x00 or ATTHOLD timing to a value greater than 0x01.

2.3.4 Spurious clock stoppage with continuous clock feature enabled

Description

With the continuous clock feature enabled, the FMC_CLK clock may spuriously stop when:

- the FMC_CLK clock is divided by 2, and
- an FMC bank set as 32-bit is accessed with a byte access.

division ratio set to 2, the FMC_CLK clock may spuriously stop upon an

Note: With static memories, a spuriously stopped clock can be restarted by issuing a synchronous transaction or any asynchronous transaction different from a byte access on 32-bit data bus width.

Workaround

With the continuous clock feature enabled, do not set the FMC_CLK clock division ratio to 2 when accessing 32-bit asynchronous memories with byte access.

2.3.5 Data read might be corrupted when the write FIFO is disabled

Description

When the write FIFO is disabled, the FIFO empty event is generated for every write access. During a write access, if a new read access occurs, the FMC grants the read access and waits till the FIFO gets empty. If another read access occurs in a very short window (one cycle of the FIFO empty event), the returned data are corrupted. This issue occurs only when the write FIFO is disabled (the WFDIS bit of the FMC_BCR1 register is set).

Workaround

Enable the write FIFO.

2.4 QUADSPI

2.4.1 First nibble of data not written after dummy phase

Description

The first nibble of data to be written to the external Flash memory is lost when the following condition is met:

- QUADSPI is used in indirect write mode.
- At least one dummy cycle is used.

Workaround

Use alternate bytes instead of dummy phase to add latency between the address phase and the data phase. This works only if the number of dummy cycles to substitute corresponds to a multiple of eight bits of data.

Example:

- To substitute one dummy cycle, send one alternate byte (only possible in DDR mode with four data lines).
- To substitute two dummy cycles, send one alternate byte in SDR mode with four data lines.
- To substitute four dummy cycles, send two alternate bytes in SDR mode with four data lines, or one alternate byte in SDR mode with two data lines.
- To substitute eight dummy cycles, send one alternate byte in SDR mode with one data line.

2.4.2 Wrong data from memory-mapped read after an indirect mode operation

Description

The first memory-mapped read in indirect mode can yield wrong data if the QUADSPI peripheral enters memory-mapped mode with bits ADDRESS[1:0] of the QUADSPI_AR register both set.

Workaround

Before entering memory-mapped mode, apply the following measure, depending on access mode:

- Indirect read mode: clear the QUADSPI_AR register then issue an abort request to stop reading and to clear the BUSY bit.
- Indirect write mode: clear the QUADSPI_AR register.

Caution: The QUADSPI_DR register must not be written after clearing the QUADSPI_AR register.

2.4.3 Memory-mapped read operations may fail when timeout counter is enabled

Description

In memory-mapped mode with the timeout counter enabled (by setting the TCEN bit of the QUADSPI_CR register), the QUADSPI peripheral may hang and memory-mapped read operation fail. This occurs if the timeout flag TOF is set at the same clock edge as a new memory-mapped read request.

Workaround

Disable the timeout counter. To raise the chip select, perform an abort at the end of each memory-mapped read operation.

2.4.4 Memory-mapped access in indirect mode clearing QUADSPI_AR register

Description

Memory-mapped accesses to the QUADSPI peripheral operating in indirect mode unduly clear the QUADSPI_AR register to 0x00.

Workaround

Adopt one of the following measures:

- Avoid memory-mapped accesses to the QUADSPI peripheral operating in indirect mode.
- After each memory-mapped access to the QUADSPI operating in indirect mode, write the QUADSPI_AR register with a desired value

2.5 ADC

2.5.1 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.

2.6 DAC

2.6.1 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.

2.6.2 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.

2.6.3 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 serving the DAC.

Workaround

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

2.7 DSI

2.7.1 Tearing effect parasitic detection

Description

When using the tearing effect mechanism over the DSI link in the Adapted Command mode, the tearing effect interrupt flag (TEIF) of the DSI wrapper interrupt status register (DSI_WISR) is asserted when an acknowledge trigger is received from the display.

An acknowledge trigger can be received from the display:

- for each packet when the acknowledge request enable (ARE) bit of the DSI Host command mode configuration register (DSI_CMCR) is set
- when a display response is expected

Workaround

Do not use the tearing effect over the link but use the dedicated TE pin.

When using the tearing effect over the link, do not use the tearing effect interrupt nor the automatic refresh mode. Instead, launch the display refresh immediately after a set_tear_on or a set_scanline DCS command (as the display is driving the DSI link until the tearing effect occurs, the refresh is automatically stalled until the tearing effect occurs).

2.7.2 Incorrect calculation of the time to activate the clock between HS transmissions

Description

In the automatic clock lane control mode, the DSI Host can turn off the clock lane between two high-speed transmissions.

To do so, the DSI Host calculates the time required for the clock lane to change from either: high-speed to low-power, or from low-power to high-speed.

These timings are configured by the HS2LP_TIME[9:0] and LP2HS_TIME[9:0] bitfields of the DSI Host clock lane timer configuration register (DSI_CLTCR). The DSI Host does not calculate the value configured in LP2HS_TIME plus HS2LP_TIME but twice the value configured in HS2LP_TIME instead.

Workaround

Configure HS2LP_TIME and LP2HS_TIME with the same value as the maximum of either HS2LP_TIME and LP2HS_TIME.

As an example, if HS2LP_TIMER = 44 and LP2HS_TIME = 113 configure the register fields as follows:

- HS2LP_TIME = 113
- LP2HS_TIME =113

2.7.3 The immediate update procedure may fail

Description

The immediate update procedure implies that both the UR and the EN bits of the DSI Host video shadow control register (DSI_VSCR) are initially cleared, and are set by the same instruction.

In some cases, the immediate update procedure fails due to a race condition between the two signals. This leads the DSI Host to wait for the next frame end before updating the configuration.

Workaround

After an immediate update procedure, check the configuration is updated by reading the auto-cleared bit UR.

If the UR bit is not cleared, repeat the process by writing first 0x0000 then 0x0101 in DSI_VSCR.

2.7.4 When used over the DSI link, the tearing effect interrupt flag is set when an acknowledge trigger is received from the display

Description

In the adapted command mode, when the tearing effect mechanism is used over the DSI link, the tearing effect interrupt flag (TEIF) of the DSI wrapper interrupt status register (DSI_WISR) is asserted when an acknowledge trigger is received from the display.

In the adapted command mode, when the tearing effect mechanism is used over the DSI link, the tearing effect interrupt Flag (TEIF) of the DSI wrapper interrupt status register (DSI_WISR) is asserted when an acknowledge trigger is received from the display.

- For each packet, when the acknowledge request enable (ARE) bit of the DSI Host command mode configuration register (DSI_CMCR) is set.
- When a response is awaited from the display.

Workaround

Do not use the tearing effect over the link, but use the dedicated TE pin.

When using the tearing effect over the link, do not use the tearing effect interrupt nor the automatic refresh mode. Instead, launch the display refresh immediately after a set_tear_on or a set_scanline DCS command (as the display is driving the DSI link until the tearing effect occurs, the refresh is automatically stalled until the tearing effect).

2.8 JPEG

2.8.1 False EOI marker is inserted after clearing the HDR bit

Description

An extra end of image (EOI) marker (0xFFD9) is written automatically into the output FIFO at the end of an encoding process with header processing. If the HDR mode is enabled and when the software clears the HDR bit at the end of the encoding process before making a software reset, an extra data (EOI marker = 0xFFD9) is inserted into the output FIFO. it implies that the extra data might be outputted from the FIFO and stored in a RAM

Workaround

The software must clear the EOC flag and perform a software reset before changing the HDR bit configuration in the JPEG codec configuration register 1 (JPEG_CONFR1).

2.8.2 No DMA transfer complete generated at the end of the encoding process after clearing the HDR bit

Description

If the JPEG is configured as the DMA flow controller, the DMA might enter an infinite wait due to the fact that the JPEG does not generate the correct last data request for it.

The JPEG might not generate the correct last request if the following conditions are met:

- The software clears the HDR bit at the end of the encoding process.
- The encoding process has the header processing enabled (the EOC flag is asserted and no software reset is performed).
- And the FIFO level is equal to the threshold.

Workaround

The software must clear the EOC flag and perform a software reset before changing the HDR bit configuration in the JPEG codec configuration register 1 (JPEG_CONFR1).

Or use the DMA as the flow controller.

2.8.3 JPEG FIFO might be corrupted

Description

The JPEG can be accessed in a concurrent way with another peripherals (OTGFS, RNG, HASH, CRYP and DCMI) on the same AHB bus. This might result in a dummy read/write access to the JPEG peripheral. as a consequence it can lead to a wrong data written into the input FIFO, or a data loss from the output FIFO.

Workaround

Avoid a concurrent access between the JPEG access and other peripherals on the same AHB bus.

2.9 HASH

2.9.1 Superseded suspend sequence for data loaded by DMA

Description

The section *HASH / Context swapping / Data loaded by DMA / Current context saving* of some reference manual revisions may suggest the following suspend sequence for using HASH with DMA:

1. Clear the DMAE bit to disable the DMA interface.
2. Wait until the current DMA transfer is complete (wait for DMAS = 0 in the HASH_SR register).

This recommendation is obsolete and superseded with the following sequence that suspends then resumes the secure digest computing in order to swap the context:

Suspend:

1. In Polling mode, wait for BUSY = 0. If the DCIS bit of the HASH_SR register is set, the hash result is available and the context swapping is useless. Otherwise, go to step 2.
2. In Polling mode, wait for BUSY = 1.
3. Disable the DMA channel. Then clear the DMAE bit of the HASH_CR register.
4. In Polling mode, wait for BUSY = 0. If the DCIS bit of the HASH_SR register is set, the hash result is available and the context swapping is useless. Otherwise, go to step 5.
5. Save the HASH_IMR, HASH_STR, HASH_CR, and HASH_CSR0 to HASH_CSR37 registers. The HASH_CSR38 to HASH_CSR53 registers must also be saved if an HMAC operation is ongoing.

Resume:

1. Reconfigure the DMA controller so that it proceeds with the transfer of the message up to the end if it is not interrupted again. Do not forget to take into account the words already pushed into the FIFO if NBW[3:0] is higher than 0x0.
2. Program the values saved in memory to the HASH_IMR, HASH_STR, and HASH_CR registers.
3. Initialize the hash processor by setting the INIT bit of the HASH_CR register.
4. Program the values saved in memory to the HASH_CSRx registers.
5. Restart the processing from the point of interruption, by setting the DMAE bit.

Note: To optimize the resume process when NBW[3:0] = 0x0, HASH_CSR22 to HASH_CSR37 registers do not need to be saved then restored as the FIFO is empty.

This is a documentation issue rather than a product limitation.

Workaround

No application workaround is required as long as the new sequence is applied.

2.9.2 Superseded suspend sequence for data loaded by the CPU

Description

The section *HASH / Context swapping / Data loaded by software* of some reference manual revisions may instruct that *“the user application must wait until DINIS ≠ 1 (last block processed and input FIFO empty) or NBW 0 (FIFO not full and no processing ongoing)”*.

This instruction is obsolete and superseded with the following:

When the DMA is not used to load the message into the hash processor, the context can be saved only when no block processing is ongoing.

To suspend the processing of a message, proceed as follows after writing 16 words 32-bit (plus one if it is the first block):

1. In Polling mode, wait for BUSY = 0, then poll if the DINIS status bit is set to 1. In Interrupt mode, implement the next step in DINIS interrupt handler (recommended).

2. Store the contents of the following registers into memory:
   - HASH_IMR
   - HASH_STR
   - HASH_CR
   - HASH_CSR0 to HASH_CSR37 and, if an HMAC operation is ongoing, also HASH_CSR38 to HASH_CSR53

To resume the processing of a message, proceed as follows:

1. Write the HASH_IMR, HASH_STR, and HASH_CR registers with the values saved in memory.
2. Initialize the hash processor by setting the INIT bit of the HASH_CR register.
3. Write the HASH_CSRx registers with the values saved in memory.
4. Restart the processing from the point of interruption.

Note: To optimize the resume process when NBW[3:0]=0x0, HASH_CSR22 to HASH_CSR37 registers do not need to be saved then restored as the FIFO is empty.

This is a documentation issue rather than a product limitation.

Workaround

No application workaround is required as long as the new sequence is applied.

2.10 TIM

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

Description

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

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

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

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

Workaround

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

2.10.2 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.

2.10.3 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).

2.10.4 TIM12 input XOR function not available

Description

The reference manual states incorrectly that for TIM12, a timer input XOR function is enabled with the TI1S bit of the TIM12_CR2 register. This feature is not available. Nevertheless the TIM12_CR2 register description is correct.

Workaround

None

2.11 LPTIM

2.11.1 Device may remain stuck in LPTIM interrupt when entering Stop mode

Description

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

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

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

Workaround

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

2.11.2 Device may remain stuck in LPTIM interrupt when clearing event flag

Description

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

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

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

Workaround

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

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

Note: The proper clear sequence is already implemented in the HAL_LPTIM_IRQHandler in the STM32Cube.

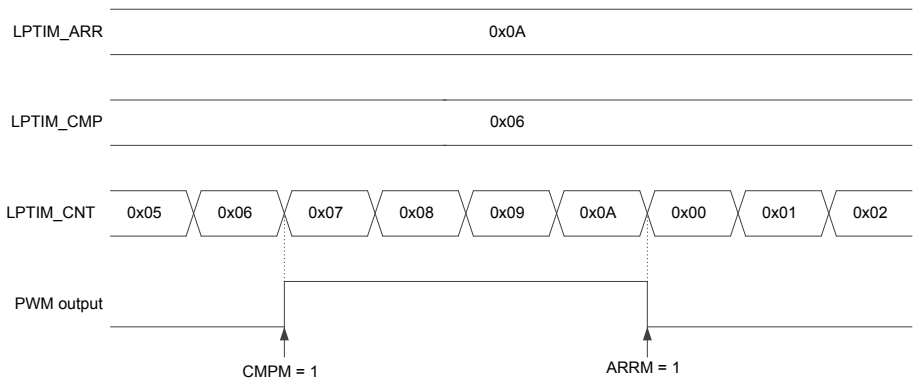
2.11.3 LPTIM events and PWM output are delayed by 1 kernel clock cycle

Description

The compare match event (CMPM), auto reload match event (ARRM), PWM output level and interrupts are updated with a delay of one kernel clock cycle.

Consequently, it is not possible to generate PWM with a duty cycle of 0% or 100%.

The following waveform gives the example of PWM output mode and the effect of the delay:



Workaround

Set the compare value to the desired value minus 1. For instance in order to generate a compare match when LPTM_CNT = 0x08, set the compare value to 0x07.

2.12 IWDG

2.12.1 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.

2.12.2 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.

2.12.3 WVU flag not reset in Stop

Description

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

Workaround

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

2.12.4 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.

2.12.5 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.

2.12.6 WVU flag not cleared at low APB clock frequency

Description

Successful write to the IWDG_WINR register raises the WVU flag and prevents further write accesses to the register until the WVU 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_WINR register is no longer possible.

Workaround

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

2.13 RTC and TAMP

2.13.1 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.

2.13.2 RTC interrupt can be masked by another RTC interrupt

Description

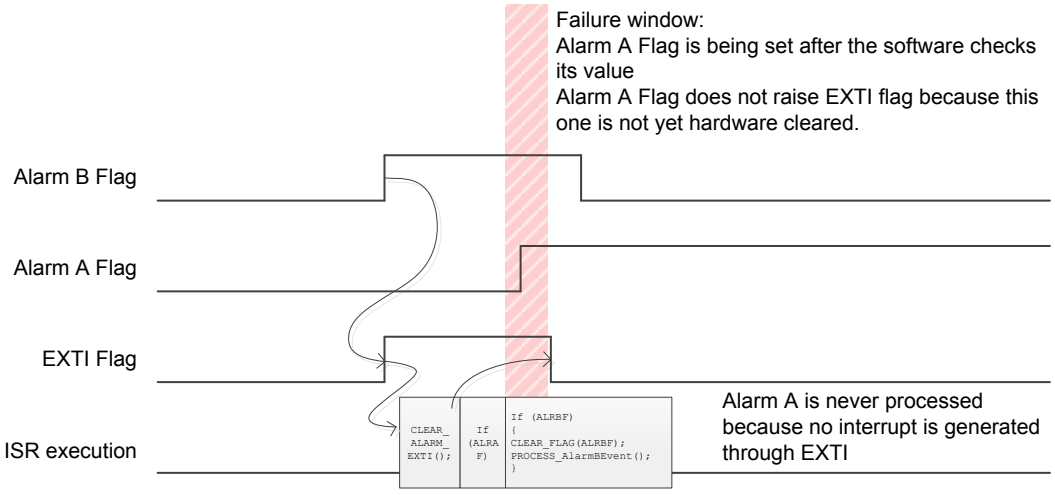
One RTC interrupt request can mask another RTC interrupt request if they share the same EXTI configurable line. For example, interrupt requests from Alarm A and Alarm B or those from tamper and timestamp events are OR-ed to the same EXTI line (refer to the *EXTI line connections* table in the *Extended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



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 */
    }
}
```

2.13.3 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.

2.13.4 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 though 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 tentative to clear the flag(s) ineffective.

Workaround

None.

2.13.5 Setting GPIO properties of PC13 used as RTC_ALARM open-drain output

Description

Some reference manual revisions may omit the information that the PC13 GPIO must be set as input when the RTC_OR register configures PC13 as open-drain output of the RTC_ALARM signal.

Note: Enabling the internal pull-up function through the PC13 GPIO settings allows sparing an external pull-up resistor.

This is a documentation issue rather than a product limitation.

Workaround

No application workaround is required provided that the described GPIO setting is respected.

2.14 I2C

2.14.1 10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave

Description

An I^2C-bus master generates STOP condition upon non-acknowledge of I^2C address that it sends. This applies to 7-bit address as well as to each byte of 10-bit address.

When the MCU set as I^2C-bus master transmits a 10-bit address of which the first byte (5-bit header + 2 MSBs of the address + direction bit) is not acknowledged, the MCU duly generates STOP condition but it then cannot start any new I^2C-bus transfer. In this spurious state, the NACKF flag of the I2C_ISR register and the START bit of the I2C_CR2 register are both set, while the START bit should normally be cleared.

Workaround

In 10-bit-address master mode, if both NACKF flag and START bit get simultaneously set, 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.

2.14.2 Wrong behavior in Stop mode

Description

The correct use of the I2C peripheral is to disable it (PE = 0) before entering Stop mode, and re-enable it when back in Run mode.

Some reference manual revisions may omit this information.

Failure to respect the above while the MCU operating as slave or as master in multi-master topology enters Stop mode during a transfer ongoing on the I^2C-bus may lead to the following:

1. BUSY flag is wrongly set when the MCU exits Stop mode. This prevents from initiating a transfer in master mode, as the START condition cannot be sent when BUSY is set.
2. If clock stretching is enabled (NOSTRETCH = 0), the SCL line is pulled low by I2C and the transfer stalled as long as the MCU remains in Stop mode.

   The occurrence of such condition depends on the timing configuration, peripheral clock frequency, and I^2C-bus frequency.

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

Workaround

No application workaround is required.

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

Description

The I^2C-bus specification and user manual specify a minimum data setup time ($t_{SU;DAT}$) as:

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

The device does not correctly sample the I^2C-bus SDA line when $t_{SU;DAT}$ is smaller than one I2C kernel clock (I^2C-bus peripheral clock) period: the previous SDA value is sampled instead of the current one. This can result in a wrong receipt of slave address, data byte, or acknowledge bit.

Workaround

Increase the I2C kernel clock frequency to get I2C kernel clock period within the transmitter minimum data setup time. Alternatively, increase transmitter's minimum data setup time. If the transmitter setup time minimum value corresponds to the minimum value provided in the I^2C-bus standard, the minimum I2CCLK frequencies are as follows:

- In Standard mode, if the transmitter minimum setup time is 250 ns, the I2CCLK frequency must be at least 4 MHz.
- In Fast mode, if the transmitter minimum setup time is 100 ns, the I2CCLK frequency must be at least 10 MHz.
- In Fast-mode Plus, if the transmitter minimum setup time is 50 ns, the I2CCLK frequency must be at least 20 MHz.

2.14.4 Spurious bus error detection in master mode

Description

In master mode, a bus error can be detected spuriously, with the consequence of setting the BERR flag of the I2C_SR register and generating bus error interrupt if such interrupt is enabled. Detection of bus error has no effect on the I^{2}C-bus transfer in master mode and any such transfer continues normally.

Workaround

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

2.14.5 Last-received byte loss in reload mode

Description

If in master receiver mode or slave receive mode with SBC = 1 the following conditions are all met:

- I^{2}C-bus stretching is enabled (NOSTRETCH = 0)
- RELOAD bit of the I2C_CR2 register is set
- NBYTES bitfield of the I2C_CR2 register is set to N greater than 1
- byte N is received on the I^{2}C-bus, raising the TCR flag
- N - 1 byte is not yet read out from the data register at the instant TCR is raised,

then the SCL line is pulled low (I^{2}C-bus clock stretching) and the transfer of the byte N from the shift register to the data register inhibited until the byte N-1 is read and NBYTES bitfield reloaded with a new value, the latter of which also clears the TCR flag. As a consequence, the software cannot get the byte N and use its content before setting the new value into the NBYTES field.

For I2C instances with independent clock, the last-received data is definitively lost (never transferred from the shift register to the data register) if the data N - 1 is read within four APB clock cycles preceding the receipt of the last data bit of byte N and thus the TCR flag raising. Refer to the product reference manual or datasheet for the I2C implementation table.

Workaround

- In master mode or in slave mode with SBC = 1, use the reload mode with NBYTES = 1.
- In master receiver mode, if the number of bytes to transfer is greater than 255, do not use the reload mode. Instead, split the transfer into sections not exceeding 255 bytes and separate them with repeated START conditions.
- Make sure, for example through the use of DMA, that the byte N - 1 is always read before the TCR flag is raised. Specifically for I2C instances with independent clock, make sure that it is always read earlier than four APB clock cycles before the receipt of the last data bit of byte N and thus the TCR flag raising.

The last workaround in the list must be evaluated carefully for each application as the timing depends on factors such as the bus speed, interrupt management, software processing latencies, and DMA channel priority.

2.14.6 Spurious master transfer upon own slave address match

Description

When the device is configured to operate at the same time as master and slave (in a multi- master I^{2}C-bus application), a spurious master transfer may occur under the following condition:

- Another master on the bus is in process of sending the slave address of the device (the bus is busy).
- The device initiates a master transfer by bit set before the slave address match event (the ADDR flag set in the I2C_ISR register) occurs.
- After the ADDR flag is set:
  - the device does not write I2C_CR2 before clearing the ADDR flag, or
  - the device writes I2C_CR2 earlier than three I2C kernel clock cycles before clearing the ADDR flag

In these circumstances, even though the START bit is automatically cleared by the circuitry handling the ADDR flag, the device spuriously proceeds to the master transfer as soon as the bus becomes free. The transfer configuration depends on the content of the I2C_CR2 register when the master transfer starts. Moreover, if the I2C_CR2 is written less than three kernel clocks before the ADDR flag is cleared, the I2C peripheral may fall into an unpredictable state.

Workaround

Upon the address match event (ADDR flag set), apply the following sequence.

Normal mode (SBC = 0):

1. Set the ADDRCF bit.
2. Before Stop condition occurs on the bus, write I2C_CR2 with the START bit low.

Slave byte control mode (SBC = 1):

1. Write I2C_CR2 with the slave transfer configuration and the START bit low.
2. Wait for longer than three I2C kernel clock cycles.
3. Set the ADDRCF bit.
4. Before Stop condition occurs on the bus, write I2C_CR2 again with its current value.

The time for the software application to write the I2C_CR2 register before the Stop condition is limited, as the clock stretching (if enabled), is aborted when clearing the ADDR flag.

Polling the BUSY flag before requesting the master transfer is not a reliable workaround as the bus may become busy between the BUSY flag check and the write into the I2C_CR2 register with the START bit set.

2.14.7 OVR flag not set in underrun condition

Description

In slave transmission with clock stretching disabled (NOSTRETCH = 1 in the I2C_CR1 register), an underrun condition occurs if the current byte transmission is completed on the I2C bus, and the next data is not yet written in the TXDATA[7:0] bitfield. In this condition, the device is expected to set the OVR flag of the I2C_ISR register and send 0xFF on the bus.

However, if the I2C_TXDR is written within the interval between two I2C kernel clock cycles before and three APB clock cycles after the start of the next data transmission, the OVR flag is not set, although the transmitted value is 0xFF.

Workaround

None.

2.14.8 Transmission stalled after first byte transfer

Description

When the first byte to transmit is not prepared in the TXDATA register, two bytes are required successively, through TXIS status flag setting or through a DMA request. If the first of the two bytes is written in the I2C_TXDR register in less than two I2C kernel clock cycles after the TXIS/DMA request, and the ratio between APB clock and I2C kernel clock frequencies is between 1.5 and 3, the second byte written in the I2C_TXDR is not internally detected. This causes a state in which the I2C peripheral is stalled in master mode or in slave mode, with clock stretching enabled (NOSTRETCH = 0). This state can only be released by disabling the peripheral (PE = 0) or by resetting it.

Workaround

Apply one of the following measures:

- Write the first data in I2C_TXDR before the transmission starts.
- Set the APB clock frequency so that its ratio with respect to the I2C kernel clock frequency is lower than 1.5 or higher than 3.

2.15 USART

2.15.1 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.

2.15.2 Receiver timeout counter wrong start in two-stop-bit configuration

Description

In two-stop-bit configuration, the receiver timeout counter starts counting from the end of the second stop bit of the last character instead of starting from the end of the first stop bit.

Workaround

Subtract one bit duration from the value in the RTO bitfield of the USARTx_RTOR register.

2.15.3 Data corruption due to noisy receive line

Description

In UART mode with oversampling by 8 or 16 and with 1 or 2 stop bits, the received data may be corrupted if a glitch to zero shorter than the half-bit occurs on the receive line within the second half of the stop bit.

Workaround

None.

2.16 SPI

2.16.1 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.

2.16.2 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.

2.16.3 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.

2.16.4 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.

2.17 SAI

2.17.1 Automatic restart upon late or anticipated frame error in I²S slave mode not supported

Description

Some reference manual revisions may omit the following information.

In I²S (FSDEF = 1) slave mode, upon detecting a late or anticipated frame error in the midst of an audio frame, the corresponding flag is duly set and the transfer restarted upon the detection of the next start of frame, but the FIFO contents may get desynchronized with respect to data slots.

Therefore, upon late or anticipated frame error detection, the SAI peripheral must be resynchronized with the master through the following sequence:

1. Disable SAI by clearing the SAIXEN bit of the SAI_xCR1 register (wait until the bit returns zero upon read).
2. Flush the FIFO via the FFLUSH bit of the SAI_xCR2 register.
3. Enable SAI by setting the SAIXEN bit.

The resynchronization with the master starts upon the nearest FS line active state.

This is a documentation issue rather than a device limitation.

Workaround

No application workaround is applicable or required if the instruction, as described, is respected.

2.17.2 Last SAI_SCK clock pulse truncated upon disabling SAI master

Description

When disabling, during the communication, the SAI peripheral configured as master, it may truncate the last SAI_SCK bit clock pulse of the transaction, potentially causing a failure to the external codec logic.

Workaround

None.

2.17.3 Last SAI_MCLK clock pulse truncated upon disabling SAI master

Description

When disabling, during the communication, the SAI peripheral configured as master with the OUTDRIV bit of the corresponding SAI_xCR1 register cleared, the device may truncate the last SAI_MCLK_x bit clock pulse of the transaction, potentially causing a failure to the external codec logic.

Workaround

Set the OUTDRIV bit of the corresponding SAI_xCR1 register.

2.17.4 SAI_MCLK clock absent in a specific configuration

Description

When configured as master with PRTCFG[1:0] = 00, NODIV = 0, and MCKDIV = 0 in the SAI_xCR1 register of the audio sub-block *x*, and FRL = 0xFF in the corresponding SAI_xFRCR register, the SAI peripheral fails to generate the master clock on the corresponding SAI_MCLK_x output.

As in this configuration, the master and bit clocks are identical, the application can use the SAI_SCK_x output also as master clock line. The absence of clock signal on the SAI_MCLK_x output allows saving power.

Workaround

Apply one of the following measures:

- In the application, use SAI_SCK_x as master and bit clock output. Optionally, disable the SAI_MCLK_x master clock output by setting NODIV.

- Configure the RCC block to set SAI kernel clock frequency to an integer multiple of the desired SAI_MCLK_x master clock frequency. Set the MCKDIV[5:0] bitfield so as to obtain the desired SAI_MCLK_x master clock frequency.

2.18 SDMMC

2.18.1 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 SDMMC_STA register is set.

Workaround

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

2.18.2 MMC stream write of less than seven bytes does not work correctly

Description

Stream write initiated with WRITE_DAT_UNTIL_STOP command (CMD20) does not define the amount of data bytes to store. The card keeps storing data coming in from the SDMMC host until it gets a valid STOP_TRANSMISSION (CMD12) command. The commands are streamed on a line separate from data line, with common clock line.

As the STOP_TRANSMISSION command is 48-bit long and due to the bus protocol, the STOP_TRANSMISSION command start bit must be advanced by 50 clocks with respect to the stop bit of the data bitstream.

Therefore, for small data chunks of up to six bytes, SDMMC hosts should normally operate such that, the start of the STOP_TRANSMISSION (CMD12) command streaming precedes the start of the data streaming.

The device duly anticipates the STOP_TRANSMISSION command streaming start, with respect to the data bitstream end. WAITPEND (bit 9 of SDMMC_CMD register) must be set for this mechanism to operate.

However, a failure occurs in case of small data chunks of up to six bytes. Instead of starting the STOP_TRANSMISSION command 50 clocks ahead of the data bitstream stop bit, the SDMMC peripheral on the device starts the command along with the first bit of the data bitstream. As the command is longer than the data, it ends a number of clocks behind the data that the software intended to store onto the card by setting the DATALENGTH register. During the clocks in excess, the SDMMC peripheral keeps the data line in logical-one level.

As a consequence, the card intercepts more data and updates more memory locations than the number set in DATALENGTH. The spuriously updated locations of memory receive 0xFF values.

Workaround

Do not use stream write WRITE_DAT_UNTIL_STOP command (CMD20) with DATALENGTH set to less than seven. Instead, use SET_BLOCKLEN command (CMD16) followed with single-block write command WRITE_BLOCK (CMD24), with a desired block length.

2.19 bxCAN

2.19.1 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. The TTCM bit of the CAN_MCR register must be kept cleared (time-triggered communication mode disabled).

Workaround

None.

2.20 OTG_FS

2.20.1 Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_FS registers

Description

When the USB on-the-go full-speed peripheral is in Device mode, interrupting transmit FIFO write sequence with read or write accesses to OTG_FS endpoint-specific registers (those ending in *0* or *x*) leads to corruption of the next data written to the transmit FIFO.

Workaround

Ensure that the transmit FIFO write sequence is not interrupted with accesses to the OTG_FS registers.

2.20.2 Host packet transmission may hang when connecting through a hub to a low-speed device

Description

When the USB on-the-go full-speed peripheral connects to a low-speed device via a hub, the transmitter internal state machine may hang. This leads, after a timeout expiry, to a port disconnect interrupt.

Workaround

None. However, increasing the capacitance on the data lines may reduce the occurrence.

2.21 OTG_HS

2.21.1 Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_HS registers

Description

When the USB on-the-go high-speed peripheral is in Device mode, interrupting transmit FIFO write sequence with read or write accesses to OTG_HS endpoint-specific registers (those ending in *0* or *x*) leads to corruption of the next data written to the transmit FIFO.

Workaround

Ensure that the transmit FIFO write sequence is not interrupted with accesses to the OTG_HS registers. Note that enabling DMA mode guarantees this.

2.21.2 Host packet transmission may hang when connecting through a hub to a low-speed device

Description

When the USB on-the-go high-speed peripheral connects to a low-speed device via a hub, the transmitter internal state machine may hang. This leads, after a timeout expiry, to a port disconnect interrupt.

Workaround

None. However, increasing the capacitance on the data lines may reduce the occurrence.

2.22 ETH

2.22.1 Incorrect L3 checksum is inserted in transmitted IPv6 packets without TCP, UDP or ICMP payloads

Description

The application provides a frame-by-frame control to instruct the MAC to insert the layer 3 (L3) checksums for TCP, UDP and ICMP packets. When automatic checksum insertion is enabled and the input packet is an IPv6 packet without the TCP, UDP or ICMP payload, then the MAC may incorrectly insert a checksum into the packet. For IPv6 packets without a TCP, UDP or ICMP payload, the MAC core considers the next header (NH) field as the extension header and continues to parse the extension header. Sometimes, the payload data in such packets matches the NH field for TCP, UDP or ICMP and, as a result, the MAC core inserts a checksum.

Workaround

When the IPv6 packets have a TCP, UDP or ICMP payload, enable checksum insertion for transmit frames, or bypass checksum insertion by using the CIC bits of the TDES0 transmit descriptor word0.

2.22.2 The ethernet MAC processes invalid extension headers in the received IPv6 frames

Description

In IPv6 frames, the extension headers which precede the actual IP payload may or may not be present. The Ethernet MAC processes the following extension headers defined in the IPv6 protocol: hop-by-hop options header, routing header and destination options header.

All extension headers, except the hop-by-hop extension header, can be present multiple times and in any order before the actual IP payload. The hop-by-hop extension header, if present, has to come immediately after the IPv6 main header.

The Ethernet MAC processes all extension headers whether valid or invalid including the hop-by-hop extension headers that are present after the first extension header. For this reason, the GMAC core will accept IPv6 frames with invalid hop-by-hop extension headers. As a consequence, it will accept any IP payload as valid IPv6 frames with TCP, UDP or ICMP payload, and then incorrectly update the receive status of the corresponding frame.

Workaround

None.

2.22.3 MAC stuck in the idle state on receiving the TxFIFO flush command exactly one clock cycle after a transmission completes

Description

When the software issues a TxFIFO flush command, the transfer of frame data stops, even in the middle of a frame transfer. The TxFIFO read controller goes into the Idle state by clearing the TFRS [1:0] bit field of the ETH_MACDBGR register. It then resumes its normal operation.

However, if the TxFIFO read controller receives the TxFIFO flush command exactly one clock cycle after receiving the status from the MAC, the controller remains stuck in the Idle state and stops transmitting frames from the TxFIFO. The system only recovers from this state with a reset (for example a soft reset).

Workaround

Wait until the TxFIFO is empty before using the TxFIFO flush command.

2.22.4 Transmit frame data corruption

Description

Frame data may get corrupted when the TxFIFO repeatedly switches from non-empty to empty, and back to non-empty again for a very short period, without causing any underflow.

The issue occurs when switching back and forth between non-empty and empty happens when the rate the data is being written to the TxFIFO is almost equal to or a little slower than the rate at which the data is read.

This corruption cannot be detected by the receiver when the CRC is inserted by the MAC, as the corrupted data is used for the CRC computation.

Workaround

Use the transmit Store-and-Forward mode by setting the TSF bit of the ETH_DMAOMR register. In this mode, the data is transmitted only when the whole packet is available in the TxFIFO.

2.22.5 Incorrect status and corrupted frames when RxFIFO overflow occurs on the penultimate word of Rx frames

Description

When operating in Threshold mode, the RxFIFO may overflow when the received frame data is written faster than the speed at which the application reads it from the RxFIFO. The RxFIFO overflow is declared at the moment that a non-EOF word is received and the RxFIFO has only two locations available. The receiver descriptor overflow error (OE) bit of the RDES0 receive descriptor word0 is set to indicate that the receive frame is incomplete.

The problem occurs after the following events:

1. RxFIFO overflow is declared exactly on the penultimate word of the Rx Frame.

   The EOF word is received in the next clock cycle.
2. The EOF word has exactly one valid byte. This is possible only when the length of the packet, after CRC or PAD stripping (if enabled), is a multiple of 4 bytes plus 1 (for example, 5, 9, 13, 17).

After the above sequence, the frames status information is corrupted and the overflow error flag is not set. Furthermore, if the next frame arrives soon enough, the MAC might falsely interpret that there is space in the RxFIFO and overwrite unread data with the next frame, thus corrupting the existing frames.

The MAC recovers automatically after transferring a few corrupt or incorrect packets.

Workaround

Operate the RxFIFO in the Store-and-Forward mode.

2.22.6 Ethernet erroneous data received in RMII configuration

Description

In the reduced media-independent interface (RMII) configuration, an erroneous data might be received on the RXD0 signal (PC4). The bit received might flip from 0 to 1 and lead to a received frame with a CRC error. The ETH_MMCRFCECR register increments each time a frame is received. This is related to internal timing constraints on the reference clock generated after the sync divider.

Using the RMII reference clock of 50 MHz, the error is seen for both cases:

- 100 Mbit/s operating rate (sync divider = div2)
- 10 Mbit/s operating rate (sync divider = div20)

The issue is not present in the MII mode with a direct reference clock from the pad (no division).

Workaround

Using the MAC management counters, the software can identify if the RMII is correctly initialized or not.

- If too many errors are detected during the initialization (received frames with CRC error counter), reset the RMII interface and restart the monitoring.
- If a good frame is received after initialization (received good unicast frame counter register), the RMII is correctly initialized and stop the monitoring.

Revision history

Table 5. **Document revision history**

Date	Version	Changes
18-Feb-2016	1	Initial release.
21-Apr-2016	2	Added QUADSPI peripheral limitation: Section 2.4.1: First nibble of data not written after a dummy phase. Added system limitation: Section 2.2.4: LSE high driving and low driving capability is not usable for TFBGA216 package under certain conditions.
29-Sep-2016	3	Added system limitation: Section 2.2.5: DTCM-RAM not accessible in read when the MCU is in Sleep mode (WFI/WFE). Added ethernet limitation: Section 2.17.6: Ethernet erroneous data received in RMII configuration. Added JPEG limitations: • Section 2.8.1: False EOI marker is inserted after clearing the HDR bit. • Section 2.8.2: No DMA transfer complete generated at the end of the encoding process after clearing the HDR bit. Added I2C limitation: Section 2.12.3: 10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave. Removed USART limitations: • Start bit detected too soon when sampling for NACK signal from the smartcard. • Break request can prevent the Transmission Complete flag (TC) from being set. Updated Table 1: Device summary adding STM32F765xx devices.
21-Oct-2016	4	Added revision Z: • Updated Table 1: Device summary. • Updated Table 3: Summary of device errata with two system limitations and one ethernet limitation marked as ‘fixed’.
18-Jul-2018	5	FMC limitation: • Added Section 2.3.4: Data read might be corrupted when the write FIFO is disabled. QUADSPI limitation: • Updated Section 2.4.1: First nibble of data not written after a dummy phase. • Added Section 2.4.2: Wrong data from memory-mapped read after an indirect mode operation. • Added Section 2.4.3: Memory-mapped read operations may fail when timeout counter is enabled. JPEG limitation: • Added Section 2.8.3: JPEG FIFO might be corrupted. I2S/SPI limitations: • Moved Section 2.14.2: BSY bit may stay high at the end of a data transfer in Slave mode from I2C limitation to I2S/SPI limitation. • Updated Section 2.14.1: I2S slave in PCM short pulse mode sensitive to timing between WS and CK. RTC limitations: • Added Section 2.11.1: RTC calendar registers are not locked properly. I2C limitations: • Added Section 2.12.4: Last-received byte loss in reload mode. • Updated Section 2.12.1: Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period. ETHERNET limitation: • Updated Section 2.17.6: Ethernet erroneous data received in RMII configuration.

Date	Version	Changes
17-Jul-2019	6	Added: • Section 2.1.1: Cortex®-M7 data corruption when using data cache configured in write-through. • Section 2.2.3: Full JTAG configuration without NJTRST pin cannot be used.
09-Mar-2021	7	Added: • Section 2.2.6: PC13 signal transitions disturb LSE. • Section 2.9.1: TIM12 input XOR function not available. • Section 2.11.2: Setting GPIO properties of PC13 used as RTC_ALARM open-drain output.
1-Feb-2022	8	Added: • 2.3.2: Missing information on prohibited 0xFF value of NAND transaction wait timing • 2.4.4:Memory-mapped access in indirect mode clearing QUADSPI_AR register • 2.6.2: DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge • 2.7.1: Tearing effect parasitic detection • 2.7.2: Incorrect calculation of the time to activate the clock between HS transmissions • 2.9.1: Superseded suspend sequence for data loaded by DMA • 2.9.2: Superseded suspend sequence for data loaded by the CPU • 2.10.1: One-pulse mode trigger not detected in master-slave reset + trigger configuration • 2.10.2: Consecutive compare event missed in specific conditions • 2.10.3: Output compare clear not working with external counter reset • 2.11.2: Device may remain stuck in LPTIM interrupt when clearing event flag • 2.11.3: LPTIM events and PWM output are delayed by 1 kernel clock cycle • 2.12.1: RVU flag not reset in Stop • 2.12.2: PVU flag not reset in Stop • 2.12.3: WVU flag not reset in Stop • 2.12.4: RVU flag not cleared at low APB clock frequency • 2.12.5: PVU flag not cleared at low APB clock frequency • 2.12.6: WVU flag not cleared at low APB clock frequency • 2.13.2: RTC interrupt can be masked by another RTC interrupt • 2.13.3: Calendar initialization may fail in case of consecutive INIT mode entry • 2.13.4: Alarm flag may be repeatedly set when the core is stopped in debug • 2.14.2: Wrong behavior in Stop mode • 2.14.6: Spurious master transfer upon own slave address match • 2.14.7: OVR flag not set in underrun condition • 2.14.8: Transmission stalled after first byte transfer • 2.15.2: Receiver timeout counter wrong start in two-stop-bit configuration • 2.15.3: Data corruption due to noisy receive line • 2.16.1: BSY bit may stay high when SPI is disabled • 2.16.2: BSY bit may stay high at the end of data transfer in slave mode • 2.16.3: Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters • 2.16.4: CRC error in SPI slave mode if internal NSS changes before CRC transfer • 2.17.1: Automatic restart upon late or anticipated frame error in I^2S slave mode not supported

Date	Version	Changes
		• 2.17.2: Last SAI_SCK clock pulse truncated upon disabling SAI master • 2.17.3: Last SAI_MCLK clock pulse truncated upon disabling SAI master • 2.17.4: SAI_MCLK clock absent in a specific configuration • 2.20.1: Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_FS registers • 2.20.2: Host packet transmission may hang when connecting through a hub to a low-speed device • 2.21.1: Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_HS registers • 2.21.2: Host packet transmission may hang when connecting through a hub to a low-speed device

Contents