STM32H7A3xI/G, STM32H7B0xB and STM32H7B3xI device errata

Applicability

This document applies to the part numbers of STM32H7A3xI/G, STM32H7B0xB and STM32H7B3xI 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 RM0455.

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
STM32H7A3xI/G	STM32H7A3RI, STM32H7A3VI, STM32H7A3QI, STM32H7A3ZI, STM32H7A3AI, STM32H7A3II, STM32H7A3NI, STM32H7A3LI, STM32H7A3RG, STM32H7A3VG, STM32H7A3ZG, STM32H7A3AG, STM32H7A3IG, STM32H7A3NG
STM32H7B0xB	STM32H7B0AB, STM32H7B0IB, STM32H7B0RB, STM32H7B0VB, STM32H7B0ZB
STM32H7B3xI	STM32H7B3RI, STM32H7B3VI, STM32H7B3QI, STM32H7B3ZI, STM32H7B3AI, STM32H7B3II, STM32H7B3NI, STM32H7B3LI

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32H7A3xI	Z	0x1001
STM32H7A3xG	Z	0x1001
STM32H7B3xI	Z	0x1001
STM32H7B0xB	Z	0x1001
STM32H7A3xI	X	0x1007
STM32H7A3xG	X	0x1007
STM32H7B3xI	X	0x1007
STM32H7B0xB	X	0x1007

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

1 Summary of device errata

The following table gives a quick reference to the STM32H7A3xI/G, STM32H7B0xB and STM32H7B3xI 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. Z	Rev. X
Core	2.1.1	CPUID register returns r1p1 instead of r1p2	N	N
System	2.2.1	Performing a system reset during Flash memory program or erase operation is not supported	N	-
	2.2.2	LSE CSS parasitic detection even when disabled	P	P
	2.2.3	Secure firmware install (SFI) is not supported	N	N
BDMA	2.3.1	BDMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	A	A
DMAMUX	2.4.1	SOFx not asserted when writing into DMAMUX_CCFR register	N	N
	2.4.2	OFx not asserted for trigger event coinciding with last DMAMUX request	N	N
	2.4.3	OFx not asserted when writing into DMAMUX_RGCFR register	N	N
	2.4.4	Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event	A	A
FMC	2.5.1	Dummy read cycles inserted when reading synchronous memories	N	N
	2.5.2	Wrong data read from a busy NAND memory	A	A
	2.5.3	Unsupported read access with unaligned address	P	P
OCTOSPI	2.6.1	Indirect read and auto-polling transfers without address phase not starting	A	A
	2.6.2	Maxtran period not respected in specific condition	P	P
	2.6.3	Octal DDR indirect read data corrupted if last two bytes are read at a specific condition	A	A
	2.6.4	Spurious interrupt in AND-match polling mode with full data masking	A	A
	2.6.5	Hybrid wrap data transfer corruption upon an internal event	A	A
	2.6.6	Hybrid wrap registers not functional	A	A
	2.6.7	Odd address alignment and odd byte number not supported at specific conditions	A	A
	2.6.8	Memory-mapped write error response when DQS output is disabled	P	P
	2.6.9	Byte possibly dropped during an SDR read in clock mode 3 when a transfer gets automatically split	A	A
	2.6.10	Single, dual and quad modes not functional with DQS input enabled	N	N
	2.6.11	Additional bytes read in indirect mode with DQS input enabled when data length is too short	A	A

Function	Section	Limitation	Status	
			Rev. Z	Rev. X
OCTOSPI	2.6.12	Data not sampled correctly on reads without DQS and with less than two cycles before the data phase	A	A
	2.6.13	Deadlock can occur under certain conditions	A	A
	2.6.14	OCTOSPI DDR mode not supported with DQS disabled	N	N
ADC	2.7.1	New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0	A	A
	2.7.2	Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0	A	A
	2.7.3	Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode	A	A
	2.7.4	ADC_AWDy_OUT reset by non-guarded channels	A	A
DAC	2.8.1	Invalid DAC channel analog output if the DAC channel MODE bitfield is programmed before DAC initialization	A	A
	2.8.2	DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge	N	N
VREFBUF	2.9.1	Overshoot on VREFBUF output	A	A
	2.9.2	VREFBUF Hold mode cannot be used	N	N
PSSI	2.10.1	Bus error if FIFO overrun occurs during master access to the FIFO	A	A
TIM	2.11.1	One-pulse mode trigger not detected in master-slave reset + trigger configuration	P	P
	2.11.2	Consecutive compare event missed in specific conditions	N	N
	2.11.3	Output compare clear not working with external counter reset	P	P
LPTIM	2.12.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	A	A
	2.12.2	Device may remain stuck in LPTIM interrupt when clearing event flag	P	P
RTC and TAMP	2.13.1	Calendar initialization may fail in case of consecutive INIT mode entry	A	A
	2.13.2	Alarm flag may be repeatedly set when the core is stopped in debug	N	N
	2.13.3	A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF	N	N
	2.13.4	REFCKON write protection associated to INIT KEY instead of CAL KEY	A	A
	2.13.5	Tamper flag not set on LSE failure detection	N	N
I2C	2.14.1	Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period	P	P
	2.14.2	Spurious bus error detection in master mode	A	A
	2.14.3	Spurious master transfer upon own slave address match	P	P
	2.14.4	OVR flag not set in underrun condition	N	N
	2.14.5	Transmission stalled after first byte transfer	A	A
USART/UART	2.15.1	Anticipated end-of-transmission signaling in SPI slave mode	A	A
	2.15.2	Data corruption due to noisy receive line	N	N
	2.15.3	DMA stream locked when transferring data to/from USART/UART	A	A
SPI2S	2.16.1	Master data transfer stall at system clock much faster than SCK	A	A
	2.16.2	Corrupted CRC return at non-zero UDRDET setting	P	P
	2.16.3	TXP interrupt occurring while SPI disabled	A	A

Function	Section	Limitation	Status	
			Rev. Z	Rev. X
SPI2S	2.16.4	Possible corruption of last-received data depending on CRCSIZE setting	A	A
FDCAN	2.17.1	Desynchronization under specific condition with edge filtering enabled	A	A
	2.17.2	Tx FIFO messages inverted under specific buffer usage and priority setting	A	A
	2.17.3	DAR mode transmission failure due to lost arbitration	A	A

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 revision r1p2 is available from http://infocenter.arm.com.

2.1.1 CPUID register returns r1p1 instead of r1p2

Description

The devices are based on Arm® Cortex®-M7 core revision r1p2.
However, when reading the CPUID register at address 0xE000 ED00, r1p1 is returned instead of r1p2.

Workaround

None.

2.2 System

2.2.1 Performing a system reset during Flash memory program or erase operation is not supported

Description

The microcontroller may be stalled if a system reset occurs during a Flash memory program or erase operation. A power-on reset must be performed to restart the system.

Workaround

None.

2.2.2 LSE CSS parasitic detection even when disabled

Description

The LSECSSD flag in RCC_BDCR register can be spuriously set in case of ESD stress when the device is in V_{BAT} mode, even if the CSS on LSE is disabled. The LSE clock is no longer propagated to the RTC and the system, even if the LSE oscillates, as long as the LSECSSD flag is set. LSECSSD can be cleared only by a Backup domain reset.
During ST functional ESD tests, the failure was observed by stressing PC13, PC14, VBAT, PE5, PE6 with -0.2 kV / +0.5 kV or +0.7 kV. No failure is detected when both V_{DD} and V_{BAT} are present. The sensitivity observed on these five pins can be quantified through IEC1000-4-2 (ESD immunity) standard, with severity estimated between 1 (low immunity) and 2 (medium immunity) according to the same standard.

Workaround

Robustness can be improved for the pins other than VBAT by inserting, where possible, serial resistors up to a value of 1 kΩ close to the microcontroller.

2.2.3 Secure firmware install (SFI) is not supported

Description

The SFI (secure firmware install) is not supported.

Workaround

None.

2.3 BDMA

2.3.1 BDMA disable failure and error flag omission upon simultaneous transfer error and global flag clear

Description

Upon a data transfer error in a BDMA channel x, both the specific TEIFx and the global GIFx flags are raised and the channel x is normally automatically disabled. However, if in the same clock cycle the software clears the GIFx flag (by setting the CGIFx bit of the BDMA_IFCR register), the automatic channel disable fails and the TEIFx flag is not raised.

This issue does not occur with ST's HAL software that does not use and clear the GIFx flag when the channel is active.

Workaround

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

2.4 DMAMUX

2.4.1 SOFx not asserted when writing into DMAMUX_CCFR register

Description

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

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

Workaround

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

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

Description

In the DMAMUX request generator, a trigger event detected in a critical instant of the last-generated DMAMUX request being served by the DMA controller does not assert the corresponding trigger overrun flag OFx. The critical instant is the clock cycle at the very end of the trigger overrun condition.

Additionally, upon the following trigger event, one single DMA request is issued by the DMAMUX request generator, regardless of the programmed number of DMA requests to generate.

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

Workaround

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

2.4.3 OFx not asserted when writing into DMAMUX_RGCFR register

Description

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

Workaround

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

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

Description

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

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

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

Workaround

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

2.5 FMC

2.5.1 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.

2.5.2 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.

2.5.3 Unsupported read access with unaligned address

Description

Read access with unaligned address, such as a half-word read access starting at odd address, is not supported.

Workaround

Compile the software that accesses the fmc region with a compiler option that ensures data alignment, such as *–no_unaligned_access*.

2.6 OCTOSPI

2.6.1 Indirect read and auto-polling transfers without address phase not starting

Description

Indirect read and auto-polling transfers, configured through the CCR register to contain command and SDR or DDR octal data phases but no address phase, do not start.

Workaround

Configure the transfer to contain address phase and no command phase, then send the command through the address register.

2.6.2 Maxtran period not respected in specific condition

Description

Under the following condition:

- arbitration activated
- memory-mapped write initiated on one OCTOSPI instance, with a data byte number corresponding to less than two OCTOSPI clock cycles in the data phase
- another OCTOSPI instance requests the I/O port,

the I/O port is not granted even if the Maxtran period expired, unless another mechanism finishes the transaction of the first OCTOSPI instance, such as timeout, new memory request, or the arrival of additional bytes to write.

For example, in octal DDR, the minimum byte number for two clock cycles in the data phase is four. A memory-mapped write of less than four bytes by the OCTOSPI1 instance that holds the I/O port prevents the OCTOSPI2 instance to take it over when requested.

Workaround

Activate the timeout feature to trigger arbitration and select a medium timeout value. A too small value would lead to excessive chip select activity and increase power consumption, and a too big value would lead to excessive arbitration delay and inappropriate system latency.

2.6.3 Octal DDR indirect read data corrupted if last two bytes are read at a specific condition

Description

Indirect read from an octal DDR memory may lead to data corruption upon the following condition:

- Number of bytes to read, defined in OCTOSPI_DLR register, is a multiple of 32 plus two, for example 34, 66, 98, and so on.
- The last two bytes are read with different requests.
- The second-last request read size is different from one byte.

Workaround

Apply one of the following measures:

- Read the last two bytes of a transfer with the same request.
- Read the last two bytes each with transfer size of one byte.

2.6.4 Spurious interrupt in AND-match polling mode with full data masking

Description

In AND-match polling mode with the MASK[31:0] bitfield set to 0x0000 0000 (all bits masked), a spurious interrupt may occur.

Workaround

Avoid setting the MASK[31:0] bitfield to 0x0000 0000.

2.6.5 Hybrid wrap data transfer corruption upon an internal event

Description

An internal event pertaining to TIMEOUT[15:0], CSBOUND[4:0], MAXTRAN[7:0], or REFRESH[31:0] bitfields may disturb any ongoing hybrid wrap transaction and result in corruption of the remaining data to transfer.

Workaround

Manage the TIMEOUT[15:0], CSBOUND[4:0], MAXTRAN[7:0], and REFRESH[31:0] bitfields such as to avoid any related internal event during hybrid wrap transactions.

2.6.6 Hybrid wrap registers not functional

Description

OCTOSPI_WPABR and OCTOSPI_WPTCR registers are not functional. As a consequence, external memory devices that require the setting of OCTOSPI_WPABR and OCTOSPI_WPTCR registers for the hybrid wrap because it is different from the settings of OCTOSPI_ABR and OCTOSPI_TCR registers used for the read, are not supported.

Note: Most memory devices allow the same settings for the hybrid wrap and the read.

Workaround

Only use memory devices allowing the same settings for the hybrid wrap and the read.

2.6.7 Odd address alignment and odd byte number not supported at specific conditions

Description

Odd address alignment and odd transaction byte number is not supported for some combinations of memory access mode, access type, and other settings. The following table summarizes the supported combinations, and provides information on consequences of accessing an illegal address and/or of setting an illegal number of bytes in a transaction.

Table 4. Summary of supported combinations

Memory access mode / other settings[1]	Access type[2]	Address allowed	Consequence of illegal address access[3]	Byte number allowed	Consequence of illegal byte number[3]
Single-SPI, Dual-SPI, Quad-SPI, RAM / DQM = 0 or Octo-SPI / SDR mode	ind read	any	N/A	any	N/A
	mm read	any	N/A	any	N/A
	ind write	any	N/A	any	N/A
	mm write	any	N/A	any	N/A
Single-SPI, Dual-SPI, Quad-SPI, RAM / DQM = 1 or Octo-SPI, RAM / DDR mode, no RDS, no WDM	ind read	even	ADDR[0] cleared	even	DLR[0] cleared
	mm read	any	N/A	any	N/A
	ind write	even	ADDR[0] cleared	even	DLR[0] cleared
	mm write	even	slave error	even	last byte lost
Octo-SPI, RAM / DDR mode, with RDS or WDM or HyperBus™	ind read	even	ADDR[0] cleared	even	DLR[0] cleared
	mm read	any	N/A	any	N/A
	ind write	any	N/A	any	N/A
	mm write	any	N/A	any	N/A

1. *"RDS" = read data strobe, "WDM" = write data mask*
2. *"ind read" = indirect read, "mm read" = memory-mapped read, "ind write" = indirect write, "mm write" = memory-mapped write*
3. *"N/A" = not applicable*

Workaround

Avoid illegal address accesses and illegal byte numbers in transactions.

2.6.8 Memory-mapped write error response when DQS output is disabled

Description

If the DQSE control bit in OCTOSPI_WCCR is set to 0 for memories without DQS pin, it results in an error response for every memory-mapped write request.

Workaround

When doing memory-mapped writes, the DQSE bit in OCTOSPI_WCCR must be set to 1 even for memories which have no DQS pin.

Limitation of this workaround: if the DQS output is asserted on memory-mapped writes while the AXI bus transfer has some byte-enable bits deasserted, the bytes which should be masked get written to the memory.

2.6.9 Byte possibly dropped during an SDR read in clock mode 3 when a transfer gets automatically split

Description

When reading a continuous stream of data from sequential addresses in a serial memory, OCTOSPI can interrupt the transfer and automatically restart it at the next address when the CSBOUND, REFRESH, TIMEOUT or MAX-TRAN features are employed. Thus, a single continuous transfer can effectively be split into multiple smaller transfers.

When OCTOSPI is configured to use clock mode 3 (CKMODE bit in OCTOSPI_DCR1 is set to 1) and a continuous stream of data is read in SDR mode (CKMODE bit in OCTOSPI_DCR1 is set to 0), the last byte sent by the memory before an automatic split might get dropped, thus causing all the subsequent bytes to be seen one address earlier.

Workaround

Use clock mode 0 (CKMODE bit in OCTOSPI_DCR1 is set to 0) when in SDR mode.

2.6.10 Single, dual and quad modes not functional with DQS input enabled

Description

Data read from memory in single, dual or quad mode with the DQS input enabled (DQSE control bit in OCTOSPI_CCR is set to 1) can be corrupted. Only octal-data mode (DMODE bit in OCTOSPI_CCR is set to 100) is functional with the DQS input enabled.

Workaround

None.

2.6.11 Additional bytes read in indirect mode with DQS input enabled when data length is too short

Description

Extra bytes reception may appear when below two conditions are met at the same time:

- Data read in indirect-read mode with DQS enabled (DQSE bit in OCTOSPI_CCR set to 1)
- The number of cycles for data read phase is less than the sum of the number of cycles required for (command + address + alternate-byte + dummy) phases.

Workaround

- Avoid programming transfers with data phase shorter than (command + address + alternate-byte + dummy) phases
- Perform an abort just after reading all the data required bytes from OCTOSPI_DR register.

2.6.12 Data not sampled correctly on reads without DQS and with less than two cycles before the data phase

Description

A command is composed of five phases:

- Command
- Address
- Alternate byte
- Dummy (latency) cycles
- Data

Data are not sampled correctly if all the following conditions are true:

- Fewer than two cycles are required by the first four phases (command, address, alternate or dummy)
- DQS is disabled (DQSE=0)

- Data phase is enabled
- Data are read in Indirect or Memory-mapped mode

Workaround

Ensure that there are at least two cycles before the data phase using one of the following methods :

- Send one byte of address in quad-SDR mode (ADMODE=011, ADSIZE=00, ADDDTR=0)
- Send two bytes of address in octal-SDR mode (ADMODE=100, ADSIZE=01, ADDDTR=0)
- Send four bytes of address in octal-DTR mode (ADMODE=100, ADSIZE=11, ADDDTR=1)

2.6.13 Deadlock can occur under certain conditions

Description

A deadlock can occur when the following conditions occur simultaneously :

- The product communicates in Multiplexed mode with one of the following:
  - an external memory
  - an external combo featuring two memories, such as two OCTOSPI blocks connected to the external memory through an I/O manager; directly or through a high speed interface.
- At some point in the datapath, both OCTOSPI data bus interfaces share the same layer of the internal interconnect matrix:
  - Both data bus interfaces are AHB and share the same layer (Indirect mode use case).
  - Both data bus interfaces are AHB or AXI and share the same layer (Memory-mapped mode use case).

Then deadlock can happen when the two following conditions occur at the same time:

- The OCTOSPI interface which currently owns the external bus (for example OCTOSPI1) is waiting for a transfer to occur with the external memory to complete its transfer on AHB/AXI bus side.
- A data transfer request on AHB/AXI bus arrives to the other OCTOSPI interface (for example OCTOSPI2).

This leads to an ownership conflict where:

- OCTOSPI2 cannot get ownership of the external bus which is currently in use by OCTOSPI1.
- OCTOSPI1 cannot get ownership of the AHB/AXI bus which is currently in use by OCTOSPI2.

Workaround

There are two possible workarounds:

- Workaround 1:

  If any of the communication regulation features are set (MAXTRAN, REFRESH, CSBOUND, TIMEOUT) then OCTOSPI1 splits its transfer at some point in time, releasing the bus. OCTOSPI2 can then process its data, and when OCTOSPI1 gets ownership back again, it will resume its transfer thanks to its embedded capability to restart at the address following the last address accessed. In this case, the deadlock is resolved.

  Limitation:

  The automatic resume of the transfer will not work with certain Flash memories in WRITE direction only, the memory devices in question require an extra "write enable" command before resuming a write transfer. This "write enable" command is not generated by the OCTOSPI.
- Workaround 2:

  The application must ensure that it has sufficient room left in the OCTOSPI internal FIFO for each and every transfer before launching it. In such a case, the AHB/AXI activity no longer depends on what happens on external bus side, and deadlock condition is avoided. This workaround has no limitation.

2.6.14 OCTOSPI DDR mode not supported with DQS disabled

Description

The Octo-SPI interface does not support DDR mode when DQS is disabled. This is true for all non-Hyperbus protocols and all data modes: octal, dual-quad, quad, dual and single.

Workaround

None.

2.7 ADC

2.7.1 New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0

Description

Once an injected conversion sequence is complete, the queue is consumed and the context changes according to the new ADC_JSQR parameters stored in the queue. This new context is applied for the next injected sequence of conversions.

However, the programming of the new context in ADC_JSQR (change of injected trigger selection and/or trigger polarity) may launch the execution of this context without waiting for the trigger if:

- the queue of context is enabled (JQDIS cleared to 0 in ADC_CFGR), and
- the queue is never empty (JQM cleared to 0 in ADC_CFGR), and
- the injected conversion sequence is complete and no conversion from previous context is ongoing

Workaround

Apply one of the following measures:

- Ignore the first conversion.
- Use a queue of context with JQM = 1.
- Use a queue of context with JQM = 0, only change the conversion sequence but never the trigger selection and the polarity.

2.7.2 Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0

Description

When an injected conversion sequence is complete and the queue is consumed, writing a new context in ADC_JSQR just after the completion of the previous context and with a length longer that the previous context, may cause both contexts to fail. The two contexts are considered as one single context. As an example, if the first context contains element 1 and the second context elements 2 and 3, the first context is consumed followed by elements 2 and 3 and element 1 is not executed.

This issue may happen if:

- the queue of context is enabled (JQDIS cleared to 0 in ADC_CFGR), and
- the queue is never empty (JQM cleared to 0 in ADC_CFGR), and
- the length of the new context is longer than the previous one

Workaround

If possible, synchronize the writing of the new context with the reception of the new trigger.

2.7.3 Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode

Description

In Dual ADC mode, an unexpected regular conversion may start at the end of the second injected conversion without a regular trigger being received, if the second injected conversion starts exactly at the same time than the end of the first injected conversion. This issue may happen in the following conditions:

- two consecutive injected conversions performed in Interleaved simultaneous mode (DUAL[4:0] of ADC_CCR = 0b00011), or
- two consecutive injected conversions from master or slave ADC performed in Interleaved mode (DUAL[4:0]of ADC_CCR = 0b00111)

Workaround

- In Interleaved simultaneous injected mode: make sure the time between two injected conversion triggers is longer than the injected conversion time.
- In Interleaved only mode: perform injected conversions from one single ADC (master or slave), making sure the time between two injected triggers is longer than the injected conversion time.

2.7.4 ADC_AWDy_OUT reset by non-guarded channels

Description

ADC_AWDy_OUT is set when a guarded conversion of a regular or injected channel is outside the programmed thresholds. It is reset after the end of the next guarded conversion that is inside the programmed thresholds.

However, the ADC_AWDy_OUT signal is also reset at the end of conversion of non-guarded channels, both regular and injected.

Workaround

When ADC_AWDy_OUT is enabled, it is recommended to use only the ADC channels that are guarded by a watchdog.

If ADC_AWDy_OUT is used with ADC channels that are not guarded by a watchdog, take only ADC_AWDy_OUT rising edge into account.

2.8 DAC

2.8.1 Invalid DAC channel analog output if the DAC channel MODE bitfield is programmed before DAC initialization

Description

When the DAC operates in Normal mode and the DAC enable bit is cleared, writing a value different from 000 to the DAC channel MODE bitfield of the DAC_MCR register before performing data initialization causes the corresponding DAC channel analog output to be invalid.

Workaround

Apply the following sequence:

1. Perform one write access to any data register.
2. Program the MODE bitfield of the DAC_MCR register.

2.8.2 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.

2.9 VREFBUF

2.9.1 Overshoot on VREFBUF output

Description

An overshoot might occur on VREFBUF output if VREF+ pin has residual voltage when VREFBUF is enabled (ENVR is set in VREFBUF_CSR register).

Workaround

Let the voltage on the VREF+ pin drop to 1 V under the target V_{REFBUF_OUT}. This can be achieved by switching VREFBUF buffer off (ENVR is cleared and HIZ is cleared in VREFBUF_CSR register) allowing sufficient time to discharge the capacitor on the VREF+ pin through the VREFBUF pull-down resistor.

2.9.2 VREFBUF Hold mode cannot be used

Description

VREFBUF can be configured to operate in Hold mode to reduce current consumption.

When VREFBUF enters Hold mode (by setting both HIZ and ENVR bits of the VREFBUF_CSR register), the VREF+ I/O transits to high impedance mode. If not discharged externally, the capacitor on the VREF+ pin keeps its charge and voltage. Exiting VREFBUF Hold mode (by clearing the HIZ bit) in this condition might lead to a voltage overshoot on the VREF+ output.

Workaround

None.

2.10 PSSI

2.10.1 Bus error if FIFO overrun occurs during master access to the FIFO

Description

If a PSSI FIFO overrun occurs while a master is accessing the FIFO, a bus error is reported to the master.

Workaround

Use the PSSI with the DMA and manage PSSI FIFO overrun through the DMA transfer error interrupt.

2.11 TIM

2.11.1 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.

2.11.2 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.

2.11.3 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).

2.12 LPTIM

2.12.1 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.

2.12.2 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.

2.13 RTC and TAMP

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

Description

If the INIT bit of the RTC_ICSR register is set between one and two 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 occurring 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.

2.13.2 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.

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

Description

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

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

Workaround

None.

2.13.4 REFCKON write protection associated to INIT KEY instead of CAL KEY

Description

The write protection of the REFCKON bit is unlocked if the key sequence is written in RTC_WPR with the privilege and security rights set by the INITPRIV and INITSEC bits, instead of being set by the CALPRIV and CALSEC bits.

Workaround

Unlock the INIT KEY before writing REFCKON.

2.13.5 Tamper flag not set on LSE failure detection

Description

With the timestamp on tamper event enabled (the TAMPTS bit of the RTC_CR register set), the LSE failure detection (LSE clock stopped) event connected to the internal tamper 3 fails to raise the ITAMP3F and ITAMP3MF flags, although it duly erases or blocks (depending on the internal tamper 3 configuration) the backup registers and other device secrets, and the RTC and TAMP peripherals resume normally upon the LSE restart.

Note: As expected in this particular case, the TSF and TSMF flags remain low as long as LSE is stopped as they require running RTCCLK clock to operate.

Workaround

None.

2.14 I2C

2.14.1 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.

2.14.2 Spurious bus error detection in master mode

Description

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

2.14.3 Spurious master transfer upon own slave address match

Description

When the device is configured to operate at the same time as master and slave (in a multi- master I^2C-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.

2.14.4 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.

2.14.5 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/UART

2.15.1 Anticipated end-of-transmission signaling in SPI slave mode

Description

In SPI slave mode, at low USART baud rate with respect to the USART kernel and APB clock frequencies, the *transmission complete* flag TC of the USARTx_ISR register may unduly be set before the last bit is shifted on the transmit line.

This leads to data corruption if, based on this anticipated end-of-transmission signaling, the application disables the peripheral before the last bit is transmitted.

Workaround

Upon the TC flag rise, wait until the clock line remains idle for more than the half of the communication clock cycle. Then only consider the transmission as ended.

2.15.2 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.

2.15.3 DMA stream locked when transferring data to/from USART/UART

Description

When a USART/UART is issuing a DMA request to transfer data, if a concurrent transfer occurs, the requested transfer may not be served and the DMA stream may stay locked.

Workaround

Use the alternative peripheral DMA channel protocol by setting bit 20 of the DMA_SxCR register.

This bit is reserved in the documentation and must be used only on the stream that manages data transfers for USART/UART peripherals.

2.16 SPI2S

2.16.1 Master data transfer stall at system clock much faster than SCK

Description

With the system clock (spi_pclk) substantially faster than SCK (spi_ker_ck divided by a prescaler), SPI master data transfer can stall upon setting the CSTART bit within one SCK cycle after the EOT event (EOT flag raise) signaling the end of the previous transfer.

Workaround

Apply one of the following measures:

- Disable then enable SPI after each EOT event.
- Upon EOT event, wait for at least one SCK cycle before setting CSTART.
- Prevent EOT events from occurring, by setting transfer size to undefined (TSIZE = 0) and by triggering transmission exclusively by TXFIFO writes.

2.16.2 Corrupted CRC return at non-zero UDRDET setting

Description

With non-zero setting of UDRDET[1:0] bitfield, the SPI slave can transmit the first bit of CRC pattern corrupted, coming wrongly from the UDRCFG register instead of SPI_TXCRC. All other CRC bits come from the SPI_TXCRC register, as expected.

Workaround

Keep TXFIFO non-empty at the end of transfer.

2.16.3 TXP interrupt occurring while SPI disabled

Description

SPI2S peripheral is set to its default state when disabled (SPE = 0). This flushes the FIFO buffers and resets their occupancy flags. TXP and TXC flags become set (the latter if the TSIZE field contains zero value), triggering interrupt if enabled with TXPIE or EOTIE bit, respectively. The resulting interrupt service can be spurious if it tries to write data into TXFIFO to clear the TXP and TXC flags, while both FIFO buffers are inaccessible (as the peripheral is disabled).

Workaround

Keep TXP and TXC (the latter if the TSIZE field contains zero value) interrupt disabled whenever the SPI2S peripheral is disabled.

2.16.4 Possible corruption of last-received data depending on CRCSIZE setting

Description

With the CRC calculation disabled (CRCEN = 0), the transfer size bitfield set to a value greater than zero (TSIZE[15:0] > 0), and the length of CRC frame set to less than 8 bits (CRCSIZE[4:0] < 00111), the last data received in the RxFIFO may be corrupted.

Workaround

Keep the CRCSIZE[4:0] bitfield at its default setting (00111) during the data reception if CRCEN = 0 and TSIZE[15:0] > 0.

2.17 FDCAN

2.17.1 Desynchronization under specific condition with edge filtering enabled

Description

FDCAN may desynchronize and incorrectly receive the first bit of the frame if:

- the edge filtering is enabled (the EFBI bit of the FDCAN_CCCR register is set), and
- the end of the integration phase coincides with a falling edge detected on the FDCAN_Rx input pin

If this occurs, the CRC detects that the first bit of the received frame is incorrect, flags the received frame as faulty and responds with an error frame.

Note: This issue does not affect the reception of standard frames.

Workaround

Disable edge filtering or wait for frame retransmission.

2.17.2 Tx FIFO messages inverted under specific buffer usage and priority setting

Description

Two consecutive messages from the Tx FIFO may be inverted in the transmit sequence if:

- FDCAN uses both a dedicated Tx buffer and a Tx FIFO (the TFQM bit of the FDCAN_TXBC register is cleared), and
- the messages contained in the Tx buffer have a higher internal CAN priority than the messages in the Tx FIFO.

Workaround

Apply one of the following measures:

- Ensure that only one Tx FIFO element is pending for transmission at any time:

  The Tx FIFO elements may be filled at any time with messages to be transmitted, but their transmission requests are handled separately. Each time a Tx FIFO transmission has completed and the Tx FIFO gets empty (TFE bit of FDACN_IR set to 1) the next Tx FIFO element is requested.
- Use only a Tx FIFO:

  Send both messages from a Tx FIFO, including the message with the higher priority. This message has to wait until the preceding messages in the Tx FIFO have been sent.
- Use two dedicated Tx buffers (for example, use Tx buffer 4 and 5 instead of the Tx FIFO). The following pseudo-code replaces the function in charge of filling the Tx FIFO:

```
Write message to Tx Buffer 4
Transmit Loop:
    Request Tx Buffer 4 - write AR4 bit in FDCAN_TXBAR
    Write message to Tx Buffer 5
    Wait until transmission of Tx Buffer 4 complete (IR bit in FDCAN_IR),
    read TO4 bit in FDCAN_TXBTO
    Request Tx Buffer 5 - write AR5 bit of FDCAN_TXBAR
    Write message to Tx Buffer 4
    Wait until transmission of Tx Buffer 5 complete (IR bit in FDCAN_IR),
    read TO5 bit in FDCAN_TXBTO
```

2.17.3 DAR mode transmission failure due to lost arbitration

Description

In DAR mode, the transmission may fail due to lost arbitration at the first two identifier bits.

Workaround

Upon failure, clear the corresponding Tx buffer transmission request bit TRPx of the FDCAN_TXBRP register and set the corresponding cancellation finished bit CFx of the FDCAN_TXBCF register, then restart the transmission.

Revision history

Table 5. Document revision history

Date	Version	Changes
29-Jan-2020	7	First public release.
29-Jun-2020	8	Added silicon revision 'X'. Removed System limitation "Unstable LSI when it clocks RTC or CSS on LSE". Added ADC limitations: Section 2.7.1 New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0, Section 2.7.2 Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0, Section 2.7.3 Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode and Section 2.7.4 ADC_AWDy_OUT reset by non-guarded channels. Added DAC limitation: Section 2.8.2 DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge. Added FMC limitation: Section 2.5.3 Unsupported read access with unaligned address. Added TIM limitations: Section 2.11.2 Consecutive compare event missed in specific conditions and Section 2.11.3 Output compare clear not working with external counter reset. Added RTC limitations: Section 2.13.3 A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF, Section 2.13.4 REFCKON write protection associated to INIT KEY instead of CAL KEY and Section 2.13.5 Tamper flag not set on LSE failure detection. Added I2C limitations: Section 2.14.4 OVR flag not set in underrun condition and Section 2.14.5 Transmission stalled after first byte transfer Added USART limitations: Section 2.15.1 Anticipated end-of-transmission signaling in SPI slave mode, Section 2.15.2 Data corruption due to noisy receive line and Section 2.15.3 DMA stream locked when transferring data to/from USART/UART. Added SPI limitation: Section 2.16.4 Possible corruption of last-received data depending on CRCSIZE setting. Added FDCAN limitations: Section 2.17.1 Desynchronization under specific condition with edge filtering enabled, Section 2.17.2 Tx FIFO messages inverted under specific buffer usage and priority setting, and Section 2.17.3 DAR mode transmission failure due to lost arbitration.
16-Oct-2020	9	Updated Section 2.3.1 BDMA disable failure and error flag omission upon simultaneous transfer error and global flag clear description. Updated Section 2.5.3 Unsupported read access with unaligned address workaround. Added the following OCTOSPI errata: Section 2.6.8 Memory-mapped write error response when DQS output is disabled, Section 2.6.9 Byte possibly dropped during an SDR read in clock mode 3 when a transfer gets automatically split, Section 2.6.10 Single, dual and quad modes not functional with DQS input enabled and Section 2.6.12 Data not sampled correctly on reads without DQS and with less than two cycles before the data phase. Removed *START bit is cleared upon setting ADDRCF, not upon address match* I2C erratum. Removed *Receiver timeout counter wrong start in two-stop-bit configuration USART* erratum and added .
25-Jun-2021	10	Added the following system limitations: LSE CSS parasitic detection even when disabled and Secure firmware install (SFI) is not supported. Removed DMA limitation *DMA stream locked when transferring data to/from USART/UART* since it is already present in USART section. Added the following OCTOSPI limitation: Deadlock can occur under certain conditions. Updated Overshoot on VREFBUF output. Replaced USART by USART/UART in USART limitations

Contents