STM32H723xx STM32H730xx STM32H733xx STM32H725xx STM32H735xx

Errata sheet

STM32H72xx/73xx device errata

Applicability

This document applies to the part numbers of STM32H72xx/73xx 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 RM0468.

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
STM32H723xx	STM32H723ZG, STM32H723VG, STM32H723ZE, STM32H723VE
STM32H730xx	STM32H730AB, STM32H730IB, STM32H730VB, STM32H730ZB
STM32H733xx	STM32H733ZG, STM32H733VG
STM32H725xx	STM32H725ZG, STM32H725VG, STM32H725RG, STM32H725IG, STM32H725AG
STM32H735xx	STM32H735ZG, STM32H735VG, STM32H735RG, STM32H735IG, STM32H735AG

Table 2. **Device variants**

Reference	Silicon revision codes	
	Device marking[1]	REV_ID[2]
STM32H723xx/STM32H730xx/733xx/725xx/735xx	A	0x1000
STM32H723xx/STM32H730xx/733xx/725xx/735xx	Z	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 DBGMCU_IDC register.*

1 Summary of device errata

The following table gives a quick reference to the STM32H72xx/73xx device limitations and their status:

A = workaround available

N = no workaround available

P = partial workaround available

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
System	2.1.1	A tamper event does not erase the backup RAM when the backup RAM clock is disabled	A	A
	2.1.2	A tamper event does not erase the OTFDEC keys when the backup RAM clock is disabled	-	A
MDMA	2.2.1	Non-flagged MDMA write attempts to reserved area	A	A
BDMA	2.3.1	BDMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	A	A
DMA	2.4.1	DMA stream locked when transferring data to/from USART/UART	A	A
DMAMUX	2.5.1	SOFx not asserted when writing into DMAMUX_CCFR register	N	N
	2.5.2	OFx not asserted for trigger event coinciding with last DMAMUX request	N	N
	2.5.3	OFx not asserted when writing into DMAMUX_RGCFR register	N	N
	2.5.4	Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event	A	A
FMC	2.6.1	Dummy read cycles inserted when reading synchronous memories	N	N
	2.6.2	Wrong data read from a busy NAND memory	A	A
	2.6.3	Unsupported read access with unaligned address	P	P
OCTOSPI	2.7.1	Spurious interrupt in AND-match polling mode with full data masking	A	A
	2.7.2	Hybrid wrap data transfer corruption upon an internal event	A	A
	2.7.3	Hybrid wrap registers not functional	A	A
	2.7.4	Odd address alignment and odd byte number not supported at specific conditions	A	A
	2.7.5	Data not sampled correctly on reads without DQS and with less than two cycles before the data phase	A	A
	2.7.6	Memory-mapped write error response when DQS output is disabled	P	P
	2.7.7	Byte possibly dropped during an SDR read in clock mode 3 when a transfer gets automatically split	A	A
	2.7.8	Single, dual and quad modes not functional with DQS input enabled	N	N
	2.7.9	Additional bytes read in indirect mode with DQS input enabled when data length is too short	A	A
ADC	2.8.1	ADC3 conversion data corrupted when switching input channels	N	-
	2.8.2	ADC3 performance decreased at low frequency	A	-

Function	Section	Limitation	Status	
			Rev. A	Rev. Z
ADC	2.8.3	ADC3 built-in offset calibration not functional	N	-
	2.8.4	ADC3 injected channel conversion while regular conversion is running may provide corrupted data	P	P
DAC	2.9.1	Invalid DAC channel analog output if the DAC channel MODE bitfield is programmed before DAC initialization	A	A
	2.9.2	DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge	N	N
VREFBUF	2.10.1	Overshoot on VREFBUF output	A	A
	2.10.2	VREFBUF Hold mode cannot be used	N	N
OTFDEC	2.11.1	OTFDEC encryption key not erased upon a tamper event	N	-
TIM	2.12.1	One-pulse mode trigger not detected in master-slave reset + trigger configuration	P	P
	2.12.2	Consecutive compare event missed in specific conditions	N	N
	2.12.3	Output compare clear not working with external counter reset	P	P
LPTIM	2.13.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	A	A
	2.13.2	Device may remain stuck in LPTIM interrupt when clearing event flag	P	P
WWDG	2.14.1	WWDG not functional when VDD is lower than 2.7 V and VOS0 or VOS1 voltage level is selected	N	N
RTC and TAMP	2.15.1	RTC interrupt can be masked by another RTC interrupt	A	A
	2.15.2	Calendar initialization may fail in case of consecutive INIT mode entry	A	A
	2.15.3	Alarm flag may be repeatedly set when the core is stopped in debug	N	N
I2C	2.16.1	Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period	P	P
	2.16.2	Spurious bus error detection in master mode	A	A
	2.16.3	OVR flag not set in underrun condition	N	N
	2.16.4	Transmission stalled after first byte transfer	A	A
USART	2.17.1	Anticipated end-of-transmission signaling in SPI slave mode	A	A
	2.17.2	Data corruption due to noisy receive line	N	N
	2.17.3	DMA stream locked when transferring data to/from USART	A	A
SPI	2.18.1	Master data transfer stall at system clock much faster than SCK	A	A
	2.18.2	Corrupted CRC return at non-zero UDRDET setting	P	P
	2.18.3	TXP interrupt occurring while SPI disabled	A	A
FDCAN	2.19.1	Desynchronization under specific condition with edge filtering enabled	A	A
	2.19.2	Tx FIFO messages inverted under specific buffer usage and priority setting	A	A
	2.19.3	DAR mode transmission failure due to lost arbitration	A	A
ETH	2.20.1	The MAC does not provide bus access to a higher priority request after a low priority request is serviced	N	N
	2.20.2	Rx DMA may fail to recover upon a DMA restart following a bus error, with Rx timestamping enabled	A	A
	2.20.3	The Tx DMA fails to recover correctly or corrupts TSO/USO header data on receiving a bus error response from the AHB DMA slave	N	N

Function	Section	Limitation	Status	
			Rev. A	Rev. Z
ETH	2.20.4	Incorrectly weighted round robin arbitration between Tx and Rx DMA channels to access the common host bus	A	A
	2.20.5	Incorrect L4 inverse filtering results for corrupted packets	N	N
	2.20.6	IEEE 1588 Timestamp interrupt status bits are incorrectly cleared on write access to the CSR register with similar offset address	A	A
	2.20.7	Bus error along with Start-of-Packet can corrupt the ongoing transmission of MAC generated packets	N	N
	2.20.8	Spurious receive watchdog timeout interrupt	A	A
	2.20.9	Incorrect flexible PPS output interval under specific conditions	A	A
	2.20.10	Packets dropped in RMII 10Mbps mode due to fake dribble and CRC error	A	A
	2.20.11	ARP offload function not effective	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 System

2.1.1 A tamper event does not erase the backup RAM when the backup RAM clock is disabled

Description

Upon a tamper event, the backup RAM is normally reset and its content erased. However, when the backup RAM clock is disabled (BKPRAMEN bit set to 0 in RCC_ AHB4ENR register), the backup RAM reset fails and the memory is not erased.

Workaround

Enable the backup RAM clock by setting BKPRAMEN bit to 1 in the RCC_AHB4 clock register (RCC_AHB4ENR). This can be done either during device initialization or during a tamper service routine.

2.1.2 A tamper event does not erase the OTFDEC keys when the backup RAM clock is disabled

Description

Upon a tamper event, the OTFDEC key registers (OTFDEC_RxKEYRy) are normally reset and their content erased. However, when the backup RAM clock is disabled (BKPRAMEN bit set to 0 in RCC_ AHB4ENR register), the reset of the OTFDEC keys fails and their content is not erased.

Workaround

Enable the backup RAM clock by setting BKPRAMEN bit to 1 in the RCC_AHB4 clock register (RCC_AHB4ENR). This can be done either during device initialization or during a tamper service routine.

2.2 MDMA

2.2.1 Non-flagged MDMA write attempts to reserved area

Description

The 0x0000 0000 - 0x0003 FFFF address space is linked to ITCM. The TCM_AXI_SHARED[1:0] bitfield of the FLASH_OPTSR2_CUR option byte defines areas of this address space valid for access and reserved areas. MDMA write access (through the CPU AHBS) to an address in any reserved area is expected to signal bus error exception.

However, with TCM_AXI_SHARED[1:0] bitfield set to 10, although MDMA write attempts to addresses in the 0x0003 0000 - 0x0003 FFFF reserved area are duly ignored (no data write effected), they do not signal bus error exception (no flag is raised), which corresponds to MDMA wrongly reporting *write completed*.

Workaround

Avoid accessing reserved areas.

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, but uses and clears the HTIFx, TCIFx, and TEIFx specific event flags instead.

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 DMA

2.4.1 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.

2.5 DMAMUX

2.5.1 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.

2.5.2 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.

2.5.3 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.

2.5.4 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.

2.6 FMC

2.6.1 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.

2.6.2 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.

2.6.3 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*.

2.7 OCTOSPI

2.7.1 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.

2.7.2 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.

2.7.3 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.

2.7.4 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.

2.7.5 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)
- Send two bytes of instruction in quad-DTR mode (IMODE=011, ISIZE=01, IDDTR=1)
- Send one instruction byte in octal followed by one dummy cycle.
- Send one instruction byte in octal followed by one alternate byte in octal.

2.7.6 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.

2.7.7 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.

2.7.8 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.

2.7.9 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.

2.8 ADC

2.8.1 ADC3 conversion data corrupted when switching input channels

Description

ADC3 conversion data are corrupted when switching input channels. As a result Single mode, Discontinuous mode, and Continuous mode with a sequence length of more than one conversion may lead to wrong data.

As a consequence ADC3 can only be used in Continuous conversion mode on a single channel.

Workaround

None.

2.8.2 ADC3 performance decreased at low frequency

Description

When ADC3 kernel clock frequency is lower than 5 MHz, the conversion accuracy is degraded.

Workaround

Keep ADC3 kernel clock frequency above 5 MHz. If needed, the sampling time can be increased without negative impact on the conversion.

2.8.3 ADC3 built-in offset calibration not functional

Description

ADC3 native negative offset can be corrected but not the native positive one. As a result ADC3 built-in offset calibration is not functional.

Workaround

None.

2.8.4 *ADC3 injected channel conversion while regular conversion is running may provide corrupted data*

Description

The result of a 12-bit ADC3 injected channel conversion may be corrupted when the injected channel conversion is launched while a regular channel conversion is already ongoing.

Workaround

Apply one of the following measures:

- Insert a dummy conversion at the beginning of the injected sequence. The second conversion will be correct.
- Avoid collisions between injected and regular conversions by using triggered regular conversions or by launching regular conversions only when injected conversions are complete.

2.9 DAC

2.9.1 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.

2.9.2 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.

2.10 VREFBUF

2.10.1 Overshoot on VREFBUF output

Description

An overshoot might occur on VREFBUF output if VREF+ pin has residual voltage when VREFBUF is enabled (ENVR set to 1 in VREFBUF_CSR register).

Workaround

Let the voltage on the VREF+ pin drop to a level lower than 1 V below the target V_{REFBUF_OUT}. This can be achieved by switching VREFBUF buffer off (ENVR = 0 and HIZ = 0 in VREFBUF_CSR register) during sufficient time to discharge the capacitor on the VREF+ pin through VREFBUF pull-down resistor.

2.10.2 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.

2.11 OTFDEC

2.11.1 OTFDEC encryption key not erased upon a tamper event

Description

Tamper events on external ports configured as RTC_TAMPx are expected to erase OTFDEC_RxKEYR OTFDEC encryption key registers and raise the key error interrupt flag (KEIF) of the OTFDEC_ISR register.

However, although tamper events are duly detected, they do not have any effect on OTFDEC_RxKEYR and KEIF.

Workaround

None.

2.12 TIM

2.12.1 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.

2.12.2 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.

2.12.3 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).

2.13 LPTIM

2.13.1 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.

2.13.2 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.*

2.14 WWDG

2.14.1 WWDG not functional when V_{DD} is lower than 2.7 V and VOS0 or VOS1 voltage level is selected

Description

The system window watchog (WWDG) is not functional, that is, it does not generate a correct system reset and/or the WWDG reset flag is not asserted, when V_{DD} is lower than 2.7 V and VOS0 or VOS1 voltage level is selected. There is no dependency on V_{DDLDO}.

Workaround

None.

2.15 RTC and TAMP

2.15.1 RTC interrupt can be masked by another RTC interrupt

Description

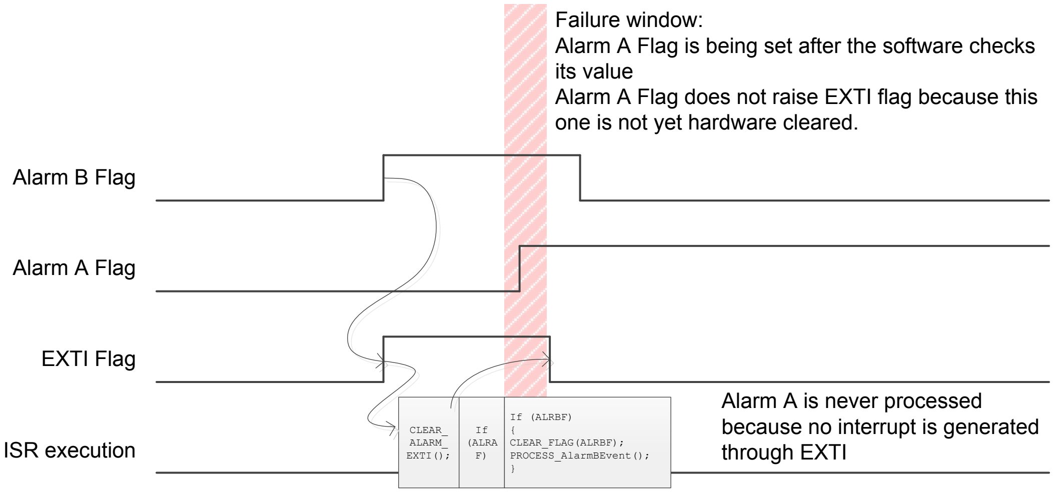
One RTC interrupt request can mask another RTC interrupt request if they share the same EXTI configurable line. For example, interrupt requests from Alarm A and Alarm B or those from tamper and timestamp events are OR-ed to the same EXTI line (refer to the *EXTI line connections* table in the *Extended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

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

Alarm interrupt service routine:

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

Figure 1. Masked RTC interrupt


Failure window:
Alarm A Flag is being set after the software checks its value
Alarm A Flag does not raise EXTI flag because this one is not yet hardware cleared.
Alarm B Flag
Alarm A Flag
EXTI Flag
ISR execution
CLEAR_ ALARM_ EXTI();
If (ALRAF)
If (ALRBF)
{
CLEAR_FLAG(ALRBF);
PROCESS_AlarmBEvent();
}
Alarm A is never processed because no interrupt is generated through EXTI

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 */
    }
}
```

2.15.2 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.

2.15.3 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.

2.16 I2C

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

Description

The I^{2}C-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^{2}C-bus SDA line when $t_{SU;DAT}$ is smaller than one I2C kernel clock (I^{2}C-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^{2}C-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.

2.16.2 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^{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.

2.16.3 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.

2.16.4 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.

2.17 USART

2.17.1 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.

2.17.2 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.

2.17.3 DMA stream locked when transferring data to/from USART

Description

When a USART 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 peripherals.

2.18 SPI

2.18.1 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.

2.18.2 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.

2.18.3 TXP interrupt occurring while SPI disabled

Description

SPI 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 SPI peripheral is disabled.

2.19 FDCAN

2.19.1 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.

2.19.2 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
```

2.19.3 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.

2.20 ETH

2.20.1 The MAC does not provide bus access to a higher priority request after a low priority request is serviced

Description

The ETH_DMAMR DMA Mode register in the MAC can be programmed to arbitrate between the DMA channels to access the system bus:

- Use a weighted round robin (WRR) algorithm for selecting between transmit or receive DMA channels by clearing DA bit
- Give higher priority to transmit or receive DMA channels by programming the TXPR bit of the ETH_DMAMR register
- Select the priority ratio of TX over RX or vice versa (as per TXPR) by programming the PR[2:0] field

For the WRR algorithm, the MAC provides bus access to a higher priority request provided it is within the priority ratio. It services a lower priority request only when higher priority requests have been serviced as per priority ratio or when there are no higher priority requests.

However, in the WRR algorithm operation, when there are requests pending from both transmit DMA and receive DMA after a lower priority request gets serviced, the MAC incorrectly selects the lower priority request, thus violating the PR ratio. The MAC continues to service all the subsequent low priority requests until there are no low priority requests, before servicing any high priority request.

This results in a delay in servicing the higher priority requests. If the high priority request is programmed for receive DMA channels (TXPR is cleared), the receive queue can overflow with a resulting loss of packets. If the high priority request is programmed for transmit DMA (TXPR is set) channels, the transmit queue can get starved in store and forward mode resulting in low throughput. Otherwise when operating in threshold mode, the transmit queue can underflow, resulting in discarding of packet by remote end. In both cases the quality of service or throughput may be affected.

Also, when priority ratio of 8:1 is programmed, the serviced request count rolls over to 0 after reaching 7 and does not reach maximum value which is 8. So, if the higher priority request is being serviced, lower priority request does not get serviced until there is no higher priority request.

These issues do not affect the functionality but impacts the performance.

Workaround

None.

2.20.2 Rx DMA may fail to recover upon a DMA restart following a bus error, with Rx timestamping enabled

Description

When the timestamping of the Rx packets is enabled, some or all of the received packets can have an Rx timestamp which is written into a descriptor upon the completion of the Rx packet/status transfer.

However, when a bus error occurs during the descriptor read (that is subsequently used as context descriptor to update the Rx timestamp), the context descriptor write is skipped by the DMA. Also, the Rx DMA does not flush the Rx timestamp stored in the intermediate buffers during the error recovery process and enters stop state. Due to this residual timestamp in the intermediate buffer remaining after the restart, the Rx DMA does not transfer any packets.

Workaround

Issue a soft reset to drop all Tx packets and Rx packets present inside the controller at the time of a bus error. After the soft reset, reconfigure the controller and re-create the descriptors.

Note: *The workaround introduces additional latency.*

2.20.3 The Tx DMA fails to recover correctly or corrupts TSO/USO header data on receiving a bus error response from the AHB DMA slave

Description

When a bus error is received from the AHB DMA slave, the controller generates an interrupt by setting the FBE bit of the ETH_DMACSR register. This stops the corresponding DMA channel by resetting the ST bit of the ETH_DMACTXCR register after recovering from the error. The software recreates the list of descriptors and restarts the DMA by setting the ST bit 0 of the ETH_DMACTXCR register without issuing the software reset to the controller.

However, the Tx DMA fails to recover or corrupts the TSO/USO header data when the TSO/USO segmentation is enabled in the Tx Descriptor and if either:

- a bus error is detected while transferring the header data from the system memory
- a bus error occurs for the intermediate beat transfer of the header data

In this case the first packet (with TSO/USO enabled after re-starts) gets corrupted after the DMA re-starts.

Workaround

Issue a soft reset to recover from this scenario. Issuing a soft reset results in loss of all Tx packets and Rx packets present inside the controller at the time of bus-error. Also, the software must reconfigure the controller and re-create the descriptors. This is an overhead which introduces additional latency.

2.20.4 Incorrectly weighted round robin arbitration between Tx and Rx DMA channels to access the common host bus

Description

The ethernet peripheral has independent transmit (Tx) and receive (Rx) DMA engines. The transaction requests from the Tx and Rx DMA engines are arbitrated to allow access to the common DMA master interface. The following two types of arbitrations are supported by programming Bit DA of the ETH_DMAMR register:

- Weighted round-robin arbitration
- Fixed-priority arbitration

The PR[2:0] bit field controls the ratio of the weights between the Tx DMA and the Rx DMA in the weighted round robin scheme.

However, the programmed polarity ratio PR[2:0] in the weighted round-robin scheme is not adhered to, when there is a priority difference between Rx and Tx. In other words when Rx DMA is given higher priority over Tx DMA or vice-versa.

The defect occurs in the following conditions:

- The weighted round robin arbitration scheme is selected by clearing the DA bit of the ETH_DMAMR
- Programming different weights in the TXPR and PR fields of ETH_DMAMR
- Both Tx and Rx DMAs are simultaneously requesting for access.

As a consequence, the expected quality of service (QoS) requirement between Tx and Rx DMA Channels for host bus bandwidth allocation might not get adhered to. This defect might have an impact only if the host bus bandwidth is limited and close to or above the total ethernet line rate traffic. The impact can be in terms of buffer underflow (for Tx in cut-through mode) or Buffer overflows (for Rx). If the host side bandwidth is much more than the ethernet line rate traffic, then this bandwidth allocation of WRR scheme is of no consequence.

Workaround

Operate in fixed priority arbitration mode where the DA bit of the ETH_DMAMR is set with Rx DMA having a higher priority over Tx clearing the TXPR bit. Operate the Tx buffers in Store-and-Forward mode to avoid any buffer underflows/overflows.

2.20.5 Incorrect L4 inverse filtering results for corrupted packets

Description

Received corrupted IP packets with payload (for IPv4) or total (IPv6) length of less than two bytes for L4 source port (SP) filtering or less than four bytes for L4 destination port (DP) filtering are expected to cause a mismatch. However, the inverse filtering unduly flags a match and the corrupted packets are forwarded to the software application. The L4 stack gets incomplete packet and drops it.

Note: The perfect filtering correctly reports a mismatch.

Workaround

None.

2.20.6 IEEE 1588 Timestamp interrupt status bits are incorrectly cleared on write access to the CSR register with similar offset address

Description

When RCWE bit of the ETH_MACCSRSWCR register is set, all interrupt status bits (events) are cleared only when the specific status bits are set.

However, the status bits[3:0] of the ETH_MACTSSR register at address 0x0B20 are unintentionally cleared when 1 is written to the corresponding bit positions in any CSR register with address offset [7:0] = 0x20. The Status bits[3:0] correspond to the following events:

- Timestamp seconds register overflow interrupt TSSOVF
- Auxiliary timestamp trigger snapshot AUXTSTRIG
- Target time interrupt TSTARGT0

- Target time programming error interrupt TSTRGTERR0

This defect occurs only when the software enables the write 1 to clear interrupt status bits, by setting RCWE of the ETH_MACCSRSWCR register.

As a consequence, when any of the target time interrupts or timestamp seconds overflow events occur, the software might inadvertently clear the corresponding status bits and as a concequence de-assert the interrupt, if it first writes to any CSR register at the shadow address (0x0_xx20 or 0x1_xx20). Consequently, the interrupt service routine might not identify the source of these interrupt events, as the corresponding status bits are already cleared.

Note: The timestamp seconds register overflow event is extremely rare (once in ~137 years) and the target time error interrupt can be avoided by appropriate programming. The frequency of target time reached interrupt events depends on the application usage.

Workaround

When RCWE is set and the timestamp event interrupts are enabled, process and clear the MAC timestamp interrupt events first in the interrupt service routine software, so that write operations to other shadow CSR registers are avoided.

2.20.7 Bus error along with Start-of-Packet can corrupt the ongoing transmission of MAC generated packets

Description

If a bus error is asserted along with the start of a new packet while the MAC is transmitting an internally generated packet such as: ARP, PTO or Pause, the error indication aborts the ongoing transmission prematurely and corrupts the MAC generated packet being transmitted.

As a consequence, the MAC generated packet is sent on the line as a runt frame with corrupted FCS. The aborted packet is not retransmitted and can cause:

- Failure of the intended flow control in case of a Pause/PFC packet corruption.
- Delay in ARP handshake from ARP offload engine; the ARP stack recovers because it sends ARP requests periodically
- Delay in PTP response/SYNC packets generated by PTP offload engine; the PTP stack recovers because it sends request packets periodically.

The probability of occurrence of an bus error on the first beat of data and coinciding with a MAC generated packet transmission is very low.

Workaround

None.

2.20.8 Spurious receive watchdog timeout interrupt

Description

Setting the RWTU[1:0] bitfield of the ETH_DMACRXIWTR register to a non-zero value while the RWT[7:0] bitfield is at zero leads to a spurious receive watchdog timeout interrupt (if enabled) and, as a consequence, to executing an unnecessary interrupt service routine with no packets to process.

Workaround

Ensure that the RWTU[1:0] bitfield is not set to a non-zero value while the RWT[7:0] bitfield is at zero. For setting RWT[7:0] and RWTU[1:0] bitfields each to a non-zero value, perform two successive writes. The first is either a byte-wide write to the byte containing the RWT[7:0] bitfield, or a 32-bit write that only sets the RWT[7:0] bitfield and keeps the RWTU[1:0] bitfield at zero. The second is either a byte-wide write to the RWTU[1:0] bitfield or a 32-bit write that sets the RWTU[1:0] bitfield while keeping the RWT[7:0] bitfield unchanged.

2.20.9 Incorrect flexible PPS output interval under specific conditions

Description

The use of the fine correction method for correcting the IEEE 1588 internal time reference, combined with a large frequency drift of the driving clock from the grandmaster source clock, leads to an incorrect interval of the flexible PPS output used in Pulse train mode. As a consequence, external devices synchronized with the flexible PPS output of the device can go out of synchronization.

Workaround

Use the coarse method for correcting the IEEE 1588 internal time reference.

2.20.10 Packets dropped in RMII 10Mbps mode due to fake dribble and CRC error

Description

When operating with the RMII interface at 10 Mbps, the Ethernet peripheral may generate a fake extra nibble of data repeating the last packet (nibble) of the data received from the PHY interface. This results in an odd number of nibbles and is flagged as a dribble error. As the RMII only forwards to the system completed bytes of data, the fake nibble would be ignored and the issue would have no consequence. However, as the CRC error is also flagged when this occurs, the error-packet drop mechanism (if enabled) discards the packets.

Note: Real dribble errors are rare. They may result from synchronization issues due to faulty clock recovery.

Workaround

When using the RMII 10 MHz mode, disable the error-packet drop mechanism by setting the FEP bit of the ETH_MTLRXQOMR register. Accept packets of transactions flagging both dribble and CRC errors.

2.20.11 ARP offload function not effective

Description

When the Target Protocol Address of a received ARP request packet matches the device IP address set in the ETH_MACARPAR register, the source MAC address in the SHA field of the ARP request packet is compared with the device MAC address in ETH_MACA0LR and ETH_MACA0HR registers (Address0), to filter out ARP packets that are looping back.

Instead, a byte-swapped comparison is performed by the device. As a consequence, the packet is forwarded to the application as a normal packet with no ARP indication in the packet status, and the device does not generate an ARP response.

For example, with the Address0 set to 0x665544332211:

- If the SHA field of the received ARP packet is 0x665544332211, the ARP response is generated while it should not.
- If the SHA field of the received ARP packet is 0x112233445566, the ARP response not is generated while it should.

Workaround

Parse the received frame by software and send the ARP response if the source MAC address matches the byte-swapped Address0.

Revision history

Table 5. **Document revision history**

Date	Version	Changes
18-Nov-2019	1	Initial release.
16-Jan-2020	2	Added Section 2.4.1 DMA stream locked when transferring data to/from USART/UART. Added the following ADC limitations: Section 2.8.1 ADC3 conversion data corrupted when switching input channels, Section 2.8.2 ADC3 performance decreased at low frequency, and Section 2.8.3 ADC3 built-in offset calibration not functional. Added the following USART limitations: Section 2.17.1 Anticipated end-of-transmission signaling in SPI slave mode and Section 2.17.2 Data corruption due to noisy receive line.
14-May-2020	3	Added STM32H723xE, STM32H730xB part numbers. Removed STM32H725AE part number. Added revision Z. Added Section 2.1.1 A tamper event does not erase the backup RAM when the backup RAM clock is disabled and Section 2.1.2 A tamper event does not erase the OTFDEC keys when the backup RAM clock is disabled system limitations. Added Section 2.10.1 Overshoot on VREFBUF output and Section 2.10.2 VREFBUF Hold mode cannot be used VREFBUF limitations. Added Section 2.12.2 Consecutive compare event missed in specific conditions and Section 2.12.3 Output compare clear not working with external counter reset timer limitations. Added Section 2.17.3 DMA stream locked when transferring data to/from USART.
31-Aug-2020	4	FMC: Added Section 2.6.3 Unsupported read access with unaligned address. WWDG: Added Section 2.14.1 WWDG not functional when VDD is lower than 2.7 V and VOS0 or VOS1 voltage level is selected. ETHERNET: Removed *Tx DMA may halt while fetching TSO header under specific conditions*. Added Section 2.20.1 The MAC does not provide bus access to a higher priority request after a low priority request is serviced, Section 2.20.3 The Tx DMA fails to recover correctly or corrupts TSO/USO header data on receiving a bus error response from the AHB DMA slave, Section 2.20.4 Incorrectly weighted round robin arbitration between Tx and Rx DMA channels to access the common host bus, Section 2.20.6 IEEE 1588 Timestamp interrupt status bits are incorrectly cleared on write access to the CSR register with similar offset address, Section 2.20.7 Bus error along with Start-of-Packet can corrupt the ongoing transmission of MAC generated packets
30-Sep-2020	5	Added Section 2.8.4 ADC3 injected channel conversion while regular conversion is running may provide corrupted data. Updated Section 2.15.1 RTC interrupt can be masked by another RTC interrupt.

Contents