STM32L496xx/4A6xx device errata

Applicability

This document applies to the part numbers of STM32L496xx/4A6xx 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 RM351.

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
STM32L496xx	STM32L496AE, STM32L496AG, STM32L496QE, STM32L496QG, STM32L496RE, STM32L496RG, STM32L496VE, STM32L496VG, STM32L496ZE, STM32L496ZG
STM32L4A6xx	STM32L4A6AG, STM32L4A6QG, STM32L4A6RG, STM32L4A6VG, STM32L4A6ZG

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32L496xx/4A6xx	B	0x2000

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_IDCODE register.*

1 Summary of device errata

The following table gives a quick reference to the STM32L496xx/4A6xx 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. B
Core	2.1.1	Interrupted loads to SP can cause erroneous behavior	A
	2.1.2	VDIV or VSQRT instructions might not complete correctly when very short ISRs are used	A
	2.1.3	Store immediate overlapping exception return operation might vector to incorrect interrupt	A
System	2.2.1	Dual-bank boot not working when the boot in Flash memory is selected by BOOT0 pin	A
	2.2.2	PCPROP area within a single Flash memory page becomes unprotected at RDP change from Level 1 to Level 0	A
	2.2.3	Data cache might be corrupted during Flash memory read-while-write operation	A
	2.2.4	MSI frequency overshoot upon Stop mode exit	A
	2.2.5	Internal voltage reference corrupted upon Stop mode entry with temperature sensing enabled	A
	2.2.6	VDDA overconsumption under specific condition on PA3 or PB0	N
	2.2.7	Spurious brown-out reset after short run sequence	A
	2.2.8	Full JTAG configuration without NJTRST pin cannot be used	A
	2.2.9	Current injection from VDD to VDDA through analog switch voltage booster	A
	2.2.10	Unstable LSI when it clocks RTC or CSS on LSE	P
	2.2.11	Both GPIOI and GPIOH pull-up or pull-down activation is done through PWR_PUCRH or PWR_PDCRH registers	P
	2.2.12	Dual-bank boot fails with stack pointer placed within aliased SRAM2	A
	2.2.13	FLASH_ECCR corrupted upon reset or power-down occurring during Flash memory program or erase operation	A
	2.2.14	PC13 signal transitions disturb LSE	N
FW	2.3.1	Code segment unprotected if non-volatile data segment length is zero	A
	2.3.2	Code and non-volatile data unprotected upon bank swap	A
DMA	2.4.1	DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	A
FMC	2.5.1	Dummy read cycles inserted when reading synchronous memories	N
	2.5.3	Wrong data read from a busy NAND memory	A
	2.5.4	Spurious clock stoppage with continuous clock feature enabled	A
	2.5.5	Data read might be corrupted when the write FIFO is disabled	A

Function	Section	Limitation	Status Rev. B
QUADSPI	2.6.1	First nibble of data not written after dummy phase	A
	2.6.2	Wrong data from memory-mapped read after an indirect mode operation	A
	2.6.3	Memory-mapped read operations may fail when timeout counter is enabled	P
	2.6.4	Memory-mapped access in indirect mode clearing QUADSPI_AR register	P
ADC	2.7.1	Writing ADC_JSQR when JADCSTART and JQDIS are set may lead to incorrect behavior	N
	2.7.2	Wrong ADC result if conversion done late after calibration or previous conversion	A
	2.7.3	Spurious temperature measurement due to spike noise	A
COMP	2.8.1	Comparator outputs cannot be configured in open-drain	N
TIM	2.11.1	One-pulse mode trigger not detected in master-slave reset + trigger configuration	P
	2.11.2	HSE/32 is not available for TIM16 input capture if RTC clock is disabled or other than HSE	A
LPTIM	2.12.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	A
	2.12.2	Device may remain stuck in LPTIM interrupt when clearing event flag	P
	2.12.3	LPTIM1 outputs cannot be configured as open-drain	N
RTC and TAMP	2.13.1	RTC calendar registers are not locked properly	A
	2.13.2	RTC interrupt can be masked by another RTC interrupt	A
	2.13.3	Calendar initialization may fail in case of consecutive INIT mode entry	A
	2.13.4	Alarm flag may be repeatedly set when the core is stopped in debug	N
	2.13.5	RTC_REFIN and RTC_OUT on PB2 not operating in Stop 2 mode	P
I2C	2.14.1	10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave	A
	2.14.3	Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period	P
	2.14.4	Spurious bus error detection in master mode	A
	2.14.5	Last-received byte loss in reload mode	P
	2.14.6	Spurious master transfer upon own slave address match	P
	2.14.8	OVR flag not set in underrun condition	N
	2.14.9	Transmission stalled after first byte transfer	A
USART	2.15.1	nRTS is active while RE = 0 or UE = 0	A
LPUART	2.16.1	LPUART1 outputs cannot be configured as open-drain	N
SPI	2.17.1	BSY bit may stay high at the end of data transfer in slave mode	A
	2.17.2	Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters	A
	2.17.3	SPI master communication failure at high fPCLK within the specified range	N
SDMMC	2.18.1	Wrong CCRCFAIL status after a response without CRC is received	P
bxCAN	2.19.1	bxCAN time-triggered communication mode not supported	N
OTG_FS	2.20.1	Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_FS registers	A

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

Table 4. Summary of device documentation errata

Function	Section	Documentation erratum
DMA	2.4.2	Byte and half-word accesses not supported
FMC	2.5.2	Missing information on prohibited 0xFF value of NAND transaction wait timing
TSC	2.9.1	Inhibited acquisition in short transfer phase configuration
AES (1)	2.10.1	TAG computation in GCM encryption mode
RTC and TAMP	2.13.6	Setting GPIO properties of PC13 used as RTC_ALARM open-drain output
I2C	2.14.2	Wrong behavior in Stop mode when wakeup from Stop mode is disabled in I2C
	2.14.7	START bit is cleared upon setting ADDRCF, not upon address match

1. *Only applicable to STM32L4A6xx*

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®-M4F core revision r0p1 is available from http://infocenter.arm.com.

2.1.1 Interrupted loads to SP can cause erroneous behavior

This limitation is registered under Arm ID number 752770 and classified into “Category B”. Its impact to the device is minor.

Description

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

The affected instructions that can result in the load transaction being repeated are:

- LDR SP, [Rn],#imm
- LDR SP, [Rn,#imm]!
- LDR SP, [Rn,#imm]
- LDR SP, [Rn]
- LDR SP, [Rn,Rm]

The affected instructions that can result in the stack-pointer being loaded from an incorrect memory address are:

- LDR SP,[Rn],#imm
- LDR SP,[Rn,#imm]!

As compilers do not generate these particular instructions, the limitation is only likely to occur with hand-written assembly code.

Workaround

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

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

This limitation is registered under Arm ID number 776924 and classified into “Category B”. Its impact to the device is limited.

Description

The VDIV and VSQRT instructions take 14 cycles to execute. When an interrupt is taken a VDIV or VSQRT instruction is not terminated, and completes its execution while the interrupt stacking occurs. If lazy context save of floating point state is enabled then the automatic stacking of the floating point context does not occur until a floating point instruction is executed inside the interrupt service routine.

Lazy context save is enabled by default. When it is enabled, the minimum time for the first instruction in the interrupt service routine to start executing is 12 cycles. In certain timing conditions, and if there is only one or two instructions inside the interrupt service routine, then the VDIV or VSQRT instruction might not write its result to the register bank or to the FPSCR.

The failure occurs when the following condition is met:

1. The floating point unit is enabled
2. Lazy context saving is not disabled
3. A VDIV or VSQRT is executed
4. The destination register for the VDIV or VSQRT is one of s0 - s15
5. An interrupt occurs and is taken
6. The interrupt service routine being executed does not contain a floating point instruction
7. Within 14 cycles after the VDIV or VSQRT is executed, an interrupt return is executed

A minimum of 12 of these 14 cycles are utilized for the context state stacking, which leaves 2 cycles for instructions inside the interrupt service routine, or 2 wait states applied to the entire stacking sequence (which means that it is not a constant wait state for every access).

In general, this means that if the memory system inserts wait states for stack transactions (that is, external memory is used for stack data), then this erratum cannot be observed.

The effect of this erratum is that the VDIV or VQSRT instruction does not complete correctly and the register bank and FPSCR are not updated, which means that these registers hold incorrect, out of date, data.

Workaround

A workaround is only required if the floating point unit is enabled. A workaround is not required if the stack is in external memory.

There are two possible workarounds:

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

2.1.3 Store immediate overlapping exception return operation might vector to incorrect interrupt

This limitation is registered under Arm ID number 838869 and classified into “Category B (rare)”. Its impact to the device is minor.

Description

The core includes a write buffer that permits execution to continue while a store is waiting on the bus. Under specific timing conditions, during an exception return while this buffer is still in use by a store instruction, a late change in selection of the next interrupt to be taken might result in there being a mismatch between the interrupt acknowledged by the interrupt controller and the vector fetched by the processor.

The failure occurs when the following condition is met:

1. The handler for interrupt A is being executed.
2. Interrupt B, of the same or lower priority than interrupt A, is pending.
3. A store with immediate offset instruction is executed to a bufferable location.
   - STR/STRH/STRB <Rt>, [<Rn>,#imm]
   - STR/STRH/STRB <Rt>, [<Rn>,#imm]!
   - STR/STRH/STRB <Rt>, [<Rn>],#imm
4. Any number of additional data-processing instructions can be executed.
5. A BX instruction is executed that causes an exception return.
6. The store data has wait states applied to it such that the data is accepted at least two cycles after the BX is executed.
   - Minimally, this is two cycles if the store and the BX instruction have no additional instructions between them.
   - The number of wait states required to observe this erratum needs to be increased by the number of cycles between the store and the interrupt service routine exit instruction.
7. Before the bus accepts the buffered store data, another interrupt C is asserted which has the same or lower priority as A, but a greater priority than B.

Example:

The processor should execute interrupt handler C, and on completion of handler C should execute the handler for B. If the conditions above are met, then this erratum results in the processor erroneously clearing the pending state of interrupt C, and then executing the handler for B twice. The first time the handler for B is executed it will be at interrupt C's priority level. If interrupt C is pended by a level-based interrupt which is cleared by C's handler then interrupt C will be pended again once the handler for B has completed and the handler for C will be executed.

As the STM32 interrupt C is level based, it eventually becomes pending again and is subsequently handled.

Workaround

For software not using the memory protection unit, this erratum can be worked around by setting DISDEFWBUF in the Auxiliary Control Register.

In all other cases, the erratum can be avoided by ensuring a DSB occurs between the store and the BX instruction. For exception handlers written in C, this can be achieved by inserting the appropriate set of intrinsics or inline assembly just before the end of the interrupt function, for example:

ARMCC:

```
...
__schedule_barrier();
__asm{DSB};
__schedule_barrier();
}
```

GCC:

```
...
__asm volatile ("dsb 0xf":::"memory");
}
```

2.2 System

2.2.1 Dual-bank boot not working when the boot in Flash memory is selected by BOOT0 pin

Description

When the nSWBoot0 option bit is set and the BOOT0 pin level is low, the dual-bank boot does not work (BFB2 option bit = 1).

The user code is only executed when BANK 0 is valid.

Workaround

Do not rely on the BOOT0 pin to select the boot in Flash memory. Instead, use the option bytes for that purpose, setting the nSWBoot0 option bit to 0, the nBOOT0 option bit to 1, and the BFB2 option bit to 1. This setting allows the correct check of the address 0 of the two memory banks, in order to execute from the correct one.

2.2.2 PCPROP area within a single Flash memory page becomes unprotected at RDP change from Level 1 to Level 0

Description

With PCROP_RDP option bit cleared, the change of RDP from Level 1 to Level 0 normally results in erasure of Flash memory banks except the Flash memory pages containing the PCROP area. The PCROP area remains read-protected.

This operates as expected if the PCROP area crosses the limits of at least one Flash memory page, which is always true if PCROP area size exceeds 2 Kbytes. The limitation occurs if the PCROP area is fully contained within one single Flash memory page. Upon the RDP change from Level 1 to Level 0, the Flash memory bank with PCROP area is not erased and the read protection of the PCROP area is removed.

Workaround

Always define PCROP area such that it crosses limits of at least one Flash memory page.

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

Description

When a write to the internal Flash memory is done, the data cache is normally updated to reflect the data value update. During this data cache update, a read to the other Flash memory bank may occur; this read can corrupt the data cache content and subsequent read operations at the same address (cache hits) will be corrupted.

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

Workaround

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

Code example:

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

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

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

/* Reset data cache */
__HAL_FLASH_DATA_CACHE_RESET();

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

2.2.4 MSI frequency overshoot upon Stop mode exit

Description

When

- the system is clocked by the MSI clock, and
- MSI is selected as system clock source upon wakeup from Stop mode, and
- a wakeup event occurs only a few system clock cycles before entering Stop mode,

then upon the exit from Stop mode, the MSI frequency can overshoot above its selected range.

The limitation applies to all Stop modes: Stop 0, Stop 1, and Stop 2.

Workaround

Apply the following sequence:

1. Select HSI16 as system clock.
2. Shut MSI down.
3. Wait for MSIRDY to go low (after 6 MSI clock cycles).
4. Mask interrupts by setting PRIMASK (in the core).
5. Initiate Stop mode entry, with MSI selected as system clock source upon wakeup. At this point, the systems enters Stop mode (unless an early wakeup event occurs). The following steps are done upon Stop mode exit.
6. Enable MSI.

7. Wait for MSIRDY to go high.
8. Select MSI as system clock.
9. Unmask interrupts by clearing PRIMASK.

The steps 6 through 8 are required to cover the case of the MCU not entering Stop mode after the step 5 (due to an early wakeup event), thus maintaining HSI16 as system clock.

This workaround guarantees the best wakeup time.

2.2.5 Internal voltage reference corrupted upon Stop mode entry with temperature sensing enabled

Description

When entering Stop mode with the temperature sensor channel and the associated ADC(s) enabled, the internal voltage reference may be corrupted.

The occurrence of the corruption depends on the supply voltage and the temperature.

The corruption of the internal voltage reference may cause:

- an overvoltage in V_{CORE} domain
- an overshoot / undershoot of internal clock (LSI, HSI, MSI) frequencies
- a spurious brown-out reset

The limitation applies to Stop 1 and Stop 2 modes.

Workaround

Before entering Stop mode:

- Disable the ADC(s) using the temperature sensor signal as input, and/or
- Disable the temperature sensor channel, by clearing the CH17SEL bit of the ADCx_CCR register.

Disabling both the ADC(s) and the temperature sensor channel reduces the power consumption during Stop mode.

2.2.6 V_{DDA} overconsumption under specific condition on PA3 or PB0

Description

An overconsumption can appear on V_{DDA} when all the following conditions are met at the same time:

- A voltage V_{PAD} is applied on PA3 or PB0 with (V_{DDA} + 50 mV) < V_{PAD} < (V_{DDA} + 600 mV)
- The OPAMP output is not used (OPAMPx_VOUT pins are not driven by the OPAMP)
- Temperature is below 0 °C

This extra consumption is constant and can reach up to 1 mA.

Workaround

None except avoiding the previous conditions.

2.2.7 Spurious brown-out reset after short run sequence

Description

When the MCU wakes up from Stop mode and enters the Stop mode again within a short period of time, a spurious brown-out reset may be generated.

This limitation depends on the supply voltage (see the following table).

Table 5. Minimum run time

V_{DD} supply voltage (V)	Minimum run time (µs)
1.71	15
1.8	13
2.0	11
2.2	9
2.4	8
2.6	6
2.8	5
3.0	3
3.2 and above	2

The minimum run time defined in the previous table corresponds to the firmware execution time on the core. There is no need to add a delay for the wakeup time. Note also that the MCO output length is longer than the firmware execution time.

This limitation applies to Stop 1 and Stop 2 modes.

Workaround

Ensure that the run time between exiting Stop mode and entering Stop mode again is long enough not to generate a brown-out reset. This can be done by adding a software loop or using a timer to add a delay; the system clock frequency can be reduced during this waiting loop in order to minimize power consumption.

2.2.8 Full JTAG configuration without NJTRST pin cannot be used

Description

When using the JTAG debug port in Debug mode, the connection with the debugger is lost if the NJTRST pin (PB4) is used as a GPIO or for an alternate function other than NJTRST. Only the 4-wire JTAG port configuration is impacted.

Workaround

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

2.2.9 Current injection from V_{DD} to V_{DDA} through analog switch voltage booster

Description

With V_{DDA} below 2.4 V and V_{DD} above 3 V, a small current injected from VDD line to VDDA line may cause V_{DDA} to exceed its nominal value.

This current injection only occurs when the I/O analog switch voltage booster is disabled (the BOOSTEN bit of the SYSCFG_CFGR1 register is cleared) and at least one of the analog peripherals (ADC, COMP, DAC, or OPAMP) is enabled.

Workaround

Enable the I/O analog switch voltage booster, by setting the BOOSTEN bit of the SYSCFG_CFGR1 register. when V_{DDA} is below 2.4 V and V_{DD} is above 3 V.

2.2.10 Unstable LSI when it clocks RTC or CSS on LSE

Description

The LSI clock can become unstable (duty cycle different from 50 %) and its maximum frequency can become significantly higher than 32 kHz, when:

- LSI clocks the RTC, or it clocks the clock security system (CSS) on LSE (which holds when the LSECSSON bit set), and
- the V_{DD} power domain is reset while the backup domain is not reset, which happens:
  - upon exiting Shutdown mode
  - if V_{BAT} is separate from V_{DD} and V_{DD} goes off then on
  - if V_{BAT} is tied to V_{DD} (internally in the package for products not featuring the VBAT pin, or externally) and a short (< 1 ms) V_{DD} drop under V_{DD}(min) occurs

Workaround

Apply one of the following measures:

- Clock the RTC with LSE or HSE/32, without using the CSS on LSE
- If LSI clocks the RTC or when the LSECSSON bit is set, reset the backup domain upon each V_{DD} power up (when the BORRSTF flag is set). If V_{BAT} is separate from V_{DD}, also restore the RTC configuration, backup registers and anti-tampering configuration.

2.2.11 Both GPIOI and GPIOH pull-up or pull-down activation is done through PWR_PUCRH or PWR_PDCRH registers

Description

The activation of Port I in pull-up or pull-down configuration through registers Power Port I pull-up control register (PWR_PUCRI) or Power Port I pull-down control register (PWR_PDCRI) does not work.

Instead the Port H control register is controlling both Port I and Port H, bit per bit, through Port H pull-up control register (PWR_PUCRH) or Power Port H pull-down control register (PWR_PDCRH).

For example PWR_PUCRH[0] will activate pull-up on both PH[0] and PI[0].

Workaround

When selecting the GPIO mapping for the applicative board, make sure that activating pull-up or pull-down on any bit of GPIO Port I (resp. Port H) will not create an electrical issue by simultaneously activating pull-up or pull-down on the corresponding GPIO bit of Port H (resp. Port I).

2.2.12 Dual-bank boot fails with stack pointer placed within aliased SRAM2

Description

If dual-bank boot is selected through option bits, the bootloader checks the contents of either Flash memory bank for validity. It considers a bank content valid if its first word (determining the stack pointer address) contains a value from 0x2000 0000 to 0x2003 FFFF (SRAM1) or from 0x1000 0000 to 0x1000 FFFF (SRAM2). Any other value is invalid.

As a consequence, the dual-bank boot fails if the first word of either Flash memory bank points to a location within the SRAM2 aliased at the address 0x2004 0000.

This limitation applies to STM32L496xx and STM32L4A6xx.

Workaround

Do not use aliased SRAM2 address range in conjunction with dual-bank boot.

2.2.13 FLASH_ECCR corrupted upon reset or power-down occurring during Flash memory program or erase operation

Description

Reset or power-down occurring during a Flash memory location program or erase operation, followed by a read of the same memory location, may lead to a corruption of the FLASH_ECCR register content.

Workaround

Under such condition, erase the page(s) corresponding to the Flash memory location.

2.2.14 PC13 signal transitions disturb LSE

Description

The PC13 port toggling disturbs the LSE clock.

Workaround

None.

2.3 FW

2.3.1 Code segment unprotected if non-volatile data segment length is zero

Description

If during FW configuration the length of firewall-protected non-volatile data segment is set to zero through the LENG[21:8] bitfield of the FW_NVDSL register, the firewall protection of code segment does not operate.

Workaround

Always set the LENG[21:8] bitfield of the FW_NVDSL register to a non-zero value, even if no firewall protection of data in the non-volatile data segment is required.

2.3.2 Code and non-volatile data unprotected upon bank swap

Description

With firewall-protected code and non-volatile data segments located in the same Flash memory bank, both segments become unprotected (available to illegal access) upon Flash memory bank swap by software.

Workaround

Map the firewall-protected code segment in one Flash memory bank and the non-volatile data segment in another Flash memory bank in a symmetric way (same relative address and same length).

2.4 DMA

2.4.1 DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear

Description

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

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

Workaround

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

2.4.2 Byte and half-word accesses not supported

Description

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

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

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

Workaround

No application workaround is required.

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

Description

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

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

This is a documentation error rather than a device limitation.

Workaround

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

2.5.3 Wrong data read from a busy NAND memory

Description

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

Workaround

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

2.5.4 Spurious clock stoppage with continuous clock feature enabled

Description

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

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

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

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

Workaround

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

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

Description

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

Workaround

Enable the write FIFO.

2.6 QUADSPI

2.6.1 First nibble of data not written after dummy phase

Description

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

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

Workaround

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

Example:

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

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

Description

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

Workaround

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

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

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

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

Description

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

Workaround

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

2.6.4 Memory-mapped access in indirect mode clearing QUADSPI_AR register

Description

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

Workaround

Adopt one of the following measures:

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

2.7 ADC

2.7.1 Writing ADC_JSQR when JADCSTART and JQDIS are set may lead to incorrect behavior

Description

Some reference manual revisions specifiy that the ADC_JSQR register can be written when an injected conversion is ongoing (JADCSTART = 1). This may lead to unpredictable ADC behavior if the queues of context are not enabled (JQDIS = 1).

Workaround

No application workaround is required for this description inaccuracy issue.

2.7.2 Wrong ADC result if conversion done late after calibration or previous conversion

Description

The result of an ADC conversion done more than 1 ms later than the previous ADC conversion or ADC calibration might be incorrect.

Workaround

Perform two consecutive ADC conversions in single, scan or continuous mode. Reject the result of the first conversion and only keep the result of the second.

2.7.3 Spurious temperature measurement due to spike noise

Description

Depending on the MCU activity, internal interference may cause temperature-dependent spike noise on the temperature sensor output to the ADC, resulting in occasional spurious (outlying) temperature measurement.

Workaround

Perform a series of measurements and process the acquired data samples such as to obtain a mean value not affected by the outlying samples.

For this, it is recommended to use interquartile mean (IQM) algorithm with at least 64 samples. IQM is based on rejecting the quarters (quartiles) of sample population with the lowest and highest values and on computing the mean value only using the remaining (interquartile) samples.

The acquired sample values are first sorted from lowest to highest, then the sample sequence is truncated by removing the lowest and highest sample quartiles.

Example:

Table 6. Measurement result after IQM post-processing

Data	Sample												Mean
	1	2	3	4	5	6	7	8	9	10	11	12	
Acquired	17.2	10.92	9.56	2.12	9.82	10.72	10.6	3.5	9.46	9.78	9.5	1.1	8.69
Sorted	1.1	2.12	3.5	9.46	9.5	9.56	9.78	9.82	10.6	10.72	10.92	17.2	8.69
Truncated	-	-	-	9.46	9.5	9.56	9.78	9.82	10.6	-	-	-	9.79

The measurement result after the IQM post-processing in the example is 9.79. For consistent results, use a minimum of 64 samples. It is recommended to optimize the code performing the sort task such as to minimize its processing power requirements.

2.8 COMP

2.8.1 Comparator outputs cannot be configured in open-drain

Description

Comparator outputs are always forced in push-pull mode whatever the GPIO output type configuration bit value.

Workaround

None.

2.9 TSC

2.9.1 Inhibited acquisition in short transfer phase configuration

Description

Some revisions of the reference manual may omit the information that the following configurations of the TSC_CR register are forbidden:

- The PGPSC[2:0] bitfield set to 000 and the CTPL[3:0] bitfield to 0000 or 0001
- The PGPSC[2:0] bitfield set to 001 and the CTPL[3:0] bitfield to 0000

Failure to respect this restriction leads to an inhibition of the acquisition.

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

Workaround

No application workaround is required.

2.10 AES

2.10.1 TAG computation in GCM encryption mode

Description

Some revisions of the reference manual may omit the following application guidelines for the device to correctly compute GCM encryption authentication tags when the input data in the last block is inferior to 128 bits.

During GCM encryption payload phase and before inserting a last plaintext block smaller than 128 bits, apply the following steps:

1. Disable the AES peripheral by clearing the EN bit of the AES_CR register.
2. Change the mode to CTR by writing 010 to the CHMOD[2:0] bitfield of the AES_CR register.
3. Pad the last block (smaller than 128 bits) with zeros to have a complete block of 128 bits, then write it into AES_DINR register.
4. Upon encryption completion, read the 128-bit ciphertext from the AES_DOUTR register and store it as intermediate data.
5. Change again the mode to GCM by writing 011 to the CHMOD[2:0] bitfield of the AES_CR register.
6. Select Final phase by writing 11 to the GCMPH[1:0] bitfield of the AES_CR register.
7. In the intermediate data, set to zero the bits corresponding to the padded bits of the last block of payload, then insert the resulting data into AES_DINR register.
8. Wait for operation completion, and read data on AES_DOUTR. This data is to be discarded.
9. Apply the normal Final phase as described in the datasheet

This is a documentation issue rather than a product limitation.

Workaround

No further application workaround is required, provided that these guidelines are respected.

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 HSE/32 is not available for TIM16 input capture if RTC clock is disabled or other than HSE

Description

If the RTC clock is either disabled or other than HSE, the HSE/32 clock is not available for TIM16 input capture even if selected (bitfield TI1_RMP[2:0] = 101 in the TIM16_OR1 register).

Workaround

Apply the following procedure:

1. Enable the power controller clock (bit PWREN = 1 in the RCC_APB1ENR1 register).
2. Disable the backup domain write protection (bit DBP = 0 in the PWR_CR1 register).
3. Enable RTC clock and select HSE as clock source for RTC (bits RTCSEL[1:0] = 11 and bit RTCEN = 1 in the RCC_BDCR register).
4. Select the HSE/32 as input capture source for TIM16 (bitfield TI1_RMP[2:0] = 101 in the TIM16_OR1 register).

Alternatively, use TIM17 that implements the same features as TIM16, and is not affected by the limitation described.

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_APBxRSTRz 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.

2.12.3 LPTIM1 outputs cannot be configured as open-drain

Description

LPTIM1 outputs are set in push-pull mode regardless of the configuration of corresponding GPIO outputs.

Workaround

None.

2.13 RTC and TAMP

2.13.1 RTC calendar registers are not locked properly

Description

When reading the calendar registers with BYPSHAD = 0, the RTC_TR and RTC_DR registers may not be locked after reading the RTC_SSR register. This happens if the read operation is initiated one APB clock period before the shadow registers are updated. This can result in a non-consistency of the three registers. Similarly, the RTC_DR register can be updated after reading the RTC_TR register instead of being locked.

Workaround

Apply one of the following measures:

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

2.13.2 RTC interrupt can be masked by another RTC interrupt

Description

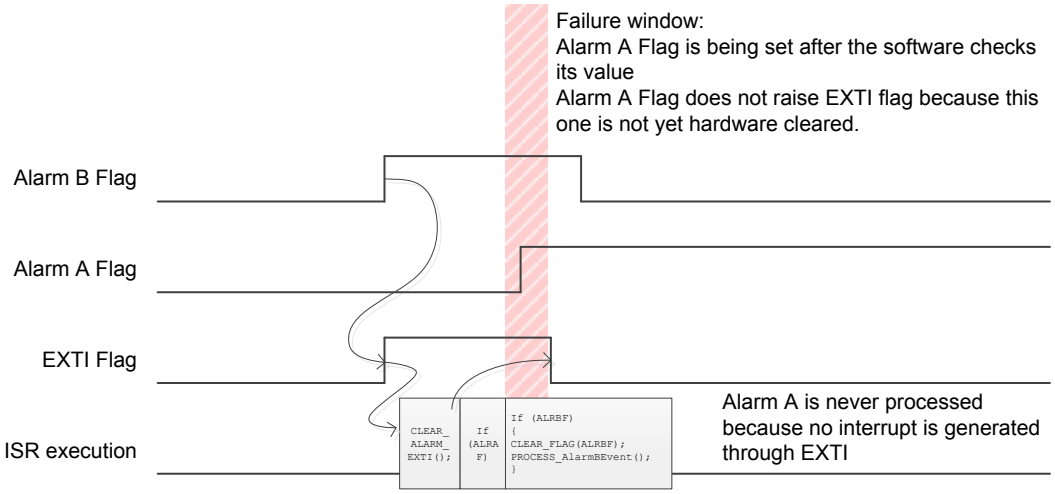
One RTC interrupt request can mask another RTC interrupt request if they share the same EXTI configurable line. For example, interrupt requests from Alarm A and Alarm B or those from tamper and timestamp events are OR-ed to the same EXTI line (refer to the *EXTI line connections* table in the *Extended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

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

Alarm interrupt service routine:

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

Figure 1. Masked RTC interrupt



Workaround

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

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

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

Description

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

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

Workaround

After existing the initialization mode, clear the BYPSHAD bit (if set) then wait for RSF to rise, before entering the initialization mode again.

Note: It is recommended to write all registers in a single initialization session to avoid accumulating synchronization delays.

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

Description

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

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

Workaround

None.

2.13.5 RTC_REFIN and RTC_OUT on PB2 not operating in Stop 2 mode

Description

In Stop 2 low-power mode, the RTC_REFIN function does not operate and the RTC_OUT function does not operate if mapped on the PB2 pin.

Workaround

Apply one of the following measures:

- Use Stop 1 mode instead of Stop 2. This ensures the operation of both functions.
- Map RTC_OUT to the PC13 pin. This ensures the operation of the RTC_OUT function in either low-power mode. However, it has no effect to the RTC_REFIN function.

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

Description

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

Note: Enabling the internal pull-up function through the PC13 GPIO settings allows sparing an external pull-up resistor.
This is a documentation issue rather than a product limitation.

Workaround

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

2.14 I2C

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

Description

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

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

Workaround

In 10-bit-address master mode, if both NACKF flag and START bit get simultaneously set, proceed as follows:

1. Wait for the STOP condition detection (STOPF = 1 in I2C_ISR register).
2. Disable the I2C peripheral.
3. Wait for a minimum of three APB cycles.
4. Enable the I2C peripheral again.

2.14.2 Wrong behavior in Stop mode when wakeup from Stop mode is disabled in I2C

Description

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

Some reference manual revisions may omit this information.

Failure to respect the above while the MCU operating as slave or as master in multi-master topology enters Stop mode during a transfer ongoing on the I²C-bus may lead to the following:

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

   The occurrence of such condition depends on the timing configuration, peripheral clock frequency, and I²C-bus frequency.

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

Workaround

No application workaround is required.

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

Description

The I²C-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 I²C-bus SDA line when $t_{SU;DAT}$ is smaller than one I2C kernel clock (I²C-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 I²C-bus standard, the minimum I2CCLK frequencies are as follows:

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

2.14.4 Spurious bus error detection in master mode

Description

In master mode, a bus error can be detected spuriously, with the consequence of setting the BERR flag of the I2C_SR register and generating bus error interrupt if such interrupt is enabled. Detection of bus error has no effect on the I²C-bus transfer in master mode and any such transfer continues normally.

Workaround

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

2.14.5 Last-received byte loss in reload mode

Description

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

- I²C-bus stretching is enabled (NOSTRETCH = 0)
- RELOAD bit of the I2C_CR2 register is set
- NBYTES bitfield of the I2C_CR2 register is set to N greater than 1
- byte N is received on the I²C-bus, raising the TCR flag
- N - 1 byte is not yet read out from the data register at the instant TCR is raised,

then the SCL line is pulled low (I²C-bus clock stretching) and the transfer of the byte N from the shift register to the data register inhibited until the byte N-1 is read and NBYTES bitfield reloaded with a new value, the latter of which also clears the TCR flag. As a consequence, the software cannot get the byte N and use its content before setting the new value into the NBYTES field.

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

Workaround

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

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

2.14.6 Spurious master transfer upon own slave address match

Description

When the device is configured to operate at the same time as master and slave (in a multi- master I²C-bus application), a spurious master transfer may occur under the following condition:

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

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

Workaround

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

Normal mode (SBC = 0):

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

Slave byte control mode (SBC = 1):

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

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

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

2.14.7 START bit is cleared upon setting ADDRCF, not upon address match

Description

Some reference manual revisions may state that the START bit of the I2C_CR2 register is cleared upon slave address match event.

Instead, the START bit is cleared upon setting, by software, the ADDRCF bit of the I2C_ICR register, which does not guarantee the abort of master transfer request when the device is being addressed as slave. This product limitation and its workaround are the subject of a separate erratum.

Workaround

No application workaround is required for this description inaccuracy issue.

2.14.8 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.

2.14.9 Transmission stalled after first byte transfer

Description

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

Workaround

Apply one of the following measures:

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

2.15 USART

2.15.1 nRTS is active while RE = 0 or UE = 0

Description

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

Workaround

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

2.16 LPUART

2.16.1 LPUART1 outputs cannot be configured as open-drain

Description

LPUART1 outputs are set in push-pull mode regardless of the configuration of corresponding GPIO outputs.

Workaround

None.

2.17 SPI

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

Description

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

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

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

Workaround

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

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

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

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

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

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

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

Description

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

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

Workaround

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

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

2.17.3 SPI master communication failure at high f_{PCLK} within the specified range

Description

The SPI peripheral configured as master, with the SPIx_CR1 register's CPHA bit set and BR[2:0] bitfield written with 001 ($f_{PCLK}/4$), may spuriously generate an extra clock pulse at the end of the last data frame to transfer (at the end of which the clock is expected to stop) if the PCLK frequency exceeds the values as per the table. This leads to a desynchronization of the SPI data flow.

Table 7. Maximum f_{PCLK} for safe SPI operation

SPI instance	f_{PCLK}(max) [MHz]	
	Range 1	Range 2
SPI1	58.1	23.4
SPI2	53.3	21.1
SPI3	62.3	25.2

Workaround

None.

2.18 SDMMC

2.18.1 Wrong CCRCFAIL status after a response without CRC is received

Description

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

Workaround

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

2.19 bxCAN

2.19.1 bxCAN time-triggered communication mode not supported

Description

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

Workaround

None.

2.20 OTG_FS

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

Description

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

Workaround

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

Revision history

Table 8. **Document revision history**

Date	Version	Changes
24-Feb-2016	1	Initial release.
05-Sep-2016	2	Added: • Section 2.1.2: PCPROP area within a single Flash memory page becomes unprotected at RDP change from level1 to level0 • Section 2.4.2: Wrong ADC conversion results when delay between calibration and first conversion or between 2 consecutive conversions is too long • Section 2.9.4: Master new transfer cannot be launched if first part of the 10-bit address is NOT Acknowledged by the slave • Section 2.5: LPTIM, Section 2.6: TIM16, Section 2.7: LPUART, Section 2.8: JTAG, Section 2.10: TSC, Section 2.11: AES Modified: • Table 3: Summary of silicon limitation • Section 2.12.1: MMC stream write of less than 7 bytes does not work correctly
09-Jan-2017	3	Added: • Section 2.1.1: Dual bank boot not working in RDP level 1 when the boot in flash is selected by BOOT0 pin • Section 2.1.3: Data Cache might be corrupted during Flash Read While Write operation • Section 2.1.4: MSI frequency overshoot upon Stop mode exit • Section 2.1.5: Internal voltage reference corrupted upon Stop mode entry with temperature sensing enabled • Section 2.3.1: First nibble of data is not written after dummy phase • Section 2.3.2: Wrong data can be read in memory-mapped after an indirect mode operation Modified: • Table 4: Summary of silicon limitation • Applicability
01-Mar-2017	4	Added: • Section 2.1.7: PCPROP area within a single Flash memory page becomes unprotected at RDP change from level1 to level0 • Section 2.9.5: START bit is not cleared when the address is not acknowledged by the slave device Modified: • Table 4: Summary of silicon limitation • Section 2.1.4: MSI frequency overshoot upon Stop mode exit • Section 2.3.1: First nibble of data is not written after dummy phase
08-Jun-2017	5	Added: • Section 2.1.8: Spurious Brown Out Reset after short Run sequence • Section 2.1.6: OPAMP output: VDDA overconsumption • Section 2.9.6: Last received byte can be lost when using Reload mode with NBYTES > 1 • Section 2.5.2: MCU may remain stuck in LPTIM interrupt when entering Stop mode • Section 2.14: OTG_FS Modified: • cover page: replaced former “Silicon identification” with Applicability, and former table Device identification with Table 2: Device variants • Table 4: Summary of silicon limitation

Date	Version	Changes
		• Section 2: STM32L496xx STM32L4A6xx silicon limitations
20-Dec-2017	6	Added: • Section 2.3.1: Code segment unprotected if non-volatile data segment length is zero • Section 2.3.2: Code and non-volatile data unprotected upon bank swap Removed: • the limitation “START bit is not cleared when the address is not acknowledged by the slave device”, as describing the same limitation as Section 2.13.1 Modified: • the structure of the document • move of core limitation to Section 2: Description of device limitations • section TIM16 renamed in TIM • reference manual (RM0351) associated with this document • alignment of the order of limitations with the reference manual • limitation descriptors in Section 2.13: I2C
16-Feb-2018	7	Added: • Section 2.1.2: VDIV or VSQRT instructions might not complete correctly when very short ISRs are used • Section 2.1.3: Store immediate overlapping exception return operation might vector to incorrect interrupt • Section 2.6.3: Spurious temperature measurement due to spike noise • Section 2.16.1: BSY bit may stay high at the end of data transfer in slave mode • Section 2.16.2: Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters • third-party logo in Section 2: Description of device limitations Removed: • former section 2.2.7., as duplicating Section 2.2.2: PCPROP area within a single Flash memory page becomes unprotected at RDP change from Level 1 to Level 0 Modified: • Section 2.1.1: Interrupted loads to SP can cause erroneous behavior aligned with Arm’s description • Section 2.2.4: MSI frequency overshoot upon Stop mode exit • Section 2.3.2: Code and non-volatile data unprotected upon bank swap • minor modifications (language, style or simplification) in some other sections • Document identification number in the page footers, from DocID026121 to ES0335
18-Apr-2018	8	Added: • Section 2.12.1: RTC_REFIN and RTC_OUT on PB2 not operating in Stop 2 mode • Section 2.12.2: RTC calendar registers are not locked properly • Section 2.12.3: RTC interrupt can be masked by another RTC interrupt
06-Jun-2018	9	Added: • Section 2.13.4: Spurious bus error detection in master mode • Section Spurious temperature measurement due to spike noise limitation Modified: • Table 3: Summary of device limitations updated

Date	Version	Changes
		• Section 2.6.2: Writing the register ADCx_JSQR when JADCSTART=1 and JQDIS=1 might lead to incorrect behavior moved from TEMP to Section 2.6: ADC
25-Mar-2019	10	Added: • Section 2.2.9 Current injection from VDD to VDDA through analog switch voltage booster • Section 2.2.10 Unstable LSI when it clocks RTC or CSS on LSE • Section 2.2.11 Both GPIOI and GPIOH pull-up or pull-down activation is done through PWR_PUCRH or PWR_PDCRH registers • Section 2.4.1 DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear • Section 2.4.2 Byte and half-word accesses not supported • Section 2.5.2 Missing information on prohibited 0xFF value of NAND transaction wait timing • Section 2.5.3 Wrong data read from a busy NAND memory • Section 2.5.4 Spurious clock stoppage with continuous clock feature enabled • Section 2.5.5 Data read might be corrupted when the write FIFO is disabled • Section 2.6.3 Memory-mapped read operations may fail when timeout counter is enabled • Section 2.6.4 Memory-mapped access in indirect mode clearing QUADSPI_AR register • Section 2.11.1 One-pulse mode trigger not detected in master-slave reset + trigger configuration • Section 2.12.2 Device may remain stuck in LPTIM interrupt when clearing event flag • Section 2.13.3 Calendar initialization may fail in case of consecutive INIT mode entry • Section 2.13.4 Alarm flag may be repeatedly set when the core is stopped in debug • Section 2.14.7 START bit is cleared upon setting ADDRCF, not upon address match • Section 2.14.8 OVR flag not set in underrun condition • Section 2.14.9 Transmission stalled after first byte transfer Modified: • Section 2.2.1 Dual-bank boot not working when the boot in Flash memory is selected by BOOT0 pin (removed RDP condition) • Section 2.7.2 Wrong ADC result if conversion done late after calibration or previous conversion - workaround qualifier corrected • Section 2.10.1 TAG computation in GCM encryption mode, Section 2.9.1 Inhibited acquisition in short transfer phase configuration, and Section 2.14.2 Wrong behavior in Stop mode when wakeup from Stop mode is disabled in I2C references moved to the Table 4
03-Jul-2019	11	Added erratum Dual-bank boot fails with stack pointer placed within aliased SRAM2.
12-Sep-2019	12	Added erratum SPI master communication failure at high fPCLK within the specified range
13-Feb-2020	13	Added erratum FLASH_ECCR corrupted upon reset or power-down occurring during Flash memory program or erase operation
23-Apr-2021	14	Added errata: • PC13 signal transitions disturb LSE • Setting GPIO properties of PC13 used as RTC_ALARM open-drain output

Date	Version	Changes
		Modified errata: • Full JTAG configuration without NJTRST pin cannot be used (added *or for an alternate function other than NJTRST*) • Device may remain stuck in LPTIM interrupt when entering Stop mode (*MCU* replaced with *device*) • Device may remain stuck in LPTIM interrupt when clearing event flag (*MCU* replaced with *device* and the description corrected) • DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear (rephrasing in the description and workaround) • Inhibited acquisition in short transfer phase configuration (the second forbidden setting corrected from 111 to 001) • RTC calendar registers are not locked properly (semantic improvement in the description) • Last-received byte loss in reload mode (added information for instances with independent clock)

Contents