Errata sheet

STM32G471xx/473xx/474xx/483xx/484xx device errata

Applicability

This document applies to the part numbers of STM32G471xx/473xx/474xx/483xx/484xx 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 RM0440.

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
STM32G471xx	STM32G471CB, STM32G471CC, STM32G471CE, STM32G471MC, STM32G471ME, STM32G471QB, STM32G471QC, STM32G471QE, STM32G471RB, STM32G471RC, STM32G471RE, STM32G471VB, STM32G471VC, STM32G471VE
STM32G473xx	STM32G473CB, STM32G473CC, STM32G473CE, STM32G473MB, STM32G473MC, STM32G473ME, STM32G473QB, STM32G473QC, STM32G473QE, STM32G473RB, STM32G473RC, STM32G473RE, STM32G473VB, STM32G473VC, STM32G473VE
STM32G474xx	STM32G474CB, STM32G474CC, STM32G474CE, STM32G474MB, STM32G474MC, STM32G474ME, STM32G474QB, STM32G474QC, STM32G474QE, STM32G474RB, STM32G474RC, STM32G474RE, STM32G474VB, STM32G474VC, STM32G474VE
STM32G483xx	STM32G483CE, STM32G483ME, STM32G483QE, STM32G483RE, STM32G483VE
STM32G484xx	STM32G484CE, STM32G484ME, STM32G484QE, STM32G484RE, STM32G484VE

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32G471xx/473xx/474xx/484xx	Z	0x2001

1. *Refer to the device data sheet 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 STM32G471xx/473xx/474xx/483xx/484xx 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. Z
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	Full JTAG configuration without NJTRST pin cannot be used	A
	2.2.2	Data cache might be corrupted during Flash memory read-while-write operation	A
	2.2.3	First double-word of Flash memory corrupted upon reset or power-down while programming	A
	2.2.4	Unstable LSI when it clocks RTC or CSS on LSE	P
DMA	2.3.1	DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	A
DMAMUX	2.4.1	SOFx not asserted when writing into DMAMUX_CFR register	N
	2.4.2	OFx not asserted for trigger event coinciding with last DMAMUX request	N
	2.4.3	OFx not asserted when writing into DMAMUX_RGCFR register	N
	2.4.4	Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event	A
ADC	2.7.1	End of 10/8/6-bit ADC conversion disturbing other ADCs	A
	2.7.2	ADC input channel switching disturbs ongoing conversions	P
	2.7.3	Wrong ADC result if conversion done late after calibration or previous conversion	A
FMC	2.5.1	Dummy read cycles inserted when reading synchronous memories	N
	2.5.2	Wrong data read from a busy NAND memory	A
QUADSPI	2.6.1	QUADSPI cannot be used in indirect read mode when only data phase is activated	P
	2.6.2	QUADSPI_CCR hangs when QUADSPI_CR is cleared	P
	2.6.3	QUADSPI internal timing criticality	A
	2.6.4	Memory-mapped read of last memory byte fails	P
ADC	2.7.2	ADC input channel switching disturbs ongoing conversions	A
TIM	2.8.1	One-pulse mode trigger not detected in master-slave reset + trigger configuration	P

Function	Section	Limitation	Status Rev. Z
LPTIM	2.9.1	MCU may remain stuck in LPTIM interrupt when entering Stop mode	A
	2.9.2	MCU may remain stuck in LPTIM interrupt when clearing event flag	P
RTC and TAMP	2.10.1	Calendar initialization may fail in case of consecutive INIT mode entry	A
	2.10.2	Alarm flag may be repeatedly set when the core is stopped in debug	N
I2C	2.11.1	Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period	P
	2.11.2	Spurious bus error detection in master mode	A
	2.11.3	Spurious master transfer upon own slave address match	P
SPI	2.12.1	BSY bit may stay high when SPI is disabled	A
	2.12.2	BSY bit may stay high at the end of data transfer in slave mode	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.

arm

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

2.1 Core

Errata notice for the Arm® Cortex®-M4 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 Full JTAG configuration without NJTRST pin cannot be used

Description

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

Workaround

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

2.2.2 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();
```

2.2.3 First double-word of Flash memory corrupted upon reset or power-down while programming

Description

Power-down and external reset events occurring during Flash memory program operation may result in zeroing its first double-word (64 bits on the address 0x0800 0000). When this occurs, the boot sequence ends immediately after reset, generating Hard Fault.

In most cases, this corruption is temporary and the correct value is read again after a few milliseconds.

Note: Corruption of Flash memory word on the address accessed in the instant of a reset event occurring during program or erase operation is a normal (specified) behavior.

Workaround

Clone the first Flash memory page contents at another Flash memory location to use as a backup page. In the Hard Fault handler:

1. Compare the first Flash memory double-word content with the value backed up. If different:
   a. Erase the first Flash memory page.
   b. From the backup page, restore the first Flash memory page contents, excluding the first double-word.
   c. From the backup page, restore the first Flash memory double-word.
2. Execute a software reset (by setting the SYSRESETREQ bit), to resume execution.

Note: In the process of firmware development, the first two 32-bit words of the Flash memory are used by the core as program counter (PC) and as stack pointer (SP), respectively, so this 64-bit value as well as the remaining content of the page can vary upon each build.

2.2.4 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.

2.3 DMA

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

Workaround

Do not clear GIFx flags. Instead, use HTIFx, TCIFx, and TEIFx specific event flags and their corresponding clear bits.

2.4 DMAMUX

2.4.1 SOFx not asserted when writing into DMAMUX_CFR 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_CFR 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_CFR register (to set the corresponding CSOFx bit), overrun occurring from another DMAMUX channel operating during that write operation fails to raise its corresponding SOFx flag.

Workaround

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

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

Description

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

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

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

Workaround

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

2.4.3 OFx not asserted when writing into DMAMUX_RGCFR register

Description

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

Workaround

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

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

Description

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

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

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

Workaround

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

2.5 FMC

2.5.1 Dummy read cycles inserted when reading synchronous memories

Description

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

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

Workaround

None.

2.5.2 Wrong data read from a busy NAND memory

Description

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

Workaround

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

2.6 QUADSPI

2.6.1 QUADSPI cannot be used in indirect read mode when only data phase is activated

Description

When the QUADSPI peripheral is configured in indirect read with only the data phase activated (in single, dual, or quad I/O mode), the QUADSPI peripheral hangs and the BUSY flag of the QUADSPI_SR register remains high. An abort must be performed to reset the BUSY flag and exit the hanging state.

Workaround

Insert a dummy phase with at least two dummy cycles.

2.6.2 QUADSPI_CCR hangs when QUADSPI_CR is cleared

Description

Writing 0x0000 0000 to the QUADSPI_CCR register causes the QUADSPI peripheral to hang while the BUSY flag of the QUADSPI_SR register remains set. Even an abort does not allow exiting this status.

Workaround

Clear then set the EN bit of the QUADSPI_CR register.

2.6.3 QUADSPI internal timing criticality

Description

The timing of some internal signals of the QUADSPI peripheral is critical. At certain conditions, this can lead to a general failure of the peripheral. As these conditions cannot be exactly determined, it is recommended to systematically apply the workaround as described.

Workaround

The code below have to be executed upon reset and upon switching from memory-mapped to any other mode:

```
// Save QSPI_CR and QSPI_CCR values if necessary
QSPI->QSPI_CR = 0; // ensure that prescaling factor is not at maximum, and  disable the perip
heral
while(QSPI->QSPI_SR & 0x20){}; // wait for BUSY flag to fall if not already low
QSPI->QSPI_CR = 0xFF000001; // set maximum prescaling factor, and  enable the peripheral
QSPI->QSPI_CCR = 0x20000000; // activate the free-running clock
QSPI->QSPI_CCR = 0x20000000; // repeat the previous instruction to prevent a back-to-back dis
able

// The following command must complete less than 127 kernel clocks after the first write to t
he QSPI_CCR register
QSPI->QSPI_CR = 0; // disable QSPI
while(QSPI->QSPI_SR & 0x20){}; // wait for busy to fall

// Restore CR and CCR values if necessary
```

For the worakround to be effective, it is important to complete the disable instruction less than 127 kernel clock pulses after the first write to the QSPI_CCR register.

2.6.4 Memory-mapped read of last memory byte fails

Description

Regardless of the number of I/O lines used (1, 2 or 4), a memory-mapped read of the last byte of the memory region defined through the FSIZE[4:0] bitfield of the QUADSPI_DCR register always yields 0x00, whatever the memory byte content is. A repeated attempt to read that last byte causes the AXI bus to stall.

Workaround

Apply one of the following measures:

- Avoid reading the last byte of the memory region defined through FSIZE, for example by taking margin in FSIZE bitfield setting.
- If the last byte is read, ignore its value and abort the ongoing process so as to prevent the AXI bus from stalling.
- For reading the last byte of the memory region defined through FSIZE, use indirect read.

2.7 ADC

2.7.1 End of 10/8/6-bit ADC conversion disturbing other ADCs

Description

The end-of-conversion event of an ADC instance set to 10-, 8-, or 6-bit resolution disturbs the reference voltage, causing a conversion error to any other ADC instances with conversion in progress.

Workaround

Either set the ADCs to 12-bit resolution, or, with concurrent operation of multiple ADCs, avoid the end of conversion of one ADC to occur during the conversion phase of another ADC. For example, set them all to the same resolution and the same sampling duration, and start their sampling phase at the same time.

2.7.2 ADC input channel switching disturbs ongoing conversions

Description

A switch of the ADC input multiplexer to a different input channel disturbs all ADC instances with conversion in progress, thus negatively impacting their DNL. As the device operates the multiplexer during the ADC conversion, either to select the following input of a scan sequence or to open all multiplexer inputs (idle mode), this disturbance source may affect all operating modes except continuous and except bulb sampling.

The DNL impact depends on the input channel and it increases with increasing number of concurrently converting ADCs, the worst case (a few LSBs) being when all the ADCs on the device operate synchronously.

Workaround

When using a single ADC input channel, activate bulb sampling. This prevents the opening of all multiplexer inputs during the conversion cycle.

In scan conversion mode, set the input scan sequence such that two successive conversions are performed on each input. Keep the result of the first and discard the second, as only the second is affected by the input channel switch disturbance.

2.7.3 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.

2.8 TIM

2.8.1 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 to decrease the timer latency to external trigger events.

2.9 LPTIM

2.9.1 MCU 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 MCU from Stop mode may be frozen in active state. Consequently, when trying to enter Stop mode, this limitation prevents the MCU 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.

2.9.2 MCU 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 by writing the LPTIM_ICR bit in the LPTIM_ISR 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 at ‘1’.

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 MCU 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 .

2.10 RTC and TAMP

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

Description

If the INIT bit of the RTC_ICSR register is set between one and two RTCCLK cycles after being cleared, the INITF flag is set immediately instead of waiting for synchronization delay (which should be between one and two RTCCLK cycles), and the initialization of registers may fail. Depending on the INIT bit clearing and setting instants versus the RTCCLK edges, it can happen that, after being immediately set, the INITF flag is cleared during one RTCCLK period then set again. As writes to calendar registers are ignored when INITF is low, a write occurring during this critical period might result in the corruption of one or more calendar registers.

Workaround

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

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

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

Description

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

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

Workaround

None.

2.11 I2C

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

Description

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

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

The MCU 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.

2.11.2 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 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.

2.11.3 Spurious master transfer upon own slave address match

Description

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

2.12 SPI

2.12.1 BSY bit may stay high when SPI is disabled

Description

The BSY flag may remain high upon disabling the SPI while operating in:

- master transmit mode and the TXE flag is low (data register full).
- master receive-only mode (simplex receive or half-duplex bidirectional receive phase) and an SCK strobing edge has not occurred since the transition of the RXNE flag from low to high.
- slave mode and NSS signal is removed during the communication.

Workaround

When the SPI operates in:

- master transmit mode, disable the SPI when TXE = 1 and BSY = 0.
- master receive-only mode, ignore the BSY flag.
- slave mode, do not remove the NSS signal during the communication.

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

Description

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

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

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

Workaround

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

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

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

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

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

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

Revision history

Table 4. **Document revision history**

Date	Version	Changes
19-Apr-2019	1	Initial release.

Contents