STM32WB55Cx STM32WB55Rx STM32WB55Vx

Errata sheet

STM32WB55Cx/Rx/Vx device errata

Applicability

This document applies to the part numbers of STM32WB55Cx/Rx/Vx 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 RM0434.

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
STM32WB55Cx	STM32WB55CC, STM32WB55CE, STM32WB55CG
STM32WB55Rx	STM32WB55RC, STM32WB55RE, STM32WB55RG
STM32WB55Vx	STM32WB55VC, STM32WB55VE, STM32WB55VG

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32WB55Cx/Rx/Vx	Y	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 STM32WB55Cx/Rx/Vx 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. Y
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	LSI2 not usable as RF low-power clock	P
	2.2.2	Unstable LSI1 when it clocks RTC or CSS on LSE	P
	2.2.3	Internal voltage reference corrupted upon Stop mode entry with temperature sensing enabled	A
	2.2.4	First double-word of Flash memory corrupted upon reset or power-down while programming	A
	2.2.5	Full JTAG configuration without NJTRST pin cannot be used	A
	2.2.6	Auto-incrementing feature of the debug port cannot span more than 1 Kbyte	A
	2.2.7	PC13 signal transitions disturb LSE	N
	2.2.8	Wrong DMAMUX synchronization and trigger input connections to EXTI	A
	2.2.9	Incomplete Stop 2 mode entry after a wakeup from debug upon EXTI line 48 event	A
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
QUADSPI	2.5.1	First nibble of data not written after dummy phase	A
	2.5.2	Wrong data from memory-mapped read after an indirect mode operation	A
	2.5.3	Memory-mapped read operations may fail when timeout counter is enabled	P
	2.5.4	Memory-mapped access in indirect mode clearing QUADSPI_AR register	P
ADC	2.6.1	Wrong ADC result if conversion done late after calibration or previous conversion	A

Function	Section	Limitation	Status Rev. Y
LPTIM	2.7.1	MCU may remain stuck in LPTIM interrupt when entering Stop mode	A
	2.7.2	MCU may remain stuck in LPTIM interrupt when clearing event flag	P
RTC and TAMP	2.8.1	RTC interrupt can be masked by another RTC interrupt	A
	2.8.2	Calendar initialization may fail in case of consecutive INIT mode entry	A
	2.8.3	Alarm flag may be repeatedly set when the core is stopped in debug	N
I2C	2.9.1	Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period	P
	2.9.2	Spurious bus error detection in master mode	A
	2.9.3	Spurious master transfer upon own slave address match	P
	2.9.4	START bit is cleared upon setting ADDRCF, not upon address match	P
SPI	2.10.1	BSY bit may stay high when SPI is disabled	A
	2.10.2	BSY bit may stay high at the end of data transfer in slave mode	A
	2.10.3	Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters	A
USB	2.11.1	Possible packet memory overrun/underrun (PMAOVR) with APB frequency below 9.3 MHz	N

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 FPU 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).
- Disable lazy context save of floating point state by clearing LSPEN to 0 (bit 30 of the FPCCR at address 0xE000EF34).

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 LSI2 not usable as RF low-power clock

Description

Due to its calibration mechanism and peaks of jitter, the LSI2 clock exceeds 500 ppm maximum frequency deviation specified in the Bluetooth® Low Energy standard for low-speed clock.

Workaround

To clock the Bluetooth® peripheral, use the LSE oscillator, instead of the LSI2.

2.2.2 Unstable LSI1 when it clocks RTC or CSS on LSE

Description

The LSI1 clock can become unstable (duty cycle different from 50 %) and its maximum frequency can become significantly higher than 32 kHz, when:

- LSI1 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 LSI1 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.

2.2.3 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 allows consuming less power during Stop mode.

2.2.4 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.

2.2.5 Full JTAG configuration without NJTRST pin cannot be used

Description

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

Workaround

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

2.2.6 Auto-incrementing feature of the debug port cannot span more than 1 Kbyte

Description

The address auto-increment function of the AP2AHB access ports is limited to 1 Kbyte of contiguous data.

Workaround

To write a contiguous chunk of data larger than 1 Kbyte to an AP2AHB access port, split and write it in blocks not exceeding 1 Kbyte each.

2.2.7 PC13 signal transitions disturb LSE

Description

The PC13 port toggling disturbs the LSE clock. It may not be usable when LSE is used.

Workaround

None.

2.2.8 Wrong DMAMUX synchronization and trigger input connections to EXTI

Description

By error, synchronization and trigger inputs of the DMAMUX peripheral are connected to interrupt output lines of the EXTI block, instead of being connected to its SYSCFG multiplexer output lines.

The EXTI interrupt lines exhibit a rising-edge transition upon each active transition (rising, falling or both) of corresponding GPIOs, as defined in the EXTI_RTSR1 and EXTI_FTSR registers.

As a consequence, the falling active edge option of the DMAMUX synchronization and trigger inputs is unusable because falling edges on these inputs do not occur upon GPIO events but upon clearing the EXTI interrupt pending flags (by setting the corresponding PIF bits of the EXTI_PR1 register).

Workaround

For the DMAMUX synchronization and trigger events to occur upon determined rising or/and falling edge of the corresponding GPIOs:

- Set the desired active edge polarities of the corresponding GPIOs through the EXTI_RTSRx and EXTI_FTSRx registers.
- Set the active edge polarity to rising for all corresponding DMAMUX input lines, through the SPOL bits of the DMAMUX_CxCR register (for synchronization inputs) and the GPOL bits of the DMAMUX_RGxCR register (for trigger inputs).
- Ensure that EXTI interrupt pending flags corresponding to the GPIOs used for DMAMUX inputs are cleared in the EXTI interrupt service routine.

Note: This can be ensured if using the HAL_GPIO_IrqHandler function provided by STMicroelectronics.

2.2.9 Incomplete Stop 2 mode entry after a wakeup from debug upon EXTI line 48 event

Description

With the JTAG debugger enabled on GPIO pins and after a wakeup from debug triggered by an event on EXTI line 48 (CDBGPWRUPREQ), the device may enter in a state in which attempts to enter Stop 2 mode are not fully effective: The CPUs duly enter their respective CStop modes and are able to wake up so the software execution is not disturbed. However, the system does not transit to a low-power state and thus keeps a power consumption higher than expected.

Workaround

Before entering Stop 2 mode, mask *wakeup with interrupt request* from EXTI line 48 (CDBGPWRUPREQ) in both EXTI_IMR2 and EXTI_C2IMR2 registers.

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 QUADSPI

2.5.1 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.

2.5.2 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.

2.5.3 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.

2.5.4 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

2.6 ADC

2.6.1 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 LPTIM

2.7.1 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_APByRSTRz register.

2.7.2 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 STM32Cube™.

2.8 RTC and TAMP

2.8.1 RTC interrupt can be masked by another RTC interrupt

Description

One RTC interrupt can mask another RTC interrupt if both share the same EXTI configurable line, such as the RTC Alarm A and Alarm B, of which the event flags are OR-de 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).

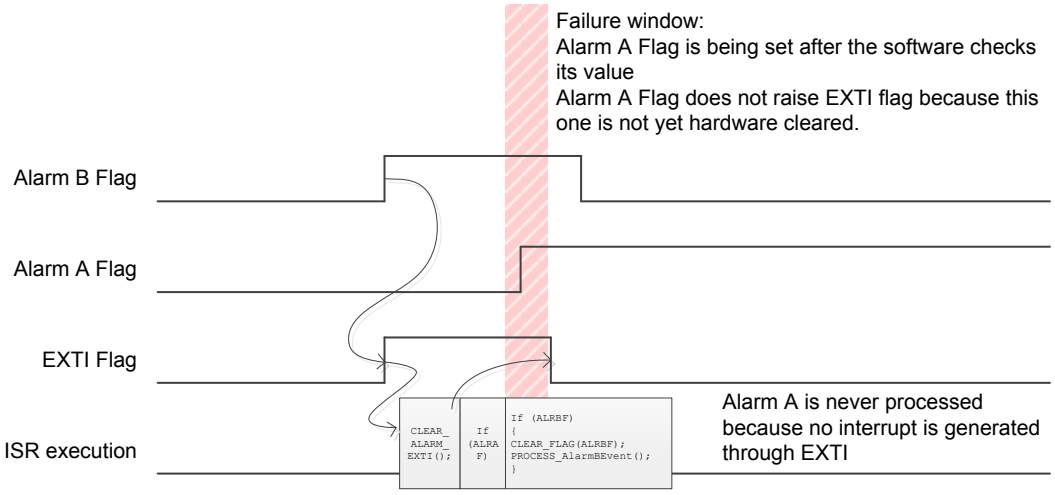
The following code example and figure illustrate the failure mechanism: The Alarm A event is lost (fails to generate interrupt) as it occurs in the failure window, that is, after checking the Alarm A event flag but before the effective clear of the EXTI interrupt flag by hardware. The effective clear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

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

Description

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

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

Workaround

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

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

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

Description

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

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

Workaround

None.

2.9 I2C

2.9.1 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 I^2C-bus SDA line when $t_{SU;DAT}$ is smaller than one I2C kernel clock (I^2C-bus peripheral clock) period: the previous SDA value is sampled instead of the current one. This can result in a wrong receipt of slave address, data byte, or acknowledge bit.

Workaround

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

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

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

2.9.2 Spurious bus error detection in master mode

Description

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

Workaround

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

2.9.3 Spurious master transfer upon own slave address match

Description

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

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

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

Workaround

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

Normal mode (SBC = 0):

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

Slave byte control mode (SBC = 1):

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

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

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

2.9.4 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.

2.10 SPI

2.10.1 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.

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

Description

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

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

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

Workaround

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

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

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

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

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

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

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

Description

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

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

Workaround

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

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

2.11 USB

2.11.1 Possible packet memory overrun/underrun (PMAOVR) with APB frequency below 9.3 MHz

Description

The USB interface is expected to operate correctly with an APB frequency as low as 8 MHz. However, in the 8 to 9.3 MHz APB frequency range, some packet buffer memory (PBM) overrun/underrun issues may be observed.

- Overrun for OUT transactions in the 8 to 9.3 MHz APB frequency range:

  The USB peripheral may be late to store the received data into the PBM before the next byte is received on the USB (PBM overrun). In this case the USB cell detects an internal error condition, discards the last received byte, stops writing into the PBM, sends no acknowledge (so that the Host is forced to retry the transaction), and informs the application by setting the PMAOVR flag/interrupt.
- Underrun for IN transactions in the 8 to 8.7 MHz APB frequency range:

  The USB peripheral was not in time to read from the PBM the next byte to be transmitted before the transmission of the previous one is completed on the USB. In this case the USB cell detects an internal error condition, stops reading from PBM, generates a bit stuffing error on the USB (so that the Host is forced to retry the transaction), and informs the application by setting the PMAOVR flag/interrupt.

Workaround

None.

Revision history

Table 4. **Document revision history**

Date	Version	Changes
21-Feb-2019	4	First public release.

Contents