STM32WL54xx, STM32WL55xx device errata

Applicability

This document applies to the part numbers of STM32WL54xx, STM32WL55xx 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 RM0453.

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
STM32WL54xx	STM32WL54CC, STM32WL54JC, STM32WL54UC
STM32WL55xx	STM32WL55CC, STM32WL55JC, STM32WL55UC

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32WL55xx	Z	0x1001
STM32WL54xx	Z	0x1001

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

1 Summary of device errata

The following table gives a quick reference to the STM32WL54xx, STM32WL55xx 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 1	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	2.3.1	Wrong DMAMUX synchronization and trigger input connections to EXTI	A
	2.3.2	Perpetual CPU2 boot upon illegal access	A
	2.3.3	Overwriting with all zeros a Flash memory location previously programmed with all ones fails	N
	2.3.4	Option byte loading failure at high MSI system clock frequency	A
	2.3.5	A system reset occurs when nRST_SHDW is set and nRST_STDBY is cleared and Shutdown mode is entered	A
	2.3.6	FLASH_ECCR corrupted upon reset or power-down occurring during Flash memory program or erase operation	A
	2.3.7	Voltage drop on the 1.2 V regulated supply when switching MSI to 48 MHz	A
	2.3.8	Sensitivity affected by HSE activation in high bandwidth channel	A
	2.3.9	Sensitivity degradation in LNA boosted mode	A
	2.3.10	Systick trigger in debug emulation generates hard fault	A
	2.3.11	Debug HALT command in debug emulation generates a hard fault	A
	2.3.12	Potential deadlock condition on wakeup from a lower-power (LP) mode	A
	2.3.13	JTAG cannot be used without the JTAG NRST pin	N
	2.3.14	Flash PCROP is not operating properly	N
	2.3.15	Unexpected C2DS flag starting from 2nd Standby wakeup	N
	2.3.16	CPU2 boot on system reset after illegal access detection	A
	2.3.17	Peripherals freeze on debug HALT command while CPU1 is in Stop mode and CPU2 is in Run mode	A
TIM	2.4.1	One-pulse mode trigger not detected in master-slave reset + trigger configuration	P
	2.4.2	Consecutive compare event missed in specific conditions	N
	2.4.3	Output compare clear not working with external counter reset	P

Function	Section	Limitation	Status Rev. Z
LPTIM	2.5.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	A
	2.5.2	ARRM and CMPM flags are not set when APB clock is slower than kernel clock	P
	2.5.3	Device may remain stuck in LPTIM interrupt when clearing event flag	P
RTC and TAMP	2.6.1	Alarm flag may be repeatedly set when the core is stopped in debug	N
	2.6.2	A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF	N
	2.6.3	REFCKON write protection associated to INIT KEY instead of CAL KEY	A
	2.6.4	Tamper flag not set on LSE failure detection	N
I2C	2.7.1	Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period	P
	2.7.2	Spurious bus error detection in master mode	A
	2.7.3	OVR flag not set in underrun condition	N
	2.7.4	Transmission stalled after first byte transfer	A
USART	2.8.1	Anticipated end-of-transmission signaling in SPI slave mode	A
	2.8.2	Data corruption due to noisy receive line	N
	2.8.3	DMA stream locked when transferring data to/from USART	A

2 Description of device errata

The following sections describe limitations of the applicable devices with Arm® core and provide workarounds if available. They are grouped by device functions.

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

2.1 Core 1

Reference manual and 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).
- 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 Core 2

Errata notice for the Arm® Cortex®-M0+ core revision r0p1 is available from http://infocenter.arm.com.

2.3 System

2.3.1 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_RTSRx and EXTI_FTSRx 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_PRx 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.

2.3.2 Perpetual CPU2 boot upon illegal access

Description

After handling an illegal access, the CPU2 re-boots upon each low-power mode entry as long as the C2BOOT bit remains set, whereas it should only re-boot upon the first low-power mode entry.

Workaround

In the ILAC handler of CPU2, execute the following sequence:

```
IWDG->KR = 0x0000CCCCu;
IWDG->RLR = 0x01u;
PWR->CR4 &= ~PWR_CR4_C2BOOT;
SET_BIT(SCB->SCR, ((uint32_t)SCB_SCR_SLEEPDEEP_Msk));
__WFI();
```

2.3.3 Overwriting with all zeros a Flash memory location previously programmed with all ones fails

Description

Any attempt to re-program with all zeros (0x0000 0000 0000 0000) a Flash memory location previously programmed with 0xFFFF FFFF FFFF FFFF fails and the PROGERR flag of the FLASH_SR register is set.

Workaround

None.

2.3.4 Option byte loading failure at high MSI system clock frequency

Description

The option bytes are not loaded correctly upon setting the OBL_LAUNCH bit of the FLASH_CR register when the frequency of MSI oscillator used as system clock source is higher than 16 MHz.

Workaround

Before loading the option bytes by setting the OBL_LAUNCH bit of the FLASH_CR register, either change the system clock source to other than MSI oscillator, or set the MSI clock frequency to less than 16 MHz.

2.3.5 A system reset occurs when nRST_SHDW is set and nRST_STDBY is cleared and Shutdown mode is entered

Description

When the following configuration is selected:

- nRST_SHDW is set
- nRST_STDBY is cleared,

a system reset is generated when Shutdown mode is entered.

Workaround

The only valid configuration to avoid a reset after entering into Shutdown mode is the following:

- nRST_SHDW is set
- nRST_STDBY is set.

2.3.6 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.

2.3.7 Voltage drop on the 1.2 V regulated supply when switching MSI to 48 MHz

Description

A voltage drop to 1.08 V may occur on the 1.2 V regulated supply when the MSI frequency is changed as follows:

- from MSI at 400 KHz to MSI at 24 MHz and above
- from MSI at 1 MHZ to MSI at 48 MHz

As a result, the voltage drop may cause a CPU hard fault.

Workaround

To ensure there is no impact on the 1.2 V supply, introduce an intermediate MSI frequency change step as follows:

- Change the MSI frequency range from 400 KHz to 12 MHz, then to 24 MHz and above, as illustrated in these examples:
  - MSI@400 KHz → MSI@12 MHz → MSI@24 MHz
  - MSI@400 KHz → MSI@12 MHz → MSI@48 MHz.
- Change the MSI frequency range from 1 MHz to 12 MHz, then to 48 MHz, as illustrated in this example: MSI@400 KHz → MSI@12 MHz → MSI@48 MHz.

2.3.8 Sensitivity affected by HSE activation in high bandwidth channel

Description

The sensitivity of the high bandwidth channels (HB) EU864 and US928 is affected when HSE is used as system clock. HSE cannot be used as system clock when channels EU864 or US928 are used.

Workaround

Use a different clock source for the system, for example MSI or HSI with or without PLL.

2.3.9 Sensitivity degradation in LNA boosted mode

Description

Reduced performance is observed in some specific LNA boost mode use cases. There is a potential loss of sensitivity of approximately 1.5 dB versus specification for LoRa and GFSK modulation at 915 MHz.

Workaround

None.

2.3.10 Systick trigger in debug emulation generates hard fault

Description

When the CPU enters a *deepsleep state* and debug emulation is active, the CPU systick is still running and able to trigger interrupts. The CPU is woken up and fetches code from the system Flash memory. As the Flash memory is not active, the CPU starts fetching code from a resource which is not available and receives random data from the bus, causing a hard fault.

When the debug emulation is active and the CPU enters its *deepsleep state*, the system exhibits unpredictable behavior due to the systick activity.

Workaround

Always disable systick before the CPU enters its *deepsleep state*. In this case we consistently prevents any interrupt triggering and any possibility of a hard fault.

2.3.11 Debug HALT command in debug emulation generates a hard fault

Description

When the CPU is in *deepsleep state* with active debug emulation and a "debug HALT - debug RUN" command sequence is submitted, the CPU wakes up and starts fetching data from Flash memory. As Flash memory is unavailable, the CPU receives random data from the bus which causes a hard fault.

Debug cannot be performed on the system when the CPU is in *deepsleep state* and debug emulation is active.

Workaround

Change the debug sequence to perform a detach/attach procedure before any "HALT/RUN". Also enable the CDBGPWRUP wakeup on the AIEC before the CPU goes into *deepsleep state*.

2.3.12 Potential deadlock condition on wakeup from a lower-power (LP) mode

Description

Following multiple wakeups from Stop1, Stop2 or Standby, the power supply management block does not restart properly. This occurs on a specific timing of the wakeup event and Run duration.

The issue may occur in the event of an asynchronous wakeup when the MCU remains in Run for less than 60 µs after a wakeup.

Workaround

There are two possible workarounds, depending on the conditions:

- General workaround (valid also if SMPS is used before entering wakeup): wait for the LDO to reach a stable condition before entering LP mode.

  A software implementation solution is to check that the LDORDY bit of the PWR_SR2 register is set before entering LP mode.
- LDO use case only (SMPS was not used before entering wakeup): the deadlock condition, which can potentially occur at wakeup from LP mode, can be solved by generating a pulse on the SMPSEN bit of the PWR_CR5 register controlling the SMPS activation.

  A software implementation solution is to toggle the SMPSEN bit of the PWR_CR5 register at each wakeup from LP mode.

2.3.13 JTAG cannot be used without the JTAG NRST pin

Description

When CPU1 is in a Deepsleep state with active debug emulation and CPU2 is stopped as well, and a `debug HALT - debug RUN` command sequence is submitted, CPU1 wakes up and starts to fetch from Flash memory. In this condition, the Flash memory is off, CPU1 retrieves random information from the bus which causes a HardFault.

By default after reset, PB4 is configured as alternate function for JTAG NRESET function. If this pin is reused in another configuration (GPIO, analog or alternate function for another function), the product jtag nreset internal signal is forced to 0.

Workaround

None.

2.3.14 Flash PCROP is not operating properly

Description

The PCROP protection on the Flash memory area does not work as expected.

Workaround

None.

2.3.15 Unexpected C2DS flag starting from 2nd Standby wakeup

Description

If the system enters Standby while CPU2 is not booted (C2BOOT is cleared) and a wakeup request directed to CPU2 is received and then redirected to CPU1, the C2DS flag indicating the CPU2 Deepsleep state behaves in the following manner:

- On 1st wakeup from Standby, C2DS is set as expected.
- Starting from 2nd wakeup from Standby, C2DS is cleared, while it should be set.

Workaround

None.

2.3.16 CPU2 boot on system reset after illegal access detection

Description

After an illegal access is detected with CPU2 active where C2BOOT is set and treated through ILAC handling procedure, on next system reset, CPU2 boots without any trigger generated by CPU1.

The process can be modelled as follows:

1. CPU1 runs through the boot sequence and boots CPU2 normally through C2BOOT.
2. An illegal access is generated from either: CPU1 or CPU2, is normally detected by CPU2.
3. At next system reset, CPU2 boots immediately.

Workaround

- Perform a power-on reset to restart from a reliable state.
- Handle the unexpected CPU2 boot by software.

2.3.17 Peripherals freeze on debug HALT command while CPU1 is in Stop mode and CPU2 is in Run mode

Description

When CPU1 is in Stop mode, CPU2 is in Run mode, debug emulation is active, and a `debug HALT - debug RUN` command sequence is submitted, CPU1 wakes up and starts, but peripherals exclusively associated to CPU1 are not clocked by RCC.

No debug can be performed on the system when CPU1 is in Stop mode, CPU2 is in Run mode and debug emulation is active.

Workaround

Change the debug sequence to perform a detach/attach operation before the HALT/RUN and set the CDBGPWRUPREQ wakeup on the AIEC before CPU1 goes into Deepsleep.

2.4 TIM

2.4.1 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.

2.4.2 Consecutive compare event missed in specific conditions

Description

Every match of the counter (CNT) value with the compare register (CCR) value is expected to trigger a compare event. However, if such matches occur in two consecutive counter clock cycles (as consequence of the CCR value change between the two cycles), the second compare event is missed for the following CCR value changes:

- in edge-aligned mode, from ARR to 0:
  - first compare event: CNT = CCR = ARR
  - second (missed) compare event: CNT = CCR = 0
- in center-aligned mode while up-counting, from ARR-1 to ARR (possibly a new ARR value if the period is also changed) at the crest (that is, when TIMx_RCR = 0):
  - first compare event: CNT = CCR = (ARR-1)
  - second (missed) compare event: CNT = CCR = ARR
- in center-aligned mode while down-counting, from 1 to 0 at the valley (that is, when TIMx_RCR = 0):
  - first compare event: CNT = CCR = 1
  - second (missed) compare event: CNT = CCR = 0

This typically corresponds to an abrupt change of compare value aiming at creating a timer clock single-cycle-wide pulse in toggle mode.

As a consequence:

- In toggle mode, the output only toggles once per counter period (squared waveform), whereas it is expected to toggle twice within two consecutive counter cycles (and so exhibit a short pulse per counter period).
- In center mode, the compare interrupt flag does note rise and the interrupt is not generated.

Note: The timer output operates as expected in modes other than the toggle mode.

Workaround

None.

2.4.3 Output compare clear not working with external counter reset

Description

The output compare clear event (ocref_clr) is not correctly generated when the timer is configured in the following slave modes: Reset mode, Combined reset + trigger mode, and Combined gated + reset mode.

The PWM output remains inactive during one extra PWM cycle if the following sequence occurs:

1. The output is cleared by the ocref_clr event.
2. The timer reset occurs before the programmed compare event.

Workaround

Apply one of the following measures:

- Use BKIN (or BKIN2 if available) input for clearing the output, selecting the Automatic output enable mode (AOE = 1).
- Mask the timer reset during the PWM ON time to prevent it from occurring before the compare event (for example with a spare timer compare channel open-drain output connected with the reset signal, pulling the timer reset line down).

2.5 LPTIM

2.5.1 Device may remain stuck in LPTIM interrupt when entering Stop mode

Description

This limitation occurs when disabling the low-power timer (LPTIM).

When the user application clears the ENABLE bit in the LPTIM_CR register within a small time window around one LPTIM interrupt occurrence, then the LPTIM interrupt signal used to wake up the device from Stop mode may be frozen in active state. Consequently, when trying to enter Stop mode, this limitation prevents the device from entering low-power mode and the firmware remains stuck in the LPTIM interrupt routine.

This limitation applies to all Stop modes and to all instances of the LPTIM. Note that the occurrence of this issue is very low.

Workaround

In order to disable a low power timer (LPTIMx) peripheral, do not clear its ENABLE bit in its respective LPTIM_CR register. Instead, reset the whole LPTIMx peripheral via the RCC controller by setting and resetting its respective LPTIMxRST bit in RCC_APByRSTRz register.

2.5.2 ARRM and CMPM flags are not set when APB clock is slower than kernel clock

Description

When LPTIM is configured in one shot mode and APB clock is lower than kernel clock, there is a chance that ARRM and CMPM flags are not set at the end of the counting cycle defined by the repetition value REP[7:0]. This issue can only occur when the repetition counter is configured with an odd repetition value.

Workaround

To avoid this issue the following formula must be respected:

{ARR, CMP} ≥ KER_CLK / (2* APB_CLK),

where APB_CLK is the LPTIM APB clock frequency, and KER_CLK is the LPTIM kernel clock frequency. ARR and CMP are expressed in decimal value.

Example: The following example illustrates a configuration where the issue can occur:

- APB clock source (MSI) = 1 MHz , Kernel clock source (HSI) = 16 MHz
- Repetition counter is set with REP[7:0] = 0x3 (odd value)

The above example is subject to issue, unless the user respects:

{CMP, ARR} ≥ 16 MHz / (2 * 1 MHz)

→ ARR must be ≥ 8 and CMP must be ≥ 8

Note: REP set to 0x3 means that effective repetition is REP+1 (= 4) but the user must consider the parity of the value loaded in LPTIM_RCR register (=3, odd) to assess the risk of issue.

2.5.3 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.

2.6 RTC and TAMP

2.6.1 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.

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

Description

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

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

Workaround

None.

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

Description

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

Workaround

Unlock the INIT KEY before writing REFCKON.

2.6.4 Tamper flag not set on LSE failure detection

Description

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

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

Workaround

None.

2.7 I2C

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

Description

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

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

The device does not correctly sample the I^2C-bus SDA line when $t_{SU;DAT}$ is smaller than one I2C kernel clock (I^2C-bus peripheral clock) period: the previous SDA value is sampled instead of the current one. This can result in a wrong receipt of slave address, data byte, or acknowledge bit.

Workaround

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

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

2.7.2 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.

2.7.3 OVR flag not set in underrun condition

Description

In slave transmission with clock stretching disabled (NOSTRETCH = 1 in the I2C_CR1 register), an underrun condition occurs if the current byte transmission is completed on the I2C bus, and the next data is not yet written in the TXDATA[7:0] bitfield. In this condition, the device is expected to set the OVR flag of the I2C_ISR register and send 0xFF on the bus.

However, if the I2C_TXDR is written within the interval between two I2C kernel clock cycles before and three APB clock cycles after the start of the next data transmission, the OVR flag is not set, although the transmitted value is 0xFF.

Workaround

None.

2.7.4 Transmission stalled after first byte transfer

Description

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

Workaround

Apply one of the following measures:

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

2.8 USART

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

Description

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

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

Workaround

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

2.8.2 Data corruption due to noisy receive line

Description

In UART mode with oversampling by 8 or 16 and with 1 or 2 stop bits, the received data may be corrupted if a glitch to zero shorter than the half-bit occurs on the receive line within the second half of the stop bit.

Workaround

None.

2.8.3 DMA stream locked when transferring data to/from USART

Description

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

Workaround

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

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

Revision history

Table 4. **Document revision history**

Date	Version	Changes
04-Nov-2020	1	Initial release.

Contents

1 **Summary of device errata** 2

2 **Description of device errata** 4

2.1 Core 1 4

2.1.1 Interrupted loads to SP can cause erroneous behavior 4

2.1.2 VDIV or VSQRT instructions might not complete correctly when very short ISRs are used . 4

2.1.3 Store immediate overlapping exception return operation might vector to incorrect interrupt 5

2.2 Core 2 6

2.3 System 6

2.3.1 Wrong DMAMUX synchronization and trigger input connections to EXTI 6

2.3.2 Perpetual CPU2 boot upon illegal access 7

2.3.3 Overwriting with all zeros a Flash memory location previously programmed with all ones fails 7

2.3.4 Option byte loading failure at high MSI system clock frequency 7

2.3.5 A system reset occurs when nRST_SHDW is set and nRST_STDBY is cleared and Shutdown mode is entered 7

2.3.6 FLASH_ECCR corrupted upon reset or power-down occurring during Flash memory program or erase operation 8

2.3.7 Voltage drop on the 1.2 V regulated supply when switching MSI to 48 MHz 8

2.3.8 Sensitivity affected by HSE activation in high bandwidth channel 8

2.3.9 Sensitivity degradation in LNA boosted mode 8

2.3.10 Systick trigger in debug emulation generates hard fault 9

2.3.11 Debug HALT command in debug emulation generates a hard fault 9

2.3.12 Potential deadlock condition on wakeup from a lower-power (LP) mode 9

2.3.13 JTAG cannot be used without the JTAG NRST pin 9

2.3.14 Flash PCROP is not operating properly 10

2.3.15 Unexpected C2DS flag starting from 2nd Standby wakeup 10

2.3.16 CPU2 boot on system reset after illegal access detection 10

2.3.17 Peripherals freeze on debug HALT command while CPU1 is in Stop mode and CPU2 is in Run mode 10

2.4 TIM 11

2.4.1 One-pulse mode trigger not detected in master-slave reset + trigger configuration 11

2.4.2 Consecutive compare event missed in specific conditions 11

2.4.3 Output compare clear not working with external counter reset . . . 12
2.5 LPTIM . . . 12
2.5.1 Device may remain stuck in LPTIM interrupt when entering Stop mode . . . 12
2.5.2 ARRM and CMPM flags are not set when APB clock is slower than kernel clock . . . 12
2.5.3 Device may remain stuck in LPTIM interrupt when clearing event flag . . . 13
2.6 RTC and TAMP . . . 13
2.6.1 Alarm flag may be repeatedly set when the core is stopped in debug . . . 13
2.6.2 A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF . . . 13
2.6.3 REFCKON write protection associated to INIT KEY instead of CAL KEY . . . 14
2.6.4 Tamper flag not set on LSE failure detection . . . 14
2.7 I2C . . . 14
2.7.1 Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period . . . 14
2.7.2 Spurious bus error detection in master mode . . . 15
2.7.3 OVR flag not set in underrun condition . . . 15
2.7.4 Transmission stalled after first byte transfer . . . 15
2.8 USART . . . 15
2.8.1 Anticipated end-of-transmission signaling in SPI slave mode . . . 15
2.8.2 Data corruption due to noisy receive line . . . 16
2.8.3 DMA stream locked when transferring data to/from USART . . . 16
Revision history . . . 17