STM32F410x8 and STM32F410xB

Errata sheet

STM32F410x8 and STM32F410xB device errata

Applicability

This document applies to the part numbers of STM32F410x8 and STM32F410xB 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 RM0401.

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
STM32F410x8	STM32F410T8, STM32F410C8, STM32F410R8
STM32F410xB	STM32F410TB, STM32F410CB, STM32F410RB

Table 2. **Device variants**

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32F410x8, STM32F410xB	A	0x1000
	1	

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

1 Summary of device errata

The following table gives a quick reference to the STM32F410x8 and STM32F410xB device limitations and their status:

A = limitation present, workaround available

N = limitation present, no workaround available

P = limitation present, partial workaround available

“-” = limitation absent

Applicability of a workaround may depend on specific conditions of target application. Adoption of a workaround may cause restrictions to target application. Workaround for a limitation is deemed partial if it only reduces the rate of occurrence and/or consequences of the limitation, or if it is fully effective for only a subset of instances on the device or in only a subset of operating modes, of the function concerned.

Table 3. Summary of device limitations

Function	Section	Limitation	Status	
			Rev. A	Rev. 1
Core	2.1.1	Interrupted loads to SP can cause erroneous behavior	A	A
	2.1.2	VDIV or VSQRT instructions might not complete correctly when very short ISRs are used	A	A
	2.1.3	Store immediate overlapping exception return operation might vector to incorrect interrupt	A	A
System	2.2.1	Debugging Stop mode and SysTick timer	A	A
	2.2.2	Debugging Stop mode with WFE entry	A	A
	2.2.3	Debugging Sleep/Stop mode with WFE/WFI entry	A	A
	2.2.4	Wake-up sequence from Standby mode when using more than one wake-up source	A	A
	2.2.5	Full JTAG configuration without NJTRST pin cannot be used	A	A
	2.2.6	MPU attribute to RTC and IWDG registers incorrectly managed	A	A
	2.2.7	Delay after an RCC peripheral clock enabling	A	A
	2.2.8	Internal noise impacting the ADC accuracy	A	A
	2.2.9	Possible delay in backup domain protection disabling/enabling after programming the DBP bit	A	A
	2.2.10	PC13 signal transitions disturb LSE	A	A
	2.2.11	In some specific cases, DMA2 data corruption occurs when managing AHB and APB2 peripherals in a concurrent way	A	A
ADC	2.3.1	ADC sequencer modification during conversion	A	A
DAC	2.4.1	DMA request not automatically cleared by clearing DMAEN	A	A
	2.4.2	DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge	N	N
TIM	2.5.1	PWM re-enabled in automatic output enable mode despite of system break	P	P
	2.5.3	Consecutive compare event missed in specific conditions	N	N
	2.5.4	Output compare clear not working with external counter reset	P	P
IWDG	2.6.1	RVU flag not reset in Stop	A	A
	2.6.2	PVU flag not reset in Stop	A	A
	2.6.3	RVU flag not cleared at low APB clock frequency	A	A

Function	Section	Limitation	Status	
			Rev. A	Rev. 1
IWDG	2.6.4	PVU flag not cleared at low APB clock frequency	A	A
RTC	2.7.1	Spurious tamper detection when disabling the tamper channel	N	N
	2.7.2	RTC calendar registers are not locked properly	A	A
	2.7.3	RTC interrupt can be masked by another RTC interrupt	A	A
	2.7.4	Calendar initialization may fail in case of consecutive INIT mode entry	A	A
	2.7.5	Alarm flag may be repeatedly set when the core is stopped in debug	N	N
FMPI2C	2.8.1	10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave	A	A
	2.8.2	Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one FMPI2C kernel clock period	P	P
	2.8.3	Spurious bus error detection in master mode	A	A
	2.8.4	Last-received byte loss in reload mode	P	P
	2.8.5	Spurious master transfer upon own slave address match	P	P
	2.8.7	OVR flag not set in underrun condition	N	N
	2.8.8	Transmission stalled after first byte transfer	A	A
	2.8.9	SDA held low upon SMBus timeout expiry in slave mode	A	A
I2C	2.9.2	SMBus standard not fully supported	A	A
	2.9.3	Start cannot be generated after a misplaced Stop	A	A
	2.9.4	Mismatch on the “Setup time for a repeated Start condition” timing parameter	A	A
	2.9.5	Data valid time ($t_{VD;DAT}$) violated without the OVR flag being set	A	A
	2.9.6	Both SDA and SCL maximum rise times (t_r) violated when the VDD_I2C bus voltage is higher than ((V_{DD} + 0.3) / 0.7) V	A	A
	2.9.1	Spurious bus error detection in master mode	A	A
USART	2.10.1	Idle frame is not detected if the receiver clock speed is deviated	N	N
	2.10.2	In full-duplex mode, the Parity Error (PE) flag can be cleared by writing to the data register	A	A
	2.10.3	Parity Error (PE) flag is not set when receiving in Mute mode using address mark detection	N	N
	2.10.4	Break frame is transmitted regardless of CTS input line status	N	N
	2.10.5	RTS signal abnormally driven low after a protocol violation	A	A
	2.10.6	Start bit detected too soon when sampling for NACK signal from the smartcard	N	N
	2.10.7	Break request can prevent the transmission complete flag (TC) from being set	A	A
	2.10.8	Guard time not respected when data are sent on TXE events	A	A
	2.10.9	RTS is active while RE or UE = 0	A	A
SPI/I2S	2.11.1	BSY bit may stay high when SPI is disabled	A	A
	2.11.2	Anticipated communication upon SPI transit from slave receiver to master	A	A
	2.11.3	Wrong CRC calculation when the polynomial is even	A	A
	2.11.4	Corrupted last bit of data and/or CRC received in Master mode with delayed SCK feedback	A	A
	2.11.5	BSY flag may stay high at the end of a data transfer in Slave mode	A	A

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

Table 4. Summary of device documentation errata

Function	Section	Documentation erratum
TIM	2.5.2	TRGO and TRGO2 trigger output failure
FMPI2C	2.8.6	START bit is cleared upon setting ADDRCF, not upon address match
	2.8.10	Inconsistent FMPI2C peripheral instance naming

2 Description of device errata

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

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

2.1 Core

Reference manual and errata notice for the Arm® Cortex®-M4F core revision r0p1 is available from http://infocenter.arm.com. Only applicable information from the Arm errata notice is replicated in this document.

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 Debugging Stop mode and SysTick timer

Description

If the SysTick timer interrupt is enabled during the Stop mode debug (DBG_STOP bit set in the DBGMCU_CR register), it wakes up the system from Stop mode.

Workaround

To debug the Stop mode, disable the SysTick timer interrupt.

2.2.2 Debugging Stop mode with WFE entry

Description

When the Stop debug mode is enabled (DBG_STOP bit set in the DBGMCU_CR register), the software debugging is allowed during Stop mode. However, if the application software uses the WFE instruction to enter Stop mode, after wake-up, some instructions may be missed if the WFE is followed by sequential instructions. This affects only Stop debug mode with WFE entry.

Workaround

To debug Stop mode with WFE entry, the WFE instruction must be inside a dedicated function with one instruction (NOP) between the execution of the WFE and the Bx LR. For example:

```
__asm void _WFE(void)
WFE
NOP
BX lr }
```

2.2.3 Debugging Sleep/Stop mode with WFE/WFI entry

Description

When the Sleep debug or Stop debug mode is enabled (DBG_SLEEP bit or DBG_STOP bit are set in the DBGMCU_CR register), software debugging is allowed during Sleep or Stop mode. After wake-up, some unreachable instructions can be executed if the following conditions are met:

- The application software disables the Prefetch queue,
- the number of wait states configured for the flash memory interface is higher than zero, and
- the linker places the WFE or WFI instruction on a 4-byte aligned addresses (0x080xx xxx4).

Workaround

Apply one of the following measures:

- Add three NOPs after WFI/WFE instruction.
- Keep one AHB master active during Sleep (example keep DMA1 or DMA2 RCC clock enable bit set).
- Execute WFI/WFE instruction from routines inside the SRAM.

2.2.4 Wake-up sequence from Standby mode when using more than one wake-up source

Description

The various wake-up sources are logically OR-ed in front of the rising-edge detector that generates the wake-up flag (WUF). The WUF needs to be cleared before Standby mode entry, otherwise the MCU wakes up immediately. If one of the configured wake-up sources is kept high during the clearing of the WUF (by setting the CWUF bit), it may mask further wake-up events on the input of the edge detector. As a consequence, the MCU may not be able to wake up from Standby mode.

Workaround

To avoid this problem, apply the following sequence before entering Standby mode:

1. Disable all used wake-up sources.
2. Clear all related wake-up flags.
3. Reenable all used wake-up sources.
4. Enter Standby mode.

Note: Be aware that, when applying this workaround, if one of the wake-up sources is still kept high, the MCU enters Standby mode but then it wakes up immediately and generates a power reset.

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 MPU attribute to RTC and IWDG registers incorrectly managed

Description

If the MPU is used and the nonbufferable attribute is set to the RTC or IWDG memory map region, the CPU access to the RTC or IWDG registers may be treated as bufferable, provided there is no APB prescaler configured (AHB/APB prescaler is equal to 1).

Workaround

If the nonbufferable attribute is required for these registers, perform by software a read after the write to guaranty the completion of the write access.

2.2.7 Delay after an RCC peripheral clock enabling

Description

A delay may be observed between an RCC peripheral clock enable and the effective peripheral enabling. It must be taken into account in order to manage the peripheral read/write from/to registers.

This delay depends on the peripheral mapping:

- If the peripheral is mapped on the AHB: the delay may be equal to two AHB cycles.
- If the peripheral is mapped on the APB: the delay may be equal to 1 + (AHB/APB prescaler) cycles.

Workaround

Apply one of the following measures:

- Use the DSB instruction to stall the Arm® Cortex®-M4 CPU pipeline until the instruction has completed.
- Insert “n” NOPs between the RCC enable bit write and the peripheral register writes (n = 2 for AHB peripherals, n = 1 + AHB/APB prescaler for APB peripherals).
- Simply insert a dummy read operation from the corresponding register just after enabling the peripheral clock.

2.2.8 Internal noise impacting the ADC accuracy

Description

An internal noise generated on V_{DD} supplies and propagated internally may impact the ADC accuracy. This noise is always present whatever the power mode of the MCU (Run or Sleep).

Workaround

Use the following sequence to adapt the accuracy level to the application requirements:

1. Configure the flash memory ART with prefetch OFF and data + instruction cache ON.
2. Use averaging and filtering algorithms on ADC output codes.

For more detailed workarounds, refer to the application note "*How to improve ADC accuracy when using STM32F2xx and STM32F4xx microcontrollers*" (AN4073).

2.2.9 Possible delay in backup domain protection disabling/enabling after programming the DBP bit

Description

Depending on the AHB/APB1 prescaler, a delay between DBP bit programming and the effective disabling/enabling of the backup domain protection can be observed and must be taken into account.

The higher the APB1 prescaler value, the higher the delay.

Workaround

Apply one of the following measures:

- Insert a dummy read operation to the PWR_CR register just after programming the DBP bit.
- Wait for the end of the operation (reset through the BDRST bit or write to the backup domain) via a polling loop on targeted registers.

2.2.10 PC13 signal transitions disturb LSE

Description

The PC13 input/output toggling disturbs the LSE clock. As a result, PC13 may not be usable when LSE is used.

Workaround

Use an external clock with the LSE in bypass mode.

2.2.11 In some specific cases, DMA2 data corruption occurs when managing AHB and APB2 peripherals in a concurrent way

Description

When the DMA2 is managing concurrent requests of AHB and APB2 peripherals, the transfer on the AHB can be performed several times. Impacted peripheral are:

- QUADSPI: indirect mode read and write transfers
- FSMC: read and write operation with external device having FIFO
- GPIO: DMA2 transfers to GPIO registers (in memory-to-peripheral transfer mode). The transfers from the GPIOs register are not impacted.

The data corruption is due to multiple DMA2 accesses over the AHB peripheral port impacting peripherals embedding a FIFO.

For transfer to the internal SRAM through the DMA2 AHB peripheral port, the accesses can be performed several times but without data corruptions in cases of concurrent requests.

Workaround

- Use the DMA2 AHB memory port when reading/writing from/to QUADSPI and FSMC instead of DMA2 AHB default peripheral port.
- Use the DMA2 AHB memory port when writing to GPIOs instead of DMA2 AHB default peripheral port.

For more details about DMA controller features, refer to the section *Take benefits of DMA2 controller and system architecture flexibility* of the application note "*Using the STM32F2, STM32F4 and STM32F7 Series DMA controller*" (AN4031).

2.3 ADC

2.3.1 ADC sequencer modification during conversion

Description

When a software start-of-conversion is used as an ADC trigger, and if the ADC_SQRx or ADC_JSQRx register is modified during the conversion, the current conversion is reset and the ADC does not automatically restart the new conversion sequence. The hardware start-of-conversion trigger is not impacted and the ADC automatically restarts the new sequence when the next hardware trigger occurs.

Workaround

When a software start-of-conversion is used, apply the following sequence:

1. First set the SWSART bit in the ADC_CR2 register.
2. Then restart the new conversion sequence.

2.4 DAC

2.4.1 DMA request not automatically cleared by clearing DMAEN

Description

Upon an attempt to stop a DMA-to-DAC transfer, the DMA request is not automatically cleared by clearing the DAC channel bit of the DAC_CR register (DMAEN) or by disabling the DAC clock.

If the application stops the DAC operation while the DMA request is pending, the request remains pending while the DAC is reinitialized and restarted, with the risk that a spurious DMA request is serviced as soon as the DAC is enabled again.

Workaround

Apply the following sequence to stop the current DMA-to-DAC transfer and restart the DAC:

1. Check if DMAUDR bit is set in DAC_CR.
2. Clear the DAC channel DMAEN bit.
3. Disable the DAC clock.
4. Reconfigure the DAC, DMA and the triggers.
5. Restart the application.

2.4.2 DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge

Description

When the DAC channel operates in DMA mode (DMAEN of DAC_CR register set), the DMA channel underrun flag (DMAUDR of DAC_SR register) fails to rise upon an internal trigger detection if that detection occurs during the same clock cycle as a DMA request acknowledge. As a result, the user application is not informed that an underrun error occurred.

This issue occurs when software and hardware triggers are used concurrently to trigger DMA transfers.

Workaround

None.

2.5 TIM

2.5.1 PWM re-enabled in automatic output enable mode despite of system break

Description

In automatic output enable mode (AOE bit set in TIMx_BDTR register), the break input can be used to do a cycle-by-cycle PWM control for a current mode regulation. A break signal (typically a comparator with a current threshold) disables the PWM output(s) and the PWM is re-armed on the next counter period.

However, a system break (typically coming from the CSS Clock security System) is supposed to stop definitively the PWM to avoid abnormal operation (for example with PWM frequency deviation).

In the current implementation, the timer system break input is not latched. As a consequence, a system break indeed disables the PWM output(s) when it occurs, but PWM output(s) is (are) re-armed on the following counter period.

Workaround

Preferably, implement control loops with the output clear enable function (OCxCE bit in the TIMx_CCMR1/CCMR2 register), leaving the use of break circuitry solely for internal and/or external fault protection (AOE bit reset).

2.5.2 TRGO and TRGO2 trigger output failure

Description

Some reference manual revisions may omit the following information.

The timers can be linked using ITRx inputs and TRGOx outputs. Additionally, the TRGOx outputs can be used as triggers for other peripherals (for example ADC). Since this circuitry is based on pulse generation, care must be taken when initializing master and slave peripherals or when using different master/slave clock frequencies:

- If the master timer generates a trigger output pulse on TRGOx prior to have the destination peripheral clock enabled, the triggering system may fail.
- If the frequency of the destination peripheral is modified on-the-fly (clock prescaler modification), the triggering system may fail.

As a conclusion, the clock of the slave timer or slave peripheral must be enabled prior to receiving events from the master timer, and must not be changed on-the-fly while triggers are being received from the master timer.

This is a documentation issue rather than a product limitation.

Workaround

No application workaround is required or applicable as long as the application handles the clock as indicated.

2.5.3 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:

- <u>in edge-aligned mode</u>, from ARR to 0:
  - first compare event: CNT = CCR = ARR
  - second (missed) compare event: CNT = CCR = 0
- <u>in center-aligned mode while up-counting</u>, 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
- <u>in center-aligned mode while down-counting</u>, 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.

2.5.4 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).

2.6 IWDG

2.6.1 RVU flag not reset in Stop

Description

Successful write to the IWDG_RLR register raises the RVU flag and prevents further write accesses to the register until the RVU flag is automatically cleared by hardware. However, if the device enters Stop mode while the RVU flag is set, the hardware never clears that flag, and writing to the IWDG_RLR register is no longer possible.

Workaround

Ensure that the RVU flag is cleared before entering Stop mode.

2.6.2 PVU flag not reset in Stop

Description

Successful write to the IWDG_PR register raises the PVU flag and prevents further write accesses to the register until the PVU flag is automatically cleared by hardware. However, if the device enters Stop mode while the PVU flag is set, the hardware never clears that flag, and writing to the IWDG_PR register is no longer possible.

Workaround

Ensure that the PVU flag is cleared before entering Stop mode.

2.6.3 RVU flag not cleared at low APB clock frequency

Description

Successful write to the IWDG_RLR register raises the RVU flag and prevents further write accesses to the register until the RVU flag is automatically cleared by hardware. However, at APB clock frequency lower than twice the IWDG clock frequency, the hardware never clears that flag, and writing to the IWDG_RLR register is no longer possible.

Workaround

Set the APB clock frequency higher than twice the IWDG clock frequency.

2.6.4 PVU flag not cleared at low APB clock frequency

Description

Successful write to the IWDG_PR register raises the PVU flag and prevents further write accesses to the register until the PVU flag is automatically cleared by hardware. However, at APB clock frequency lower than twice the IWDG clock frequency, the hardware never clears that flag, and writing to the IWDG_PR register is no longer possible.

Workaround

Set the APB clock frequency higher than twice the IWDG clock frequency.

2.7 RTC

2.7.1 Spurious tamper detection when disabling the tamper channel

Description

If the tamper detection is configured for detecting on the falling edge event (TAMPFLT = 00 and TAMPxTRG = 1) and if the tamper event detection is disabled when the tamper pin is at high level, a false tamper event is detected.

Workaround

None.

2.7.2 RTC calendar registers are not locked properly

Description

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

Workaround

Apply one of the following measures:
- Use BYPSHAD = 1 mode (bypass shadow registers), or
- If BYPSHAD = 0, read SSR again after reading SSR/TR/DR to confirm that SSR is still the same, otherwise read the values again.

2.7.3 RTC interrupt can be masked by another RTC interrupt

Description

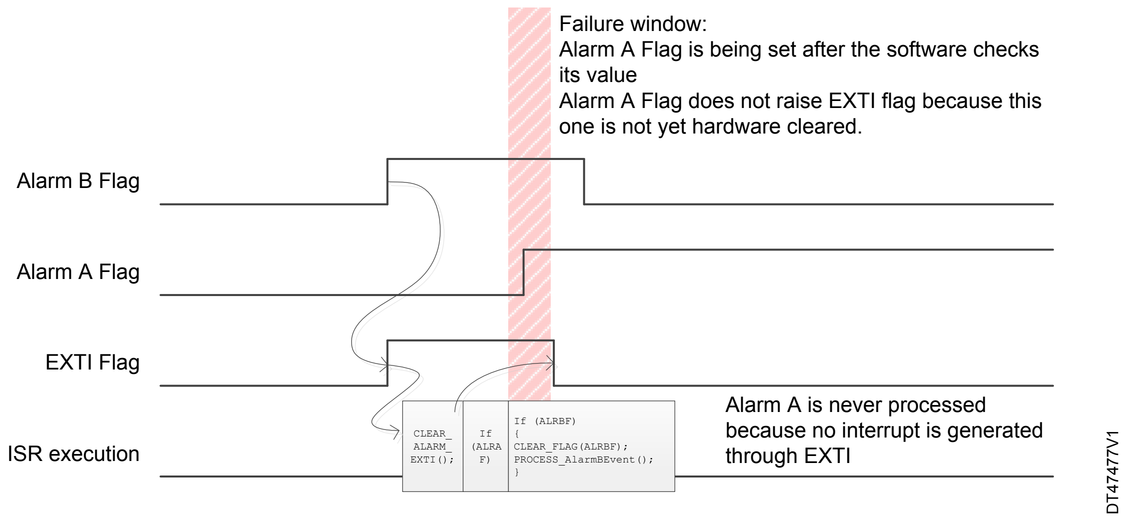
One RTC interrupt request can mask another RTC interrupt request if they share the same EXTI configurable line. For example, interrupt requests from Alarm A and Alarm B or those from tamper and timestamp events are OR-ed to the same EXTI line (refer to the *EXTI line connections* table in the *Extended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

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

Alarm interrupt service routine:

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

Figure 1. Masked RTC interrupt



Workaround

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

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

2.7.4 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.

2.7.5 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 when 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 attempt to clear the flag(s) ineffective.

Workaround

None.

2.8 FMPI2C

2.8.1 10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave

Description

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

When the MCU set as I^2C-bus master transmits a 10-bit address of which the first byte (5-bit header + 2 MSBs of the address + direction bit) is not acknowledged, the MCU duly generates STOP condition but it then cannot start any new I^2C-bus transfer. In this spurious state, the NACKF flag of the FMPI2C_ISR register and the START bit of the FMPI2C_CR2 register are both set, while the START bit should normally be cleared.

Workaround

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

1. Wait for the STOP condition detection (STOPF = 1 in FMPI2C_ISR register).
2. Disable the FMPI2C peripheral.
3. Wait for a minimum of three APB cycles.
4. Enable the FMPI2C peripheral again.

2.8.2 Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one FMPI2C 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 FMPI2C 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 FMPI2C kernel clock frequency to get FMPI2C kernel clock period within the transmitter minimum data setup time. Alternatively, increase transmitter's minimum data setup time. If the transmitter setup time minimum value corresponds to the minimum value provided in the I^{2}C-bus standard, the minimum FMPI2CCLK frequencies are as follows:

- In Standard mode, if the transmitter minimum setup time is 250 ns, the FMPI2CCLK frequency must be at least 4 MHz.
- In Fast mode, if the transmitter minimum setup time is 100 ns, the FMPI2CCLK frequency must be at least 10 MHz.
- In Fast-mode Plus, if the transmitter minimum setup time is 50 ns, the FMPI2CCLK frequency must be at least 20 MHz.

2.8.3 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 FMPI2C_SR register and generating bus error interrupt if such interrupt is enabled. Detection of bus error has no effect on the I^{2}C-bus transfer in master mode and any such transfer continues normally.

Workaround

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

2.8.4 Last-received byte loss in reload mode

Description

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

- I^{2}C-bus stretching is enabled (NOSTRETCH = 0)
- RELOAD bit of the FMPI2C_CR2 register is set
- NBYTES bitfield of the FMPI2C_CR2 register is set to N greater than 1
- byte N is received on the I^{2}C-bus, raising the TCR flag
- N - 1 byte is not yet read out from the data register at the instant TCR is raised,

then the SCL line is pulled low (I^{2}C-bus clock stretching) and the transfer of the byte N from the shift register to the data register inhibited until the byte N-1 is read and NBYTES bitfield reloaded with a new value, the latter of which also clears the TCR flag. As a consequence, the software cannot get the byte N and use its content before setting the new value into the NBYTES field.

Workaround

- In master mode or in slave mode with SBC = 1, use the reload mode with NBYTES = 1.
- In master receiver mode, if the number of bytes to transfer is greater than 255, do not use the reload mode. Instead, split the transfer into sections not exceeding 255 bytes and separate them with repeated START conditions.
- Make sure, for example through the use of DMA, that the byte N - 1 is always read before the TCR flag is raised.

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

2.8.5 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^{2}C-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 FMPI2C_ISR register) occurs.
- After the ADDR flag is set:
  - the device does not write FMPI2C_CR2 before clearing the ADDR flag, or
  - the device writes FMPI2C_CR2 earlier than three FMPI2C 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 FMPI2C_CR2 register when the master transfer starts. Moreover, if the FMPI2C_CR2 is written less than three kernel clocks before the ADDR flag is cleared, the FMPI2C 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 FMPI2C_CR2 with the START bit low.

Slave byte control mode (SBC = 1):

1. Write FMPI2C_CR2 with the slave transfer configuration and the START bit low.
2. Wait for longer than three FMPI2C kernel clock cycles.
3. Set the ADDRCF bit.
4. Before Stop condition occurs on the bus, write FMPI2C_CR2 again with its current value.

The time for the software application to write the FMPI2C_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 FMPI2C_CR2 register with the START bit set.

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

Description

Some reference manual revisions may state that the START bit of the FMPI2C_CR2 register is cleared upon slave address match event.

Instead, the START bit is cleared upon setting, by software, the ADDRCF bit of the FMPI2C_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.

2.8.7 OVR flag not set in underrun condition

Description

In slave transmission with clock stretching disabled (NOSTRETCH = 1 in the FMPI2C_CR1 register), an underrun condition occurs if the current byte transmission is completed on the I^{2}C 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 FMPI2C_ISR register and send 0xFF on the bus.

However, if the FMPI2C_TXDR is written within the interval between two FMPI2C 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.

2.8.8 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 FMPI2C_TXDR register in less than two FMPI2C kernel clock cycles after the TXIS/DMA request, and the ratio between APB clock and FMPI2C kernel clock frequencies is between 1.5 and 3, the second byte written in the FMPI2C_TXDR is not internally detected. This causes a state in which the FMPI2C 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 FMPI2C_TXDR before the transmission starts.
- Set the APB clock frequency so that its ratio with respect to the FMPI2C kernel clock frequency is lower than 1.5 or higher than 3.

2.8.9 SDA held low upon SMBus timeout expiry in slave mode

Description

For the slave mode, the SMBus specification defines $t_{TIMEOUT}$ (detect clock low timeout) and $t_{LOW:SEXT}$ (cumulative clock low extend time) timeouts. When one of them expires while the FMPI2C peripheral in slave mode drives SDA low to acknowledge either its address or a data transmitted by the master, the device is expected to report such an expiry and release the SDA line.

However, although the device duly reports the timeout expiry, it fails to release SDA. This stalls the I^2C bus and prevents the master from generating RESTART or STOP condition.

Workaround

When a timeout is reported in slave mode (TIMEOUT bit of the FMPI2C_ISR register is set), apply this sequence:

1. Wait until the frame is expected to end.
2. Read the STOPF bit of the FMPI2C_ISR register. If it is low, reset the FMPI2C kernel by clearing the PE bit of the FMPI2C_CR1 register.
3. Wait for at least three APB clock cycles before enabling again the FMPI2C peripheral.

2.8.10 Inconsistent FMPI2C peripheral instance naming

Description

In the silicon product documentation (reference manual, data sheet, and application notes), and in the STM32CubeF4 and STM32CubeMX ecosystem tools and documentation, *I2C4* and *FMPI2C1* refer to the same peripheral instance. This also impacts the associated signal and pin names.

This is a documentation erratum, not a device limitation.

Workaround

No application workaround is required. Read all occurrences of *I2C4* as *FMPI2C1*.

2.9 I2C

2.9.1 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.

2.9.2 SMBus standard not fully supported

Description

The I2C peripheral is not fully compliant with the SMBus v2.0 standard since it does not support the capability to NACK an invalid byte/command.

Workaround

A higher-level mechanism must be used to verify that a write operation is being performed correctly at the target device, such as:

- the SMBAL pin if it is supported by the host
- the alert response address (ARA) protocol
- the host-notify protocol

2.9.3 Start cannot be generated after a misplaced Stop

Description

If a master generates a misplaced Stop on the bus (bus error) while the microcontroller I2C peripheral attempts to switch to Master mode by setting the START bit, the Start condition is not properly generated.

Workaround

In the I^2C standard, it is allowed to send a Stop only at the end of the full byte (8 bits + acknowledge), so this scenario is not allowed. Other derived protocols such as CBUS allow it, but they are not supported by the I2C peripheral.
A software workaround consists in asserting the software reset using the SWRST bit of the I2C_CR1 control register.

2.9.4 Mismatch on the “Setup time for a repeated Start condition” timing parameter

Description

In case of repeated Start, the “Setup time for a repeated Start condition” (named Tsu;sta in the I^2C specification) can be slightly violated when the I2C operates in Master standard mode at a frequency between 88 kHz and 100 kHz.
The issue can occur only in the following configuration:

- In Master mode
- In Standard mode at a frequency between 88 kHz and 100 kHz (no limitation in Fast mode)
- SCL rise time:
  - If the slave does not stretch the clock and the SCL rise time is more than 300 ns (if the SCL rise time is less than 300 ns, the issue does not occur).
  - If the slave stretches the clock.

The setup time can be violated independently of the APB peripheral frequency.

Workaround

Reduce the frequency down to 88 kHz or use the I^2C Fast mode, if it is supported by the slave.

2.9.5 Data valid time ($t_{VD;DAT}$) violated without the OVR flag being set

Description

The data valid time ($t_{VD;DAT}$, $t_{VD;ACK}$) described by the I^2C standard can be violated (as well as the maximum data hold time of the current data ($t_{HD;DAT}$)) under the conditions described below. This violation cannot be detected because the OVR flag is not set (no transmit buffer underrun is detected).

This limitation can occur only under the following conditions:

- in Slave transmit mode
- with clock stretching disabled (NOSTRETCH = 1)
- if the software is late to write to the DR data register, but not late enough to set the OVR flag (the data register is written before)

Workaround

If the master device allows it, use the clock stretching mechanism by clearing the bit NOSTRETCH of the I2C_CR1 register.

If the master device does not allow it, ensure that the software is fast enough when polling the TXE or ADDR flag to immediately write to the DR data register. For instance, use an interrupt on the TXE or ADDR flag and boost its priority to the higher level.

2.9.6 Both SDA and SCL maximum rise times (t_r) violated when the VDD_I2C bus voltage is higher than ((V_{DD} + 0.3) / 0.7) V

Description

When an external legacy I^2C bus voltage (V_{DD_I2C}) is set to 5 V while the MCU is powered from V_{DD}, the internal 5-Volt tolerant circuitry is activated as soon the input voltage (V_{IN}) reaches the V_{DD} + diode threshold level. An additional internal large capacitance then prevents the external pull-up resistor (R_P) from rising the SDA and SCL signals within the maximum timing (t_r), which is 300 ns in Fast mode and 1000 ns in Standard mode.

The rise time (t_r) is measured from V_{IL} and V_{IH} with levels set at 0.3 V_{DD_I2C} and 0.7 V_{DD_I2C}.

Workaround

The external V_{DD_I2C} bus voltage must be limited to a maximum value of ((V_{DD} + 0.3) / 0.7) V. As a result, when the MCU is powered from V_{DD} = 3.3 V, V_{DD_I2C}must not exceed 5.14 V to be compliant with I^2C specifications.

2.10 USART

2.10.1 Idle frame is not detected if the receiver clock speed is deviated

Description

If the USART receives an idle frame followed by a character, and the clock of the transmitter device is faster than the USART receiver clock, the USART receive signal falls too early when receiving the character start bit, with the result that the idle frame is not detected (the IDLE flag is not set).

Workaround

None.

2.10.2 In full-duplex mode, the Parity Error (PE) flag can be cleared by writing to the data register

Description

In full-duplex mode, when the Parity Error flag is set by the receiver at the end of a reception, it may be cleared while transmitting by reading the USART_SR register to check the TXE or TC flags and writing data to the data register. Consequently, the software receiver can read the PE flag as '0' even if a parity error occurred.

Workaround

The Parity Error flag should be checked after the end of reception and before transmission.

2.10.3 Parity Error (PE) flag is not set when receiving in Mute mode using address mark detection

Description

If the USART receiver is in Mute mode, and is configured to exit from Mute mode using the address mark detection, when the USART receiver recognizes a valid address with a parity error, it exits from Mute mode without setting the Parity Error flag.

Workaround

None.

2.10.4 Break frame is transmitted regardless of CTS input line status

Description

When the CTS hardware flow control is enabled (CTSE = 1) and the send break bit (SBK) is set, the transmitter sends a break frame at the end of the current transmission regardless of CTS input line status. Consequently, if an external receiver device is not ready to accept a frame, the transmitted break frame is lost.

Workaround

None.

2.10.5 RTS signal abnormally driven low after a protocol violation

Description

When RTS hardware flow control is enabled, the RTS signal goes high when data is received. If this data was not read and new data is sent to the USART (protocol violation), the RTS signal goes back to low level at the end of this new data.

Consequently, the sender gets the wrong information that the USART is ready to receive further data.

On the USART side, an overrun is detected, which indicates that data has been lost.

Workaround

A workaround is required only if the other USART device violates the communication protocol, which is not the case in most applications.

Two workarounds can be used:

- After data reception and before reading the data in the data register, the software takes over the control of the RTS signal as a GPIO, and holds it high as long as needed. If the USART device is not ready, the software holds the RTS pin high, and releases it when the device is ready to receive new data.
- Make sure the time required by the software to read the received data is always lower than the duration of the second data reception. For example, this can be ensured by handling all the receptions in DMA mode.

2.10.6 Start bit detected too soon when sampling for NACK signal from the smartcard

Description

According to ISO/IEC 7816-3 standard, when a character parity error is detected, the receiver shall transmit a NACK error signal 10.5 ± 0.2 ETUs after the character START bit falling edge. In this case, the transmitter is able to detect correctly the NACK signal until 11 ± 0.2 ETUs after the character START bit falling edge. In Smartcard mode, the USART peripheral monitors the NACK signal during the receiver time frame (10.5 ± 0.2 ETUs), while it should wait for it during the transmitter one (11 ± 0.2 ETUs). In real cases, this would not be a problem as the card itself needs to respect a 10.7 ETU period when sending the NACK signal. However, this may be an issue to undertake a certification.

Workaround

None.

2.10.7 Break request can prevent the transmission complete flag (TC) from being set

Description

After the end of transmission of a data (D1), the transmission complete (TC) flag is not set if the following conditions are met:

- CTS hardware flow control is enabled,
- D1 is being transmitted,
- a break transfer is requested before the end of D1 transfer,
- CTS is de-asserted before the end of D1 data transfer.

Workaround

If the application needs to detect the end of a data transfer, check that the TC flag is set, and issue a break request.

2.10.8 Guard time not respected when data are sent on TXE events

Description

In Smartcard mode, when sending a data on TXE event, the programmed guard time is not respected, that is the data written in the data register is transferred to the bus without waiting the completion of the guard-time duration corresponding to the previous transmitted data.

Workaround

Since in Smartcard mode the TC flag is set at the end of the guard time duration, wait until TC is set, then write the data.

2.10.9 RTS is active while RE or UE = 0

Description

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

Workaround

After setting the UE and RE bits, configure the I/O used for RTS as an alternate function.

2.11 SPI/I2S

2.11.1 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.

2.11.2 Anticipated communication upon SPI transit from slave receiver to master

Description

Regardless of the master mode configured, the communication clock starts upon setting the MSTR bit even though the SPI is disabled, if transiting from receive-only (RXONLY = 1) or half-duplex receive (BIDIMODE = 1 and BIDIOE = 0) slave mode to master mode.

Workaround

Apply one of the following measures:

- Before transiting to master mode, hardware-reset the SPI via the reset controller.
- Set the MSTR and SPE bits of the SPI configuration register simultaneously, which forces the immediate start of the communication clock. In transmitter configuration, load the data register in advance with the data to send.

2.11.3 Wrong CRC calculation when the polynomial is even

Description

When the CRC is enabled, the CRC calculation is wrong if the polynomial is even.

Workaround

Use odd polynomial.

2.11.4 Corrupted last bit of data and/or CRC received in Master mode with delayed SCK feedback

Description

When performing a receive transaction in I2S or SPI Master mode, the last bit of the transacted frame is not captured when the signal provided by an internal feedback loop from the SCK pin exceeds a critical delay. The lastly transacted bit of the stored data then keeps the value from the pattern received previously. As a consequence, the last receive data bit may be wrong, and/or the CRCERR flag can be unduly asserted in the SPI mode if any data under checksum, and/or just the CRC pattern is wrongly captured.

In SPI mode, data are synchronous with the APB clock. A delay of up to two APB clock periods can thus be tolerated for the internal feedback delay.

The I2S mode is more sensitive than the SPI mode, especially in the case where an odd I2S prescaler factor is set and the APB clock is the system clock divided by two. In this case, the internal feedback delay is lower than 1.5 APB clock period.

The main factors contributing to the delay increase are low V_{DD} level, high temperature, high SCK pin capacitive load, and low SCK I/O output speed. The SPI communication speed has no impact.

Workaround

The following workarounds can be adopted, jointly or individually:

- Decrease the APB clock speed.
- Configure the I/O pad of the SCK pin to be faster.

The following table gives the maximum allowable APB frequency (that still prevents the issue from occurring) versus GPIOx_OSPEEDR output speed for the SCK pin, with a 30 pF capacitive load.

Table 5. Maximum allowable APB frequency at 30 pF load

OSPEEDR [1:0] for SCK pin	Max. APB frequency for SPI mode (MHz)	Max. APB frequency for I2S mode (MHz)
11 (very high), 10 (high)	84	42
01 (medium)	75	35
00 (low)	25	16

2.11.5 BSY flag may stay high at the end of a data transfer in Slave mode

Description

The BSY flag may sporadically remain high at the end of a data transfer in Slave mode. The issue appears when an accidental synchronization happens between the internal CPU clock and the external SCK clock provided by the master.

This is related to the end of data transfer detection while the SPI is enabled in Slave mode.

As a consequence, the end of the data transaction may be not recognized when the software needs to monitor it (for example at the end of a session before entering the low-power mode or before the direction of the data line has to be changed at half duplex bidirectional mode). The BSY flag is unreliable to detect the end of any data sequence transaction.

Workaround

When the NSS hardware management is applied and the NSS signal is provided by the master, the end of a transaction can be detected by the NSS polling by the slave:

- If the SPI receiving mode is enabled, the end of a transaction with the master can be detected by the corresponding RXNE event signaling the last data transfer completion.
- In SPI transmit mode, the user can check the BSY under timeout corresponding to the time necessary to complete the last data frame transaction. The timeout must be measured from TXE event signaling the last data frame transaction start (it is raised once the second bit transaction is ongoing). Either BSY becomes low normally or the timeout expires when the synchronization issue happens.

When the above workarounds are not applicable, the following sequence can be used to prevent the synchronization issue during SPI transmit mode:

1. Write the last data to the data register.
2. Poll TXE until it becomes high to ensure the data transfer has started.
3. Disable SPI by clearing SPE while the last data transfer is still ongoing.
4. Poll the BSY bit until it becomes low.
5. The BSY flag works correctly and can be used to recognize the end of the transaction.

Note: This workaround can be used only when the CPU has enough performance to disable the SPI after a TXE event is detected, while the data frame transfer is still ongoing. It is impossible to achieve it when the ratio between CPU and SPI clock is low, and the data frame is short. In this specific case, the timeout can be measured from TXE, while calculating the fixed number of CPU clock periods corresponding to the time necessary to complete the data frame transaction.

Important security notice

The STMicroelectronics group of companies (ST) places a high value on product security, which is why the ST product(s) identified in this documentation may be certified by various security certification bodies and/or may implement our own security measures as set forth herein. However, no level of security certification and/or built-in security measures can guarantee that ST products are resistant to all forms of attacks. As such, it is the responsibility of each of ST's customers to determine if the level of security provided in an ST product meets the customer needs both in relation to the ST product alone, as well as when combined with other components and/or software for the customer end product or application. In particular, take note that:

- ST products may have been certified by one or more security certification bodies, such as Platform Security Architecture (www.psacertified.org) and/or Security Evaluation standard for IoT Platforms (www.trustcb.com). For details concerning whether the ST product(s) referenced herein have received security certification along with the level and current status of such certification, either visit the relevant certification standards website or go to the relevant product page on www.st.com for the most up to date information. As the status and/or level of security certification for an ST product can change from time to time, customers should re-check security certification status/level as needed. If an ST product is not shown to be certified under a particular security standard, customers should not assume it is certified.
- Certification bodies have the right to evaluate, grant and revoke security certification in relation to ST products. These certification bodies are therefore independently responsible for granting or revoking security certification for an ST product, and ST does not take any responsibility for mistakes, evaluations, assessments, testing, or other activity carried out by the certification body with respect to any ST product.
- Industry-based cryptographic algorithms (such as AES, DES, or MD5) and other open standard technologies which may be used in conjunction with an ST product are based on standards which were not developed by ST. ST does not take responsibility for any flaws in such cryptographic algorithms or open technologies or for any methods which have been or may be developed to bypass, decrypt or crack such algorithms or technologies.
- While robust security testing may be done, no level of certification can absolutely guarantee protections against all attacks, including, for example, against advanced attacks which have not been tested for, against new or unidentified forms of attack, or against any form of attack when using an ST product outside of its specification or intended use, or in conjunction with other components or software which are used by customer to create their end product or application. ST is not responsible for resistance against such attacks. As such, regardless of the incorporated security features and/or any information or support that may be provided by ST, each customer is solely responsible for determining if the level of attacks tested for meets their needs, both in relation to the ST product alone and when incorporated into a customer end product or application.
- All security features of ST products (inclusive of any hardware, software, documentation, and the like), including but not limited to any enhanced security features added by ST, are provided on an "AS IS" BASIS. AS SUCH, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ST DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, unless the applicable written and signed contract terms specifically provide otherwise.

Revision history

Table 6. Document revision history

Date	Version	Changes
28-Sep-2015	1	Initial release.
11-Jan-2017	2	Added: • Section 2.1.7: In some specific cases, DMA2 data corruption occurs when managing AHB and APB2 peripherals in a concurrent way • Table 5: Maximum allowable APB frequency at 30 pF load Updated: • Section 2.5.2: BSY bit may stay high at the end of a data transfer in Slave mode • Section 2.5.3: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback
07-Mar-2023	3	Added device revision 1 Added errata sections: TIM and RTC Added errata: • **Core:** Store immediate overlapping exception return operation might vector to incorrect interrupt • **System:** Debugging Sleep/Stop mode with WFE/WFI entry • Internal noise impacting the ADC accuracy • Possible delay in backup domain protection disabling/enabling after programming the DBP bit • PC13 signal transitions disturb LSE • **TIM:** PWM re-enabled in automatic output enable mode despite of system break • TRGO and TRGO2 trigger output failure • Consecutive compare event missed in specific conditions • Output compare clear not working with external counter reset • **IWDG:** RVU flag not reset in Stop • PVU flag not reset in Stop • RVU flag not cleared at low APB clock frequency • PVU flag not cleared at low APB clock frequency • **RTC:** RTC calendar registers are not locked properly • RTC interrupt can be masked by another RTC interrupt • Calendar initialization may fail in case of consecutive INIT mode entry • Alarm flag may be repeatedly set when the core is stopped in debug • **I2C:** Spurious bus error detection in master mode • **SPI/I2S:** BSY bit may stay high when SPI is disabled • Anticipated communication upon SPI transit from slave receiver to master • I^2S slave in PCM short pulse mode sensitive to timing between WS and CK • Wrong CRC transmitted in Master mode with delayed SCK feedback Modified errata: • **USART**: RTS signal abnormally driven low after a protocol violation • **SPI/I2S**: BSY flag may stay high at the end of a data transfer in Slave mode Removed errata not applicable for this product: • **IWDG**: RVU and PVU flags are not reset in STOP mode
01-June-2023	4	Sections reordered to match their order in the reference manual. Added errata: • **FMPI2C:** 10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave • Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one FMPI2C kernel clock period • Spurious bus error detection in master mode

Date	Version	Changes
		• Last-received byte loss in reload mode • Spurious master transfer upon own slave address match • OVR flag not set in underrun condition • Transmission stalled after first byte transfer • SDA held low upon SMBus timeout expiry in slave mode • START bit is cleared upon setting ADDRCF, not upon address match (documentation erratum) • Inconsistent FMPI2C peripheral instance naming (documentation erratum) Modified errata: • **DAC:** DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge • **IWDG:** the erratum *RVU and PVU flag not reset in Stop mode* split into two, separate for RVU and PVU • **FMPI2C:** Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one FMPI2C kernel clock period: workaround updated and re-qualified to *P* • **I2C:** Spurious bus error detection in master mode: update • **USART:** Break frame is transmitted regardless of CTS input line status:*nCTS* signal renamed to *CTS* • RTS signal abnormally driven low after a protocol violation and RTS is active while RE or UE = 0: *nRTS* signal renamed to *RTS* Removed errata not applicable for this product: • **SPI/I2S:**Wrong CRC transmitted in Master mode with delayed SCK feedback • I^2S slave in PCM short pulse mode sensitive to timing between WS and CK

Contents