

STM32L462CE/RE/VE device errata

Applicability

This document applies to the part numbers of STM32L462CE/RE/VE 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 RM0394. 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
STM32L462xx	STM32L462CE, STM32L462RE, STM32L462VE

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking ⁽¹⁾	REV_ID ⁽²⁾
STM32L462CE/RE/VE	Y	0x2001

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 STM32L462CE/RE/VE 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. Y
Core	2.1.1	Interrupted loads to SP can cause erroneous behavior	A
	2.1.2	VDIV or VSQRT instructions might not complete correctly when very short ISRs are used	A
	2.1.3	Store immediate overlapping exception return operation might vector to incorrect interrupt	A
System	2.2.1	Internal voltage reference corrupted upon Stop mode entry with temperature sensing enabled	A
	2.2.2	Spurious brown-out reset after short run sequence	A
	2.2.3	Full JTAG configuration without NJTRST pin cannot be used	A
	2.2.4	Current injection from V _{DD} to V _{DDA} through analog switch voltage booster	A
	2.2.5	Unstable LSI when it clocks RTC or CSS on LSE	P
	2.2.6	Possible incorrect code execution when transiting from SRAM2 to Flash memory	A
	2.2.7	Option validity error set after reset	A
	2.2.8	HSI48 ready interrupt capability is not supported	A
	2.2.9	FLASH_ECCR corrupted upon reset or power-down occurring during flash memory program or erase operation	A
	2.2.10	HSE oscillator long startup at low voltage	P
	2.2.11	SRAM write error	A
	2.2.12	PC13 signal transitions disturb LSE	N
FW	2.3.1	Code segment unprotected if non-volatile data segment length is zero	A
QUADSPI	2.4.1	First nibble of data not written after dummy phase	A
	2.4.2	Wrong data from memory-mapped read after an indirect mode operation	A
ADC	2.5.1	Writing ADCx_JSQR when JADCSTART and JQDIS are set might lead to incorrect behavior	N
	2.5.2	Wrong ADC result if conversion done late after calibration or previous conversion	A
	2.5.3	Spurious temperature measurement due to spike noise	A
	2.5.4	Selected external ADC inputs unduly clamped to V _{DD} when all analog peripherals are disabled	A
COMP	2.6.1	Comparator outputs cannot be configured in open-drain	N
TSC	2.7.2	TSC signal-to-noise concern under specific conditions	A

Function	Section	Limitation	Status
			Rev. Y
TIM	2.9.1	HSE/32 is not available for TIM16 input capture if RTC clock is disabled or other than HSE	A
LPTIM	2.10.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	A
	2.10.2	Device may remain stuck in LPTIM interrupt when clearing event flag	P
	2.10.3	LPTIM1 outputs cannot be configured as open-drain	N
RTC and TAMP	2.11.1	RTC calendar registers are not locked properly	A
	2.11.2	RTC interrupt can be masked by another RTC interrupt	A
	2.11.3	Calendar initialization may fail in case of consecutive INIT mode entry	A
	2.11.4	RTC_REFIN and RTC_OUT on PB2 not operating in Stop 2 mode	P
I2C	2.12.1	10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave	A
	2.12.3	Wrong data sampling when data setup time ($t_{SU, DAT}$) is shorter than one I2C kernel clock period	P
	2.12.4	Spurious bus error detection in master mode	A
	2.12.5	Last-received byte loss in reload mode	P
	2.12.6	Spurious master transfer upon own slave address match	P
USART	2.13.1	RTS is active while RE = 0 or UE = 0	A
LPUART	2.14.1	LPUART1 outputs cannot be configured as open-drain	N
SPI	2.15.1	BSY bit may stay high at the end of data transfer in slave mode	A
	2.15.2	Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters	A
	2.15.4	SPI master communication failure at high SYSCLK frequency within the specified voltage range	A
SDMMC	2.16.1	Wrong CCRCFAIL status after a response without CRC is received	A
	2.16.2	MMC stream write of less than seven bytes does not work correctly	A
bxCAN	2.17.1	bxCAN time-triggered communication mode not supported	N

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

Table 4. Summary of device documentation errata

Function	Section	Documentation erratum
TSC	2.7.1	Inhibited acquisition in short transfer phase configuration
AES	2.8.1	Burst read or write accesses not supported
	2.8.2	TAG computation in GCM encryption mode
	2.8.3	CCM authentication mode not compliant with NIST CMAC
	2.8.4	Datatype initial configuration in GCM mode
	2.8.5	Wait until BUSY is low when suspending GCM encryption
RTC and TAMP	2.11.5	Setting GPIO properties of PC13 used as RTC_ALARM open-drain output
I2C	2.12.2	Wrong behavior in Stop mode when wakeup from Stop mode is disabled in I2C
SPI	2.15.3	CRC error in SPI slave mode if internal NSS changes before CRC transfer

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>.

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 VSQRT 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 pending by a level-based interrupt which is cleared by C's handler then interrupt C will be pending 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 Internal voltage reference corrupted upon Stop mode entry with temperature sensing enabled

Description

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

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

The corruption of the internal voltage reference may cause:

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

The limitation applies to Stop 1 and Stop 2 modes.

Workaround

Before entering Stop mode:

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

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

2.2.2 Spurious brown-out reset after short run sequence

Description

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

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

Table 5. Minimum run time

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

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

This limitation applies to Stop 1 and Stop 2 modes.

Workaround

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

2.2.3 Full JTAG configuration without NJTRST pin cannot be used

Description

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

Workaround

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

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

Description

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

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

Workaround

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

2.2.5 Unstable LSI when it clocks RTC or CSS on LSE

Description

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

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

Workaround

Apply one of the following measures:

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

2.2.6 Possible incorrect code execution when transiting from SRAM2 to Flash memory

Description

When transiting from the SRAM2 to the Flash memory, a failure of loading null operands to prefetch buffer may happen, provided that the SRAM2 is accessed through its native address range from 0x1000 0000 to 0x1000 8000. Depending on the instructions placed at the entry point to the Flash memory and on memory alignment, the possible consequence is an incorrect execution or a hard fault.

The instruction in SRAM2 that precedes the potential failure described is:

```
BX Rm ;Jump or return from function call
```

Other types of code execution transiting from SRAM2 to Flash memory, such as other branching instructions or interrupt servicing, do not cause the issue.

Note that inserting wait states or enabling/disabling prefetch in the Flash memory does not prevent this failure from occurring.

Workaround

Insert two NOP instructions in the Flash memory code at any entry point for code transiting from SRAM2 to the Flash memory.

Example 1:

```
BL funC ;Call of funC located in SRAM2
NOP ;Entry point - return from call of funC function
NOP
```

Example 2:

```
Any instruction
NOP ;Entry point - target of a jump from SRAM2
NOP
```


2.2.7 Option validity error set after reset

Description

On first production lot, the complement of one word inside the device configuration area is not correctly programmed. Although this word is not used for the device configuration, it is included in the test of configuration consistency after reset. As a consequence, the OPTVERR flag of the FLASH_SR register is set.

When using HAL drivers included in STM32CubeL4, this bit is tested prior to executing any Flash memory command and as a consequence, the FLASH_WaitForLastOperation() function fails.

Note: This limitation is fixed on parts with the production date code 721 (inclusive) and later.

Workaround

As this limitation has no impact to the device functionality, the OPTVERR flag may be ignored or cleared.

When using HAL, calling the __HAL_FLASH_CLEAR_FLAG(FLASH_FLAG_OPTVERR) function after reset allows clearing the OPTVERR flag and then using HAL Flash memory access commands normally.

2.2.8 HSI48 ready interrupt capability is not supported

Description

HSI48 ready interrupt feature described in the reference manual is not supported. The bit HSI48RDYIE in the register RCC_CIER is stuck at 0. It is not possible to set it to interrupt the CPU when the oscillator is ready.

Workaround

After switching the HSI48 internal oscillator on by setting the bit HSI48ON of the RCC_CRRCR register, poll the HSI48RDY flag of the RCC_CRRCR register until it goes high, which indicates that the HSI48 oscillator is ready to provide the clock to peripherals.

2.2.9 FLASH_ECCR corrupted upon reset or power-down occurring during flash memory program or erase operation

Description

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

Workaround

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

2.2.10 HSE oscillator long startup at low voltage

Description

When V_{DD} is below 2.7 V, the HSE oscillator may take longer than specified to start up. Several hundred milliseconds might elapse before the HSERDY flag in the RCC_CR register is set.

Workaround

The following sequence is recommended:

1. Configure PH0 and PH1 as standard GPIOs in output mode and low-level state.
2. Enable the HSE oscillator.

2.2.11 SRAM write error

Description

In rare cases, system reset occurring in a critical instant may upset the SRAM state machine. The first SRAM read or write access after the system reset then restores the normal operation of the SRAM for any subsequent accesses. However, if it is a write access, it fails to write data.

Workaround

Upon system reset, make a dummy read access to each 32-Kbyte SRAM instance used by the application, through one of the following methods:

1. Place a dummy variable initialization, which allows the compiler to create the assembly code.
2. Use this initialization assembly code:

```
MOV32  R0, #0x20000000 //SRAM1 address
LDR    R1, [R0, #+0]  //read access
MOV32  R0, #0x20008000 //next SRAM1 instance
LDR    R1, [R0, #+0]  //dummy read, no consequence on R1 value
...
MOV32  R0, #0x20020000 //SRAM2 address
LDR    R1, [R0, #+0]
...
```

Follow the first method for SRAM instances with the parity check enabled. Follow the first or the second method for SRAM instances with the parity check disabled.

2.2.12 PC13 signal transitions disturb LSE

Description

The PC13 port toggling disturbs the LSE clock.

Workaround

None.

2.3 FW

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

Description

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

Workaround

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

2.4 QUADSPI

2.4.1 First nibble of data not written after dummy phase

Description

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

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

Workaround

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

Example:

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

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

Description

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

Workaround

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

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

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

2.5 ADC

2.5.1 Writing ADCx_JSQR when JADCSTART and JQDIS are set might lead to incorrect behavior

Description

Writing the ADCx_JSQR register when there is an on-going injected conversion (JADCSTART = 1) might lead to unpredictable ADC behavior if the queues of context are not enabled (JQDIS = 1).

Workaround

None.

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

Description

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

Workaround

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

2.5.3 Spurious temperature measurement due to spike noise

Description

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

Workaround

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

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

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

Example:

Table 6. Measurement result after IQM post-processing

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

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

2.5.4 Selected external ADC inputs unduly clamped to V_{DD} when all analog peripherals are disabled

Description

When all analog peripherals (other than VREFBUF) are disabled, the GPIO(s) selected as ADC input(s) are unduly clamped (through a parasitic diode) to V_{DD} instead to V_{DDA} . As a consequence, the input voltage is limited to $V_{DD} + 0.3 V$ even if V_{DDA} is higher than $V_{DD} + 0.3 V$.

Note: The selection of GPIOs as ADC inputs is done with the SQy and JSQy bitfields of the ADC_SQRx and ADC_JSQR registers, respectively.

VREFBUF enable/disable has no impact to the issue described.

Workaround

Apply one of the following measures:

- Use V_{DDA} lower than $V_{DD} + 0.3 V$.
- Keep at least one analog peripheral (other than VREFBUF) enabled if GPIOs are selected as ADC inputs.
- Deselect GPIOs as ADC inputs (by clearing ADC_SQRx or/and ADC_JSQR registers) when no analog peripheral (other than VREFBUF) is enabled.

2.6 COMP

2.6.1 Comparator outputs cannot be configured in open-drain

Description

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

Workaround

None.

2.7 TSC

2.7.1 Inhibited acquisition in short transfer phase configuration

Description

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

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

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

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

Workaround

No application workaround is required.

2.7.2 TSC signal-to-noise concern under specific conditions

Description

V_{DD} equal to or greater than V_{DDA} may lead (depending on part) to some degradation of the signal-to-noise ratio on the TSC analog I/O group 2.

The lower are the sampling capacitor (C_S) and the sensing electrode (C_X) capacitances, the worse is the signal-to-noise ratio degradation.

Workaround

Apply one of the following measures:

- Maximize C_S capacitance.
- Use the analog I/O group 2 as active shield.

2.8 AES

2.8.1 Burst read or write accesses not supported

Description

Some revisions of the reference manual may omit the information that the AES peripheral does not support LDM, STM, LDRD and STRD instructions for successive multiple-data (burst) read and write accesses to a contiguous address block.

This is a documentation issue rather than a product limitation.

Workaround

No application workaround is required, provided that the multiple-data instructions are not used to access the AES peripheral.

Note: To prevent compilers from generating LDM, STM, LDRD and STRD instructions to access the AES peripheral, organize the source code such as to avoid consecutive read or write accesses to neighboring addresses in lower-to-higher order. In case where consecutive read or write accesses to neighboring addresses cannot be avoided, order the source code such as to access higher address first.

2.8.2 TAG computation in GCM encryption mode

Description

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

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

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

This is a documentation issue rather than a product limitation.

Workaround

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

2.8.3 CCM authentication mode not compliant with NIST CMAC

Description

Some revisions of the reference manual may omit the information that setting the CHMOD[2:0] bitfield to 100 selects the CCM authentication mode. It is not compliant with NIST CMAC, as defined in Special Publication 800-38B. In order to fully implement the CCM chaining as specified in NIST Special Publication 800-38C, the payload must first be encrypted or decrypted with the AES peripheral set in CTR mode (CHMOD[2:0] = 010), then the message associated data and payload authenticated with the AES peripheral set in CCM mode (CHMOD[2:0] = 100).

This is a documentation issue rather than a product limitation.

Workaround

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

2.8.4 Datatype initial configuration in GCM mode

Description

Some revisions of the reference manual may omit the information that GCM generated TAG is not correct if data swapping is not disabled in GCM init phase (GCMPH[1:0] = 00) by setting the DATATYPE[1:0] bitfield of the AES_CR register to zero.

This is a documentation issue rather than a product limitation.

Workaround

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

2.8.5 Wait until BUSY is low when suspending GCM encryption

Description

Some revisions of the reference manual may omit the information that when suspending the GCM encryption of a message, in the payload phase the BUSY flag of the AES_SR register must be low before saving the AES_SUSPxR registers in the memory.

This is a documentation issue rather than a product limitation.

Workaround

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

2.9 TIM

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

Description

If the RTC clock is either disabled or other than HSE, the HSE/32 clock is not available for TIM16 input capture even if selected (bitfield T11_RMP[2:0] = 101 in the TIM16_OR1 register).

Workaround

Apply the following procedure:

1. Enable the power controller clock (bit PWREN = 1 in the RCC_APB1ENR1 register).
2. Disable the backup domain write protection (bit DBP = 0 in the PWR_CR1 register).
3. Enable RTC clock and select HSE as clock source for RTC (bits RTCSEL[1:0] = 11 and bit RTCEN = 1 in the RCC_BDCR register).
4. Select the HSE/32 as input capture source for TIM16 (bitfield T11_RMP[2:0] = 101 in the TIM16_OR1 register).

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

2.10 LPTIM

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

Description

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

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

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

Workaround

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

2.10.2 Device may remain stuck in LPTIM interrupt when clearing event flag

Description

This limitation occurs when the LPTIM is configured in interrupt mode (at least one interrupt is enabled) and the software clears any flag in LPTIM_ISR register by writing its corresponding bit in LPTIM_ICR register. If the interrupt status flag corresponding to a disabled interrupt is cleared simultaneously with a new event detection, the set and clear commands might reach the APB domain at the same time, leading to an asynchronous interrupt signal permanently stuck high.

This issue can occur either during an interrupt subroutine execution (where the flag clearing is usually done), or outside an interrupt subroutine.

Consequently, the firmware remains stuck in the LPTIM interrupt routine, and the device cannot enter Stop mode.

Workaround

To avoid this issue, it is strongly advised to follow the recommendations listed below:

- Clear the flag only when its corresponding interrupt is enabled in the interrupt enable register.
- If for specific reasons, it is required to clear some flags that have corresponding interrupt lines disabled in the interrupt enable register, it is recommended to clear them during the current subroutine prior to those which have corresponding interrupt line enabled in the interrupt enable register.
- Flags must not be cleared outside the interrupt subroutine.

Note: The standard clear sequence implemented in the `HAL_LPTIM_IRQHandler` in the `STM32Cube` is considered as the proper clear sequence.

2.10.3 LPTIM1 outputs cannot be configured as open-drain

Description

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

Workaround

None.

2.11 RTC and TAMP

2.11.1 RTC calendar registers are not locked properly

Description

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

Workaround

Apply one of the following measures:

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

2.11.2 RTC interrupt can be masked by another RTC interrupt

Description

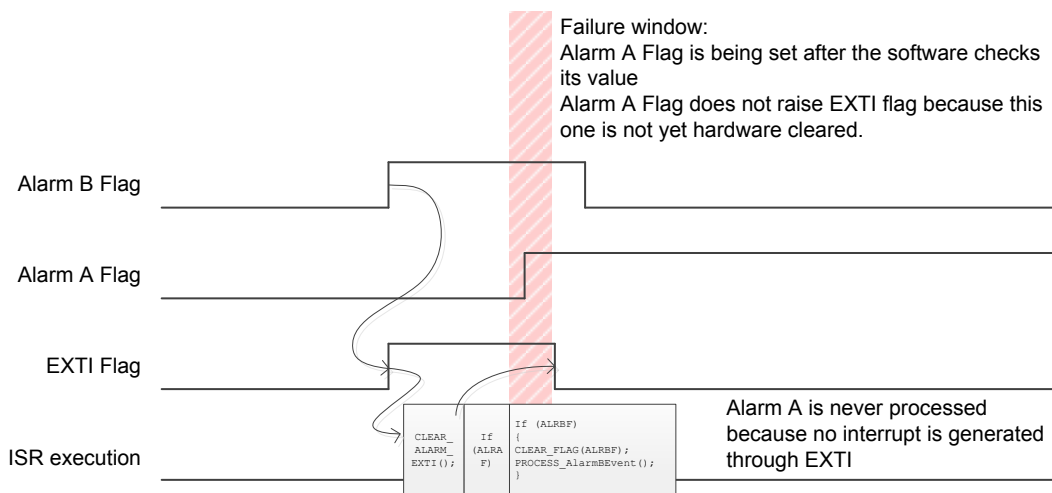
One RTC interrupt request can mask another RTC interrupt request if they share the same EXTI configurable line. For example, interrupt requests from Alarm A and Alarm B or those from tamper and timestamp events are OR-ed to the same EXTI line (refer to the *EXTI line connections* table in the *Extended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

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

Alarm interrupt service routine:

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

Figure 1. Masked RTC interrupt



DT14477V1

Workaround

In the interrupt service routine, apply three consecutive event flag checks - 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 */
    }
}
```

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

Description

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

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

Workaround

After exiting 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.

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

Description

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

Workaround

Apply one of the following measures:

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

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

Description

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

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

Workaround

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

2.12 I2C

2.12.1 10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave

Description

An I²C-bus master generates STOP condition upon non-acknowledge of I²C 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 I²C-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 I²C-bus transfer. In this spurious state, the NACKF flag of the I2C_ISR register and the START bit of the I2C_CR2 register are both set, while the START bit should normally be cleared.

Workaround

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

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

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

Description

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

Some reference manual revisions may omit this information.

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

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

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

Workaround

No application workaround is required.

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

Description

The I²C-bus specification and user manual specify a minimum data setup time ($t_{SU,DAT}$) as:

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

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

Workaround

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

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

2.12.4 Spurious bus error detection in master mode

Description

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

Workaround

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

2.12.5 Last-received byte loss in reload mode

Description

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

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

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

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

Workaround

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

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

2.12.6 Spurious master transfer upon own slave address match

Description

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

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

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

Workaround

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

Normal mode (SBC = 0):

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

Slave byte control mode (SBC = 1):

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

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

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

2.13 USART

2.13.1 RTS is active while RE = 0 or UE = 0

Description

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

Workaround

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

2.14 LPUART

2.14.1 LPUART1 outputs cannot be configured as open-drain

Description

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

Workaround

None.

2.15 SPI

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

Description

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

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

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

Workaround

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

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

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

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

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

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

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

Description

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

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

Workaround

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

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

2.15.3 CRC error in SPI slave mode if internal NSS changes before CRC transfer

Description

Some reference manual revisions may omit the information that the device operating as SPI slave must be configured in software NSS control if the SPI master pulses the NSS (for example in NSS pulse mode). Otherwise, the transition of the internal NSS signal after the CRCNEXT flag is set might result in wrong CRC value computed by the device and, as a consequence, in a CRC error. As a consequence, the NSS pulse mode cannot be used along with the CRC function.

This is a documentation error rather than a product limitation.

Workaround

No application workaround is required as long as the device operating as SPI slave is duly configured in software NSS control.

2.15.4 SPI master communication failure at high SYSCLK frequency within the specified voltage range

Description

The SPI peripheral configured as master may spuriously generate an extra clock pulse when:

- the CPHA bit of the SPIx_CR1 register is set, and
- the BR[2:0] bitfield of the SPIx_CR1 register is at 001 (f_{PCLK} divided by four), and
- the SPI clock is about to be suspended or stopped (which occurs at the end of a data frame), and
- the SYSCLK frequency exceeds the values as per the following table

This leads to a de-synchronization of the SPI data flow.

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

SPI instance	$f_{SYSCLK(max)}$ [MHz]	
	Range 1	Range 2
SPI1	>80 (OK)	>26 (OK)
SPI2	69.4	24.5
SPI3	>80 (OK)	>26 (OK)

Workaround

Depending on the application constraints, apply one of the following measures:

- Set the SYSCLK frequency to within the limits in the table.
- Choose a combination of AHB and APB prescaler settings that allows setting the BR[2:0] bitfield to a value other than 001 ($f_{PCLK}/4$).
- Use another SPI instance on another APB if this can avoid setting BR[2:0] to 001. If the other instance is available on the same pins through the GPIO alternate function, the same application hardware can be used.

2.16 SDMMC

2.16.1 Wrong CCRCFAIL status after a response without CRC is received

Description

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

Workaround

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

2.16.2 MMC stream write of less than seven bytes does not work correctly

Description

Stream write initiated with WRITE_DAT_UNTIL_STOP command (CMD20) does not define the amount of data bytes to store. The card keeps storing data coming in from the SDMMC host until it gets a valid STOP_TRANSMISSION (CMD12) command. The commands are streamed on a line separate from data line, with common clock line.

As the STOP_TRANSMISSION command is 48-bit long and due to the bus protocol, the STOP_TRANSMISSION command start bit must be advanced by 50 clocks with respect to the stop bit of the data bitstream.

Therefore, for small data chunks of up to six bytes, SDMMC hosts should normally operate such that, the start of the STOP_TRANSMISSION (CMD12) command streaming precedes the start of the data streaming.

The device duly anticipates the STOP_TRANSMISSION command streaming start, with respect to the data bitstream end. WAITPEND (bit 9 of SDMMC_CMD register) must be set for this mechanism to operate.

However, a failure occurs in case of small data chunks of up to six bytes. Instead of starting the STOP_TRANSMISSION command 50 clocks ahead of the data bitstream stop bit, the SDMMC peripheral on the device starts the command along with the first bit of the data bitstream. As the command is longer than the data, it ends a number of clocks behind the data that the software intended to store onto the card by setting the DATALENGTH register. During the clocks in excess, the SDMMC peripheral keeps the data line in logical-one level.

As a consequence, the card intercepts more data and updates more memory locations than the number set in DATALENGTH. The spuriously updated locations of memory receive 0xFF values.

Workaround

Do not use stream write WRITE_DAT_UNTIL_STOP command (CMD20) with DATALENGTH set to less than seven. Instead, use SET_BLOCKLEN command (CMD16) followed with single-block write command WRITE_BLOCK (CMD24), with a desired block length.

2.17 bxCAN

2.17.1 bxCAN time-triggered communication mode not supported

Description

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

Workaround

None.

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 8. Document revision history

Date	Version	Changes
10-Feb-2017	1	Initial release.
30-Mar-2017	2	Change of document classification from ST Restricted to Public.
26-May-2017	3	Added section <i>Option validity error set after reset</i>
07-Sep-2017	4	Added section <i>Possible incorrect code execution when transiting from SRAM2 to Flash memory</i> Removed obsolete STM32L462SL part number and the table "Device summary" (all represented part numbers now contained in the product name).
07-Dec-2018	5	<p>Added:</p> <ul style="list-style-type: none"> • Table of documentation errata • VDIV or VSQRT instructions might not complete correctly when very short ISRs are used • Store immediate overlapping exception return operation might vector to incorrect interrupt • Spurious brown-out reset after short run sequence • Unstable LSI when it clocks RTC or CSS on LSE • First double-word of Flash memory corrupted upon reset or power down while programming • Code segment unprotected if non-volatile data segment length is zero • Writing ADCx_JSQR when JADCSTART and JQDIS are set might lead to incorrect behavior • CCM authentication mode not compliant with NIST CMAC • Datatype initial configuration in GCM mode • Wait until BUSY is low when suspending GCM encryption • MCU may remain stuck in LPTIM interrupt when entering Stop mode • MCU may remain stuck in LPTIM interrupt when clearing event flag • RTC section with four limitations • Last-received byte loss in reload mode • Spurious master transfer upon own slave address match • RTS is active while RE = 0 or UE = 0 • Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters • CRC error in SPI slave mode if internal NSS changes before CRC transfer <p>Modified:</p> <ul style="list-style-type: none"> • HSI48 ready interrupt capability is not supported moved to section System • Wrong data from memory-mapped read after an indirect mode operation • Wrong ADC result if conversion done late after calibration or previous conversion • TAG computation in GCM encryption mode • Wrong behavior in Stop mode when wakeup from Stop mode is disabled in I2C moved to the documentation errata
03-Jun-2021	6	<p>Added errata:</p> <ul style="list-style-type: none"> • FLASH_ECCR corrupted upon reset or power-down occurring during flash memory program or erase operation • HSE oscillator long startup at low voltage • Selected external ADC inputs unduly clamped to V_{DD} when all analog peripherals are disabled • TSC signal-to-noise concern under specific conditions • SPI master communication failure at high SYSCLK frequency within the specified voltage range <p>Modified errata:</p>

Date	Version	Changes
		<ul style="list-style-type: none"> • VDIV or VSQRT instructions might not complete correctly when very short ISRs are used • Internal voltage reference corrupted upon Stop mode entry with temperature sensing enabled • Full JTAG configuration without NJTRST pin cannot be used • Inhibited acquisition in short transfer phase configuration • Device may remain stuck in LPTIM interrupt when entering Stop mode • Device may remain stuck in LPTIM interrupt when clearing event flag • RTC interrupt can be masked by another RTC interrupt • Wrong behavior in Stop mode when wakeup from Stop mode is disabled in I2C • Wrong data sampling when data setup time ($t_{SU, DAT}$) is shorter than one I2C kernel clock period • bxCAN time-triggered communication mode not supported (workaround qualifier) <p>Removed erratum <i>First double-word of Flash memory corrupted upon reset or power down while programming</i> (superseded with the erratum FLASH_ECCR corrupted upon reset or power-down occurring during flash memory program or erase operation).</p>
19-May-2022	7	<p>Added erratum SRAM write error.</p> <p>Added Section Important security notice.</p>
14-Jun-2022	8	<p>Updated erratum SRAM write error - correction of the SRAM instance size.</p>
23-Feb-2023	9	<p>Added erratum PC13 signal transitions disturb LSE.</p> <p>Modified erratum:</p> <ul style="list-style-type: none"> • SRAM write error • SPI master communication failure at high SYSCLK frequency within the specified voltage range - now applicable to high f_{SYSCLK} condition not only to the specific case of high $f_{PCLK} = f_{SYSCLK}$. Re-qualified to A and a workaround described.

Contents

1	Summary of device errata	2
2	Description of device errata	4
2.1	Core	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	System	6
2.2.1	Internal voltage reference corrupted upon Stop mode entry with temperature sensing enabled	6
2.2.2	Spurious brown-out reset after short run sequence	6
2.2.3	Full JTAG configuration without NJTRST pin cannot be used	7
2.2.4	Current injection from V_{DD} to V_{DDA} through analog switch voltage booster	7
2.2.5	Unstable LSI when it clocks RTC or CSS on LSE	8
2.2.6	Possible incorrect code execution when transiting from SRAM2 to Flash memory	8
2.2.7	Option validity error set after reset	9
2.2.8	HSI48 ready interrupt capability is not supported	9
2.2.9	FLASH_ECCR corrupted upon reset or power-down occurring during flash memory program or erase operation	9
2.2.10	HSE oscillator long startup at low voltage	9
2.2.11	SRAM write error	10
2.2.12	PC13 signal transitions disturb LSE	10
2.3	FW	10
2.3.1	Code segment unprotected if non-volatile data segment length is zero	10
2.4	QUADSPI	10
2.4.1	First nibble of data not written after dummy phase	10
2.4.2	Wrong data from memory-mapped read after an indirect mode operation	11
2.5	ADC	11
2.5.1	Writing ADCx_JSQR when JADCSTART and JQDIS are set might lead to incorrect behavior	11
2.5.2	Wrong ADC result if conversion done late after calibration or previous conversion	11
2.5.3	Spurious temperature measurement due to spike noise	11
2.5.4	Selected external ADC inputs unduly clamped to V_{DD} when all analog peripherals are disabled	12
2.6	COMP	12
2.6.1	Comparator outputs cannot be configured in open-drain	12
2.7	TSC	13
2.7.1	Inhibited acquisition in short transfer phase configuration	13

2.7.2	TSC signal-to-noise concern under specific conditions	13
2.8	AES	13
2.8.1	Burst read or write accesses not supported	13
2.8.2	TAG computation in GCM encryption mode	13
2.8.3	CCM authentication mode not compliant with NIST CMAC	14
2.8.4	Datatype initial configuration in GCM mode	14
2.8.5	Wait until BUSY is low when suspending GCM encryption	14
2.9	TIM	15
2.9.1	HSE/32 is not available for TIM16 input capture if RTC clock is disabled or other than HSE	15
2.10	LPTIM	15
2.10.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	15
2.10.2	Device may remain stuck in LPTIM interrupt when clearing event flag	15
2.10.3	LPTIM1 outputs cannot be configured as open-drain	16
2.11	RTC and TAMP	16
2.11.1	RTC calendar registers are not locked properly	16
2.11.2	RTC interrupt can be masked by another RTC interrupt	16
2.11.3	Calendar initialization may fail in case of consecutive INIT mode entry	18
2.11.4	RTC_REFIN and RTC_OUT on PB2 not operating in Stop 2 mode	18
2.11.5	Setting GPIO properties of PC13 used as RTC_ALARM open-drain output	18
2.12	I2C	18
2.12.1	10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave	18
2.12.2	Wrong behavior in Stop mode when wakeup from Stop mode is disabled in I2C	19
2.12.3	Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period	19
2.12.4	Spurious bus error detection in master mode	20
2.12.5	Last-received byte loss in reload mode	20
2.12.6	Spurious master transfer upon own slave address match	20
2.13	USART	21
2.13.1	RTS is active while RE = 0 or UE = 0	21
2.14	LPUART	21
2.14.1	LPUART1 outputs cannot be configured as open-drain	21
2.15	SPI	21
2.15.1	BSY bit may stay high at the end of data transfer in slave mode	21
2.15.2	Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters	22
2.15.3	CRC error in SPI slave mode if internal NSS changes before CRC transfer	22

2.15.4	SPI master communication failure at high SYSCLK frequency within the specified voltage range	23
2.16	SDMMC	23
2.16.1	Wrong CCRCFAIL status after a response without CRC is received	23
2.16.2	MMC stream write of less than seven bytes does not work correctly	23
2.17	bxCAN	24
2.17.1	bxCAN time-triggered communication mode not supported	24
Important security notice		25
Revision history		26

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2023 STMicroelectronics – All rights reserved