life.augmented

STM32L07xxx STM32L08xxx

Errata sheet

STM32L07xxx/L08xxx device errata

Applicability

This document applies to the part numbers of STM32L07xxx/L08xxx 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 RM0367/ RM0376/RM0377.

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
STM32L071x8/B/Z	STM32L071C8, STM32L071V8, STM32L071K8, STM32L071CB, STM32L071VB, STM32L071RB, STM32L071KB, STM32L071CZ, STM32L071VZ, STM32L071RZ, STM32L071KZ
STM32L072x8/B/Z	STM32L072V8, STM32L072VB, STM32L072RB, STM32L072CB, STM32L072KB, STM32L072VZ, STM32L072RZ, STM32L072CZ, STM32L072KZ
STM32L073x8/B/Z	STM32L073V8, STM32L073VB, STM32L073RB, STM32L073CB, STM32L073VZ, STM32L073RZ, STM32L073CZ
STM32L081xZ	STM32L081CB, STM32L081CZ, STM32L081KZ
STM32L082xB/Z	STM32L082KB, STM32L082KZ, STM32L082CZ
STM32L083x8/B/Z	STM32L083V8, STM32L083CB, STM32L083VB, STM32L083RB, STM32L083CZ, STM32L083VZ, STM32L083RZ

Table 2. **Device variants**

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32L071xx/72xx/73xx/81xx/82xx/83xx	A	0x1000
	B	0x2000
	1, P, Q, Z	0x2008

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

1 Summary of device errata

The following table gives a quick reference to the STM32L07xxx/L08xxx 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. B	Rev. 1, P, Q, Z
System	2.1.1	Delay after an RCC peripheral clock enabling	A	A	A
	2.1.2	Flash memory wakeup issue when waking up from Stop or Sleep with Flash in power-down mode	A	-	-
	2.1.3	Unexpected system reset when waking up from Stop mode with regulator in low-power mode	A	-	-
	2.1.4	I2C and USART cannot wake up the device from Stop mode	N	N	-
	2.1.5	LDM, STM, PUSH and POP not allowed in IOPORT bus	N	N	N
	2.1.6	BOOT_MODE bits do not reflect the selected boot mode	N	N	-
	2.1.7	Electrical sensitivity characteristics lower compared to other STM32 devices	N	N	N
	2.1.8	BFB2 bit does not select boot from Bank 2 system memory	A	A	-
	2.1.9	Interrupts masked when using dual-bank boot mechanism	A	A	A
GPIO	2.2.1	PE2 AF2 alternate function (TIM3_ETR) not available	A	-	-
	2.2.2	PB6, PB7, PD0, PD1, PD7 AF1 alternate functions not available when LCD controller is enabled	N	-	-
	2.2.3	PB4 not available on UFQFPN32 package	N	-	-
DMA	2.3.1	DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	A	A	A
ADC	2.4.1	Overrun flag is not set if EOC reset coincides with new conversion end	P	P	P
	2.4.2	Writing ADC_CFGR1 register while ADEN bit is set resets RES[1:0] bitfield	A	A	A
	2.4.3	Out-of-threshold value is not detected in AWD1 Single mode	A	A	A
DAC	2.5.1	DMA request not automatically cleared by clearing DMAEN	A	A	A
	2.5.2	DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge	N	N	N
COMP	2.6.1	COMP1_CSR and COMP2_CSR lock bit reset by SYSCFGRST bit in RCC_APB2RSTR register	N	N	N
	2.6.2	Output of comparator 2 cannot be internally connected to input 1 of low-power timer	A	-	-
TSC	2.7.2	Touch sensing channels not available in dynamic voltage range 3	A	A	A

Function	Section	Limitation	Status		
			Rev. A	Rev. B	Rev. 1, P, Q, Z
TIM	2.9.1	PWM re-enabled in automatic output enable mode despite of system break	P	P	P
	2.9.3	Consecutive compare event missed in specific conditions	N	N	N
	2.9.4	Output compare clear not working with external counter reset	P	P	P
LPTIM	2.10.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	A	A	A
	2.10.2	Device may remain stuck in LPTIM interrupt when clearing event flag	P	P	P
	2.10.3	LPTIM events and PWM output are delayed by 1 kernel clock cycle	P	P	P
RTC and TAMP	2.11.1	Spurious tamper detection when disabling the tamper channel	N	N	N
	2.11.2	RTC calendar registers are not locked properly	A	A	A
	2.11.3	RTC interrupt can be masked by another RTC interrupt	A	A	A
	2.11.4	Calendar initialization may fail in case of consecutive INIT mode entry	A	A	A
	2.11.5	Alarm flag may be repeatedly set when the core is stopped in debug	N	N	N
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	A	A
	2.12.3	Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period	P	P	P
	2.12.4	Spurious bus error detection in master mode	A	A	A
	2.12.5	Last-received byte loss in reload mode	P	P	P
	2.12.6	Spurious master transfer upon own slave address match	P	P	P
	2.12.8	OVR flag not set in underrun condition	N	N	N
	2.12.9	Transmission stalled after first byte transfer	A	A	A
USART	2.13.1	Non-compliant sampling for NACK signal from smartcard	N	N	N
	2.13.2	Break request preventing TC flag from being set	A	A	A
	2.13.3	RTS is active while RE = 0 or UE = 0	A	A	A
	2.13.4	Receiver timeout counter wrong start in two-stop-bit configuration	A	A	A
	2.13.5	Data corruption due to noisy receive line	N	N	N
LPUART	2.14.1	Break request preventing TC flag from being set	A	A	A
	2.14.2	RTS is active while RE = 0 or UE = 0	A	A	A
	2.14.3	Receiver timeout counter wrong start in two-stop-bit configuration	A	A	A
	2.14.4	Data corruption due to noisy receive line	N	N	N
SPI/I2S	2.15.1	BSY bit may stay high when SPI is disabled	A	A	A
	2.15.2	BSY bit may stay high at the end of data transfer in slave mode	A	A	A
	2.15.3	Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback	A	A	A
	2.15.4	SPI CRC corruption upon DMA transaction completion by another peripheral	P	P	P
	2.15.5	In I2S slave mode, enabling I2S while WS is active causes desynchronization	A	A	A
	2.15.6	Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters	A	A	A

Function	Section	Limitation	Status		
			Rev. A	Rev. B	Rev. 1, P, Q, Z
SPI/I2S	2.15.8	Anticipated communication upon SPI transit from slave receiver to master	A	A	A
	2.15.9	I2S slave in PCM short pulse mode sensitive to timing between WS and CK	P	P	P
	2.15.10	Limited SPI frequency when peripheral is configured in Master reception or in Slave transmission mode and VDD is below 2.7 V	N	-	-
USB	2.16.2	ESOF interrupt timing desynchronized after resume signaling	A	A	A
	2.16.3	Incorrect CRC16 in the memory buffer	N	N	N
	2.16.4	USB BCD functionality limited below –20 °C	N	N	N

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

Table 4. **Summary of device documentation errata**

Function	Section	Documentation erratum
DMA	2.3.2	Byte and half-word accesses not supported
TSC	2.7.1	Inhibited acquisition in short transfer phase configuration
RNG	2.8.1	RNG clock error does not stop random numbers
TIM	2.9.2	TRGO and TRGO2 trigger output failure
I2C	2.12.2	Wrong behavior in Stop mode
	2.12.7	START bit is cleared upon setting ADDRCF, not upon address match
SPI/I2S	2.15.7	CRC error in SPI slave mode if internal NSS changes before CRC transfer
USB	2.16.1	Possible packet memory overrun/underrun at low APB frequency

2 Description of device errata

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

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

2.1 System

2.1.1 Delay after an RCC peripheral clock enabling

Description

A delay between an RCC peripheral clock enable and the effective peripheral enabling should 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 AHB: the delay should be equal to 1 AHB clock cycle after the clock enable bit is set in the hardware register.

  For I/O peripheral, the delay should be equal to 1 AHB clock cycle after the clock enable bit is set in the hardware register (only applicable to write accesses).
- If the peripheral is mapped on APB: No delay is necessary (no limitation).

Workaround

- Enable the peripheral clock some time before the peripheral read/write register is required.
- For AHB peripheral (including I/O), insert a dummy read operation to the corresponding register.

2.1.2 Flash memory wakeup issue when waking up from Stop or Sleep with Flash in power-down mode

Description

When an external wakeup event (EXTI) occurs in a narrow time window around low-power mode entry (Stop or Sleep mode with Flash memory in power-down state), the Flash wakeup time may be increased. As a result, the first data read or instruction fetch from Flash may be incorrect.

The probability that this issue occurs is very low since it may happen only during a very narrow time window.

Workaround

Three workarounds are available:

- Do not put the Flash memory module in power-down mode when entering Sleep or Low-power sleep modes.
- Before entering Stop mode by executing a WFI instruction from RAM, set the RUN_PD bit in the FLASH_ACR register. After exiting from Stop mode, the Flash memory is automatically powered ON and you can resume program execution from Flash memory. After wakeup, clear the RUN_PD bit.
- Before entering Stop mode by executing WFI instruction from RAM, set the RUN_PD bit in the FLASH_ACR register and set the DS_EE_KOFF bit in PWR_CR register. After resuming from Stop mode, the Flash memory stays in power-down mode. Wake-up the Flash memory by clearing FLASH_ACR_RUN_PD bit and return to code execution.

2.1.3 Unexpected system reset when waking up from Stop mode with regulator in low-power mode

Description

When the device returns to Run mode after waking up from Stop mode while the internal voltage regulator is configured to switch to low-power mode in Stop mode (LPSDSR = 1 in PWR_CR register), an unexpected system reset may occur if the following conditions are met:

- The internal regulator is set to Range 2 or Range 3 before entering Stop mode.
- V_{DD} power supply is below 2.7 V.

The probability that this issue occurs is very low since it may happen only for very narrow supply voltage windows which vary from one device to another.

This reset is internal only and does not affect the NRST pin state and the flags in the Control/status register (RCC_CSR).

Workaround

Two workarounds are possible:

- Enter Stop mode with the internal voltage regulator set to main mode (LPSDSR=0 in PWR_CR).
- Set the internal voltage regulator to Range1 before entering Stop mode.

2.1.4 I2C and USART cannot wake up the device from Stop mode

Description

When the microcontroller is in Stop mode with the regulator in low-power mode, an unexpected system reset may occur if the I2C or the USART attempts to wake up the device.

This limitation also impacts LPUART when the HSI16 is used as clock source instead of LSE.

This reset is internal only and does not affect the NRST pin state and the flags in the Control/status register (RCC_CSR).

The lower the V_{DD} value, the more often this unpredictable behavior may occur.

Workaround

None.

It is recommended to avoid using the USART and I2C wakeup from Stop mode features. To disable them, keep WUPEN bit in I2C_CR1 and UESM bit in USARTx_CR1 at 0.

Two solutions are then possible to perform I2C or USART communications:

- Put the microcontroller in a mode different from Stop (or Standby mode) before initiating communications.
- Replace Stop mode with Stop mode plus regulator in main mode by keeping LPSDSR bit of PWR_CR set to 0.

2.1.5 LDM, STM, PUSH and POP not allowed in IOPORT bus

Description

The instructions Load Multiple (LM), Store Multiple (STM), PUSH and POP fail when the address points to the IOPORT bus memory area (address range = 0x5XXX XXXX).

Workaround

None.

2.1.6 BOOT_MODE bits do not reflect the selected boot mode

Description

The BOOT_MODE[1:0] bits of the SYSCFG_CFGR1 register remain set to ‘0’ while they should reflect the boot mode selected by the boot pins.

Workaround

None.

2.1.7 Electrical sensitivity characteristics lower compared to other STM32 devices

Description

The ESD absolute maximum ratings are lower compared to some other STM32 devices:

- Electrostatic discharge voltage human body model ($V_{ESD(HBM)}$) class is 1C instead of 2 and the maximum value is 1000 V instead of 2000 V.
- Electrostatic discharge voltage charge device model ($V_{ESD(CDM)}$) class is C3 and the maximum value is 250 V instead of 500 V.

Workaround

None.

2.1.8 BFB2 bit does not select boot from Bank 2 system memory

Description

The feature which enables booting either from Bank 2 or from Bank 1 depending on BFB2 bit status (bit 23 in FLASH_OPTR register) does not work properly. When BFB2 user option bit is set, the code behavior after reset is incorrect.

Workaround

BFB2 user option bit must be kept at 0. The user code stored in Bank 1 must jump to System memory if it requires a boot from Bank 2 and intends to benefit from bank swapping mechanism. Otherwise the code in Bank 1 can jump directly to Bank 2 when required.

Below a proposal of code that uses a value stored in EEPROM to define if the code to be executed is located in Bank 1 or 2. This code can be used in both banks and the EEPROM data must be the same in both EEPROM bank. This code shall be placed at the beginning of SystemInit() function:

```
#define DATA_VALUE_SELECT           0x08080000 /* Location where to store the value allowing
to boot on Bank2 and Bank1 */
#define DATA_BANK2_SELECTED         0x37       /* This value indicates that boot from Bank2
is required */
#define DATA_BANK1_SELECTED         0x73       /* This value indicates that boot from Bank1
is required */

/* Non user defines: these values must not be modified by user */
#define BANK2_START_ADDRESS_192KB  0x08018000  /* Bank 2 Start Address (192KB) */
#define BANK2_START_ADDRESS_128KB  0x08010000  /* Bank 2 Start Address (128KB) */
#define FLASH_START_ADDRESS        0x08000000  /* Flash Start Address (Bank1) */

#define ENGI_SALESTYPE_ADDRESS     0x1FF80034
#define ENGI_SALESTYPE_MASK        0x00000C00
#define ENGI_SALESTYPE_192KB       0x00000000
#define ENGI_SALESTYPE_128K        0x00000400
#define ENGI_SALESTYPE_64K         0x00000C00

#define SALESTYPE_192KB_DB         0x11001100
#define SALESTYPE_128K_DB          0x22002200
#define SALESTYPE_64K_SB           0x33003300

#define SYSCFG_MEMRMP_ADDRESS      0x40010000
#define SYSCFG_MEMMODE_MASK        0x00000003
#define SYSCFG_MEMMODE_SYSMEM      0x00000001
#define SYSCFG_FBMODE_MASK         0x00000008

void SystemInit (void)
{      void (*start_of_bootloader)(void);
        start_of_bootloader = (void(*)(void))*(unsigned int*)0x1FF00004; /* 0x1FF00004 is
the reset vector of the system flash*/

     /* Check what is the device salestype (192KB, 128KB or 64KB) */
  if  (((*(__IO uint32_t *) (ENGI_SALESTYPE_ADDRESS)) &  ENGI_SALESTYPE_MASK) ==
ENGI_SALESTYPE_192KB)
  {
    /* If 192KB salestype */
    SalesType = SALESTYPE_192KB_DB;
  }
  else if (( (*(__IO uint32_t *) (ENGI_SALESTYPE_ADDRESS)) &  ENGI_SALESTYPE_MASK) ==
ENGI_SALESTYPE_128K)
  {
    /* If 128KB salestype */
    SalesType = SALESTYPE_128K_DB;
  }
  else
  {
    /* If 64KB salestype */
    SalesType = SALESTYPE_64K_SB;
  }
/* Enable the SYSCFG APB Clock */
  *(__IO uint32_t *) 0x40021034 |= ((uint32_t)0x00000001);

  /************************ Get salestype  **********************/

  /* Manage case when single bank salestypes are used */
  if (SalesType !=  SALESTYPE_64K_SB)
  {
    /* Get bank2 start address depending on the package */
    Bank2Addr = (SalesType ==  SALESTYPE_192KB_DB)? BANK2_START_ADDRESS_192KB :
BANK2_START_ADDRESS_128KB;

    if (((SYSCFG->CFGR1 & SYSCFG_FBMODE_MASK) ==0))
        { /* we are in bank1*/
/************************ Check Data value  ***********************/
            /* If data value selected to remap Bank2  */
            if (*(__IO uint8_t *)DATA_VALUE_SELECT == DATA_BANK2_SELECTED)
            {
                if (*(unsigned int*)Bank2Addr>0x20000000 && *(unsigned
int*)Bank2Addr<=0x20005004 )
```

```
                    {
                        start_of_bootloader();
                    }
                }
            }
        }
....
}
```

2.1.9 Interrupts masked when using dual-bank boot mechanism

Description

When the dual-bank boot mechanism is enabled, either by setting BFB2 bit in FLASH_OPTR register, the microcontroller starts executing the user code with bit 0 of the core priority mask register (PRIMASK) set. As a result, the interrupts are not serviced.

Workaround

Set bit 0 of PRIMASK register before activating any interrupt. See code below for an example:

```
__ASM ("CPSIE i");
```

2.2 GPIO

2.2.1 PE2 AF2 alternate function (TIM3_ETR) not available

Description

When PE2 is configured as an alternate function 2 (AF2) to receive TIM3_ETR, incoming data may be corrupted due to the fact that the output logic of the same PAD is not properly disabled.

Workaround

Use another GPIO (PD2 AF2) for TIM3_ETR

2.2.2 PB6, PB7, PD0, PD1, PD7 AF1 alternate functions not available when LCD controller is enabled

Description

When the LCD controller is enabled, the following GPIOs cannot be used anymore for the following functions:

- PB6 as I2C1_SCL
- PB7 as I2C1_SDA
- PD0 as SPI2_NSS or I2S2_WS
- PD1 as SPI2_SCK or I2S2_CK
- PD4 as SPI2_MOSI or I2S2_SD
- PD7 as TIM21_CH2

Workaround

None.

2.2.3 PB4 not available on UFQFPN32 package

Description

PB4 GPIO does not exist on UFQFPN32 package. As a result, PB4 cannot be used.

Workaround

None.

2.3 DMA

2.3.1 DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear

Description

Upon a data transfer error in a DMA channel x, both the specific TEIFx and the global GIFx flags are raised and the channel x is normally automatically disabled. However, if in the same clock cycle the software clears the GIFx flag (by setting the CGIFx bit of the DMA_IFCR register), the automatic channel disable fails and the TEIFx flag is not raised.

This issue does not occur with ST's HAL software that does not use and clear the GIFx flag when the channel is active.

Workaround

Do not clear GIFx flags when the channel is active. Instead, use HTIFx, TCIFx, and TEIFx specific event flags and their corresponding clear bits.

2.3.2 Byte and half-word accesses not supported

Description

Some reference manual revisions may wrongly state that the DMA registers are byte- and half-word-accessible. Instead, the DMA registers must always be accessed through aligned 32-bit words. Byte or half-word write accesses cause an erroneous behavior.

ST's low-level driver and HAL software only use aligned 32-bit accesses to the DMA registers.

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

Workaround

No application workaround is required.

2.4 ADC

2.4.1 Overrun flag is not set if EOC reset coincides with new conversion end

Description

If the EOC flag is cleared by an ADC_DR register read operation or by software during the same APB cycle in which the data from a new conversion are written in the ADC_DR register, the overrun event duly occurs (which results in the loss of either current or new data) but the overrun flag (OVR) may stay low.

Workaround

Clear the EOC flag, by performing an ADC_DR read operation or by software within less than one ADC conversion cycle period from the last conversion cycle end, in order to avoid the coincidence with the end of the new conversion cycle.

2.4.2 Writing ADC_CFGR1 register while ADEN bit is set resets RES[1:0] bitfield

Description

Modifying the ADC_CFGR1 register while ADC is enabled (ADEN set in ADC_CR) resets RES[1:0] to 00 whatever the bitfield previous value.

Workaround

Apply the following sequence:
1. Set ADDIS to disable the ADC, and wait until ADEN is cleared.
2. Program the ADC_CFGR1 register according to the application requirements.
3. Set ADEN bit.

2.4.3 Out-of-threshold value is not detected in AWD1 Single mode

Description

AWD1 analog watchdog does not detect that the result of a converted channel has reached the programmed threshold when the ADC operates in Single mode, performs a sequence of conversions, and one of the converted channels other than the first one is monitored by the AWD1 analog watchdog.

Workaround

Apply one of the following measures:
- Use a conversion sequence of one single channel.
- Configure the monitored channel as the first one of the sequence.

2.5 DAC

2.5.1 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.

2.5.2 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.

2.6 COMP

2.6.1 COMP1_CSR and COMP2_CSR lock bit reset by SYSCFGRST bit in RCC_APB2RSTR register

Description

When the SYSCFGRST bit of RCC_APB2RSTR register is set, the COMP1_CSR and COMP2_CSR register contents are reset even if COMP1LOCK and COMP2LOCK bits are set in COMP1_CSR and the COMP2_CSR register, respectively.

Workaround

None.
For security reasons, it is recommended to avoid using SYSCFGRST bit of RCC_APB2RSTR when COMP1LOCK and/or COMP2LOCK bits are set.

2.6.2 Output of comparator 2 cannot be internally connected to input 1 of low-power timer

Description

The COMP2LPTIMIN1 bit (bit 13 of COMP2_CSR register) which internally connects COMP2VALUE to the low-power timer (LPTIM) input 1 has no effect.

Workaround

Connect COMP2_OUT output to an external pin and configure LPTIM_IN1 on an external pin, then connect both pins together externally.

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 Touch sensing channels not available in dynamic voltage range 3

Description

The touch sensing controller, that manages capacitive sensing functionality, is not efficient when range 3 is selected as dynamic voltage scaling.

Workaround

To perform touch sensing detection in your application, select range 2 or 1 as dynamic voltage scaling.

2.8 RNG

2.8.1 RNG clock error does not stop random numbers

Description

Some revisions of the reference manual may contain the following wrong statements to ignore:

- If the RNG clock frequency is too low, the RNG stops generating random numbers.
- The RNG operates only when the CECS flag is set to 0.

This is a documentation issue rather than a device limitation.

Workaround

No application workaround is required or applicable.

2.9 TIM

2.9.1 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).

2.9.2 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.

2.9.3 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:

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

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

As a consequence:

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

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

Workaround

None.

2.9.4 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).

2.10 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_APByRSTRz 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.

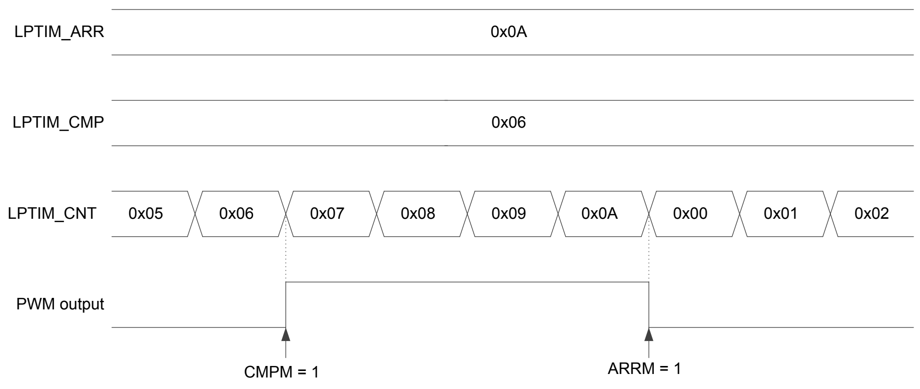
2.10.3 LPTIM events and PWM output are delayed by 1 kernel clock cycle

Description

The compare match event (CMPM), auto reload match event (ARRM), PWM output level and interrupts are updated with a delay of one kernel clock cycle.

Consequently, it is not possible to generate PWM with a duty cycle of 0% or 100%.

The following waveform gives the example of PWM output mode and the effect of the delay:



Workaround

Set the compare value to the desired value minus 1. For instance in order to generate a compare match when LPTM_CNT = 0x08, set the compare value to 0x07.

2.11 RTC and TAMP

2.11.1 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.

2.11.2 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.

2.11.3 RTC interrupt can be masked by another RTC interrupt

Description

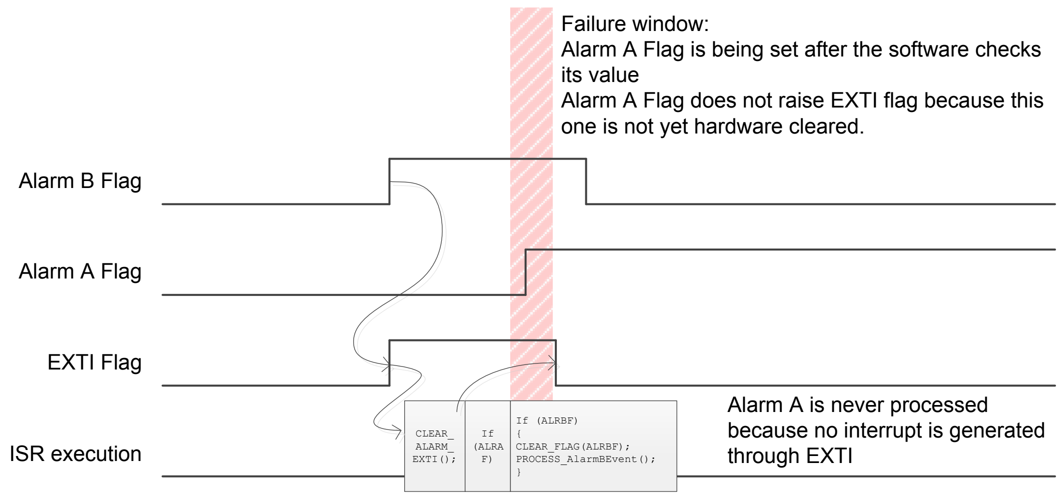
One RTC interrupt request can mask another RTC interrupt request if they share the same EXTI configurable line. For example, interrupt requests from Alarm A and Alarm B or those from tamper and timestamp events are OR-ed to the same EXTI line (refer to the *EXTI line connections* table in the *Extended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


Failure window:
Alarm A Flag is being set after the software checks its value
Alarm A Flag does not raise EXTI flag because this one is not yet hardware cleared.
Alarm B Flag
Alarm A Flag
EXTI Flag
ISR execution
Alarm A is never processed because no interrupt is generated through EXTI

```
CLEAR_
ALARM_
EXTI();
```

```
If
(ALRA
F)
```

```
If (ALRBF)
{
CLEAR_FLAG(ALRBF);
PROCESS_AlarmBEvent();
}
```

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 */
    }
}
```

2.11.4 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.*

2.11.5 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 though the device is configured to stop the RTC in debug.

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

Workaround

None.

2.12 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 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 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

Description

The correct use of the I2C peripheral 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 I^2C-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 I^2C-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 I^2C-bus specification and user manual specify a minimum data setup time ($t_{SU;DAT}$) as:

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

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

Workaround

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

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

2.12.4 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 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.

2.12.5 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:

- I^2C-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 I^2C-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^2C-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.

2.12.6 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 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 I2C_ISR register) occurs.
- After the ADDR flag is set:
  - the device does not write I2C_CR2 before clearing the ADDR flag, or
  - the device writes I2C_CR2 earlier than three I2C kernel clock cycles before clearing the ADDR flag

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

Workaround

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

Normal mode (SBC = 0):

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

Slave byte control mode (SBC = 1):

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

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

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

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

Description

Some reference manual revisions may state that the START bit of the I2C_CR2 register is cleared upon slave address match event.

Instead, the START bit is cleared upon setting, by software, the ADDRCF bit of the I2C_ICR register, which does not guarantee the abort of master transfer request when the device is being addressed as slave. This product limitation and its workaround are the subject of a separate erratum.

Workaround

No application workaround is required for this description inaccuracy issue.

2.12.8 OVR flag not set in underrun condition

Description

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

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

Workaround

None.

2.12.9 Transmission stalled after first byte transfer

Description

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

Workaround

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

2.13 USART

2.13.1 Non-compliant sampling for NACK signal from smartcard

Description

According to ISO/IEC 7816-3 standard, when a character parity error is detected, the receiver must assert a NACK signal, by pulling the transmit line low for one ETU period, at 10.3 to 10.7 ETU after the character START bit falling edge. The transmitter is expected to sample the line for NACK (for low level) from 10.8 to 11.2 ETU after the character START bit falling edge.
Instead, the USART peripheral in Smartcard mode samples the transmit line for NACK from 10.3 to 10.7 ETU after the character START bit falling edge. This is unlikely to cause issues with receivers (smartcards) that respect the ISO/IEC 7816-3 standard. However, it may cause issues with respect to certification.

Workaround

None.

2.13.2 Break request preventing TC flag from being set

Description

After the end of transmission of data (D1), the transmission complete (TC) flag is not set when the following condition is met:
- CTS hardware flow control is enabled
- D1 transmission is in progress
- a break transfer is requested before the end of D1 transfer
- CTS is de-asserted before the end of D1 transfer

As a consequence, an application relying on the TC flag fails to detect the end of data transfer.

Workaround

In the application, only allow break request after the TC flag is set.

2.13.3 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.

2.13.4 Receiver timeout counter wrong start in two-stop-bit configuration

Description

In two-stop-bit configuration, the receiver timeout counter starts counting from the end of the second stop bit of the last character instead of starting from the end of the first stop bit.

Workaround

Subtract one bit duration from the value in the RTO bitfield of the USARTx_RTOR register.

2.13.5 Data corruption due to noisy receive line

Description

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

Workaround

None.

2.14 LPUART

2.14.1 Break request preventing TC flag from being set

Description

After the end of transmission of data (D1), the transmission complete (TC) flag is not set when the following condition is met:

- CTS hardware flow control is enabled
- D1 transmission is in progress
- a break transfer is requested before the end of D1 transfer
- CTS is de-asserted before the end of D1 transfer

As a consequence, an application relying on the TC flag fails to detect the end of data transfer.

Workaround

In the application, only allow break request after the TC flag is set.

2.14.2 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 LPUART 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.

2.14.3 Receiver timeout counter wrong start in two-stop-bit configuration

Description

In two-stop-bit configuration, the receiver timeout counter starts counting from the end of the second stop bit of the last character instead of starting from the end of the first stop bit.

Workaround

Subtract one bit duration from the value in the RTO bitfield of the LPUARTx_RTOR register.

2.14.4 Data corruption due to noisy receive line

Description

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

Workaround

None.

2.15 SPI/I2S

2.15.1 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.

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

Description

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

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

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

Workaround

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

- When NSS hardware management is applied and NSS signal is provided by master, use NSS flag.
- In SPI receiving mode, use the corresponding RXNE event flag.

- In SPI transmit-only mode, use the BSY flag in conjunction with a timeout expiry event. Set the timeout such as to exceed the expected duration of the last data frame and start it upon TXE event that occurs with the second bit of the last data frame. The end of the transaction corresponds to either the BSY flag becoming low or the timeout expiry, whichever happens first.

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

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

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

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

2.15.3 Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback

Description

In receive transaction, in both I^{2}S and SPI master modes, the last bit of the transacted frame is not captured when signal provided by internal feedback loop from the SCK pin exceeds a critical delay. The lastly transacted bit of the stored data then keeps value from the previously received pattern. 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 check sum and/or 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 I^{2}S mode is more sensitive than the SPI mode since the SCK clock is not synchronized with the APB. In this case, margin of the internal feedback delay is lower than one APB clock period.

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

The following table gives the maximum allowable APB frequency versus GPIOx_OSPEEDR output speed bitfield setting for the SCK pin, at 30pF of capacitive load. The operation is safe up to that frequency.

Table 5. **Maximum allowable APB frequency at 30 pF load**

GPIOx_OSPEEDR[1:0] for SCK pin	Max. APB frequency for SPI mode [MHz]	Max. APB frequency for I^{2}S mode [MHz]
11 (very high)	32	32
10 (high)	28 (32 at V_{DD} > 1.8 V)	28 (32 at V_{DD} > 1.8 V)
01 (medium)	12 (14 at V_{DD} > 1.8 V)	8 (10 at V_{DD} > 1.8 V)
00 (low)	1.5 (2 at V_{DD} > 1.8 V)	1 (1.5 at V_{DD} > 1.8 V)

Workaround

The following measures can be adopted, jointly or individually:

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

2.15.4 SPI CRC corruption upon DMA transaction completion by another peripheral

Description

When the following conditions are all met:

- CRC function for the SPI is enabled
- SPI transaction managed by software (as opposed to DMA) is ongoing and CRCNEXT flag set
- another peripheral using the DMA channel on which the SPI is mapped completes a DMA transfer,

the CRCNEXT bit is unexpectedly cleared and the SPI CRC calculation may be corrupted, setting the CRC error flag.

Workaround

Ensure that the DMA channel on which the SPI is mapped is not concurrently in use by another peripheral.

2.15.5 In I^{2}S slave mode, enabling I2S while WS is active causes desynchronization

Description

In I^{2}S slave mode, the WS signal level is used to start the communication. If the I2S peripheral is enabled while the WS line is active (low for I^{2}S protocol, high for LSB- or MSB-justified mode), and if the master is already sending the clock, the I2S peripheral (slave) starts communicating data from the instant of its enable, which causes desynchronization between the master and the slave throughout the whole communication.

Workaround

Enable I2S peripheral while the WS line is at:

- high level, for I^{2}S protocol.
- low level, for LSB- or MSB-justified mode.

2.15.6 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.

2.15.7 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 (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.

2.15.8 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.

2.15.9 I^{2}S slave in PCM short pulse mode sensitive to timing between WS and CK

Description

When the device is configured in I^{2}S slave PCM short frame mode (I2SMOD = 1, I2SCFG[1:0] = 00 or 01, I2SSTD[1:0] = 11, PCMSYNC = 0), with the asynchronous start disabled (ASTREN = 0), then if the master does not respect the WS hold time versus clock, the data transmitted and received by the device get desynchronized (shifted by one bit). This results in discarding two frames instead of one, irrespectively of the clock polarity.

Workaround

Use legacy mode (ASTERN = 1), while ensuring a perfect synchronization between the master and the slave.

2.15.10 Limited SPI frequency when peripheral is configured in Master reception or in Slave transmission mode and V_{DD} is below 2.7 V

Description

When the SPI is configured in Master reception or in Slave transmission mode, the maximum SPI frequency should be 16 MHz in Range1 (V_{DD} ranging from 1,71 to 3,6 V). However a timing issue limits the maximum transaction reachable frequency when V_{DD} is below 2.7 V.

Workaround

None.

2.16 USB

2.16.1 Possible packet memory overrun/underrun at low APB frequency

Description

Some data sheet and/or reference manual revisions may omit the information that 10 MHz minimum APB clock frequency is required to avoid USB data overrun/underrun issues.

Operating the USB peripheral with lower APB clock frequency may lead to:

- Overrun for *out* transactions - the USB peripheral fails to store the received data into the PBM before the next byte is received on the USB (PBM overrun). The USB cell detects an internal error condition, discards the last received byte, stops writing into the PBM, sends no acknowledge (forcing the host to retry the transaction), and informs the application by setting the PMAOVR flag/interrupt.
- Underrun for *in* transactions - the USB peripheral fails to read from the PBM the next byte to transmit before the transmission of the previous one is completed on the USB. The USB cell detects an internal error condition, stops reading from PBM, generates a bit stuffing error on the USB (forcing the host to retry the transaction), and informs the application by setting the PMAOVR flag/interrupt.

This is a documentation issue rather than a device limitation.

Workaround

No application workaround is required if the minimum APB clock frequency of 10 MHz is respected.

2.16.2 ESOF interrupt timing desynchronized after resume signaling

Description

Upon signaling resume, the device is expected to allow full 3 ms of time to the host or hub for sending the initial SOF (start of frame) packet, without triggering SUSP interrupt. However, the device only allows two full milliseconds and unduly triggers SUSP interrupt if it receives the initial packet within the third millisecond.

Workaround

When the device initiates resume (remote wakeup), mask the SUSP interrupt by setting the SUSPM bit for 3 ms, then unmask it by clearing SUSPM.

2.16.3 Incorrect CRC16 in the memory buffer

Description

Memory buffer locations are written starting from the address contained in the ADDRn_RX for a number of bytes corresponding to the received data packet length, CRC16 inclusive (that is, data payload length plus two bytes), or up to the last allocated memory location defined by BL_SIZE and NUM_BLOCK, whichever comes first. In the former case, the CRC16 checksum is written wrongly, with its least significant byte going to both memory buffer byte locations expected to receive the least and the most significant bytes of the checksum.
Although the checksum written in the memory buffer is wrong, the underlying CRC checking mechanism in the USB peripheral is fully functional.

Workaround

Ignore the CRC16 data in the memory buffer.

2.16.4 USB BCD functionality limited below –20 °C

Description

Primary and secondary detection can return an incorrectly detected port type.
This limitation may be observed on a small number of devices when the temperature is below –20 °C.

Workaround

None.

Revision history

Table 6. **Document revision history**

Date	Version	Changes
02-Jul-2015	1	Initial release.
11-Sep-2015	2	Updated Section 2.1.2: BFB2 bit does not select boot from Bank 2 system memory. Added Section 2.1.8: I2C and USART cannot wake up the device from Stop mode and Section 2.1.9: LDM, STM, PUSH and POP not allowed in IOPORT bus.
14-Sep-2015	3	Updated Section 2.1.8: I2C and USART cannot wake up the device from Stop mode.
19-Oct-2015	4	Changed confidentiality level to public.
18-Dec-2015	5	The following limitations are fixed in device revision Z: • BFB2 bit does not select boot from Bank 2 system memory • I2C and USART cannot wake up the device from Stop mode • BOOT_MODE bits do not reflect the selected boot mode (this limitation has been added in current errata sheet revision). Added Section 2.2.1: COMP1_CSR and COMP2_CSR lock bit reset by SYSCFGRST bit in RCC_APB2RSTR register . Updated Section 2.5.3: Last data bit or CRC calculation may be corrupted for the data received in master mode depending on the feedback communication clock timing with respect to the APB clock (SPI or I2S).
09-Dec-2021	6	Merged all STM32L07xxx and STM32L08xxx device errata sheets into one single document. Distinction made between device and documentation errata. Added GPIO, DMA, ADC, DAC, TIM, LPTIM errata. Updated System, RTC and TAMP, I2C, SPI/I2S and USB errata. Distinction made between USART and LPUART and errata updated.
24-Mar-2022	7	Added all STM32L073xx part numbers as well as STM32L081CZ. Corrected product references in Table 2. Device variants. Updated Writing ADC_CFGR1 register while ADEN bit is set resets RES[1:0] bitfield (2.4.2), Device may remain stuck in LPTIM interrupt when clearing event flag (2.10.2) and Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback (2.15.3) errata.

Contents