Errata sheet

STM32F070x6/xB device errata

Applicability

This document applies to the part numbers of STM32F070x6/xB 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 RM0360.

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
STM32F070x6	STM32F070C6, STM32F070F6
STM32F070xB	STM32F070CB, STM32F070RB

Table 2. **Device variants**

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32F070x6	A or 1	0x1000
STM32F070xB	Y, 1, or 2	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 STM32F070x6/xB 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	
			STM32F070x6 Rev. A, 1	STM32F070xB Rev. Y, 1, 2
System	2.2.1	Wakeup sequence from Standby mode when using more than one wakeup source	A	A
	2.2.2	RDP Level 1 issue	P	P
GPIO	2.3.1	GPIOx locking mechanism not working properly for GPIOx_OTYPER register	P	P
DMA	2.4.1	DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	A	A
ADC	2.5.1	ADCAL bit is not cleared when successive calibrations are performed and system clock frequency is considerably higher than the ADC clock frequency	A	A
	2.5.2	Overrun flag is not set if EOC reset coincides with new conversion end	P	P
	2.5.3	ADEN bit cannot be set immediately after the ADC calibration	A	A
TIM	2.6.1	PWM re-enabled in automatic output enable mode despite of system break	P	P
	2.6.3	Consecutive compare event missed in specific conditions	N	N
	2.6.4	Output compare clear not working with external counter reset	P	P
IWDG	2.7.1	RVU flag not reset in Stop	A	A
	2.7.2	PVU flag not reset in Stop	A	A
	2.7.3	WVU flag not reset in Stop	A	A
	2.7.4	RVU flag not cleared at low APB clock frequency	A	A
	2.7.5	PVU flag not cleared at low APB clock frequency	A	A
	2.7.6	WVU flag not cleared at low APB clock frequency	A	A

Function	Section	Limitation	Status	
			STM32F070x6 Rev. A, 1	STM32F070xB Rev. Y, 1, 2
RTC and TAMP	2.8.1	Spurious tamper detection when disabling the tamper channel	N	N
	2.8.2	RTC calendar registers are not locked properly	A	A
	2.8.3	RTC interrupt can be masked by another RTC interrupt	A	A
	2.8.4	Calendar initialization may fail in case of consecutive INIT mode entry	A	A
	2.8.5	Alarm flag may be repeatedly set when the core is stopped in debug	N	N
	2.8.6	A tamper event preceding the tamper detect enable not detected	A	A
I2C	2.9.1	10-bit slave mode: wrong direction bit value upon Read header receipt	-	A
	2.9.2	10-bit combined with 7-bit slave mode: ADDCODE may indicate wrong slave address detection	-	N
	2.9.3	10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave	A	A
	2.9.5	Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period	P	P
	2.9.6	Spurious bus error detection in master mode	A	A
	2.9.7	Last-received byte loss in reload mode	P	P
	2.9.8	Spurious master transfer upon own slave address match	P	P
	2.9.9	OVR flag not set in underrun condition	N	N
	2.9.10	Transmission stalled after first byte transfer	A	A
USART	2.10.1	USART4 transmission does not work on PC11	-	N
	2.10.2	Last byte written in TDR might not be transmitted if TE is cleared just after writing in TDR	A	A
	2.10.3	Break request preventing TC flag from being set	A	A
	2.10.4	RTS is active while RE = 0 or UE = 0	A	A
	2.10.5	Receiver timeout counter wrong start in two-stop-bit configuration	A	A
	2.10.6	Anticipated end-of-transmission signaling in SPI slave mode	A	A
	2.10.7	Data corruption due to noisy receive line	N	N
SPI	2.11.1	BSY bit may stay high when SPI is disabled	A	A
	2.11.2	BSY bit may stay high at the end of data transfer in slave mode	A	A
	2.11.3	SPI CRC corruption upon DMA transaction completion by another peripheral	P	P
USB	2.12.2	ESOF interrupt timing desynchronized after resume signaling	A	A
	2.12.3	Incorrect CRC16 in the memory buffer	N	N
	2.12.4	The USB BCD functionality limited below -20°C	N	N
	2.12.5	DCD function not compliant	P	P

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

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

Function	Section	Documentation erratum
DMA	2.4.2	Byte and half-word accesses not supported
TIM	2.6.2	TRGO and TRGO2 trigger output failure
I2C	2.9.4	Wrong behavior in Stop mode
SPI	2.11.4	CRC error in SPI slave mode if internal NSS changes before CRC transfer
USB	2.12.1	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 Core

Reference manual and errata notice for the Arm® Cortex®-M0 core revision r0p0 is available from http://infocenter.arm.com.

2.2 System

2.2.1 Wakeup sequence from Standby mode when using more than one wakeup source

Description

The various wakeup sources are logically OR-ed in front of the rising-edge detector that generates the wakeup flag (WUF). The WUF needs to be cleared prior to Standby mode entry, otherwise the MCU wakes up immediately.

If one of the configured wakeup sources is kept high during the clearing of the WUF (by setting the CWUF bit), it may mask further wakeup events on the input of the edge detector. As a consequence, the MCU might not be able to wake up from Standby mode.

Workaround

Apply the following sequence before entering Standby mode:

1. Disable all used wakeup sources
2. Clear all related wakeup flags
3. Re-enable all used wakeup sources
4. Enter Standby mode

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

2.2.2 RDP Level 1 issue

Description

When the RDP Level 1 protection is set, there exists a logic issue that compromises protection of the Flash memory against debugger access. When the debugger is connected to the device, the first transaction with the Flash memory after a power on reset/power up is granted because of a race condition existing between this debugger access and the protection mechanism of the Flash memory. As a result, the debugger may access one data in the Flash memory after power up.

Workaround

For customers concerned by the confidentiality of their firmware, it is recommended to use the RDP Level 2 protection.

2.3 GPIO

2.3.1 GPIOx locking mechanism not working properly for GPIOx_OTYPER register

Description

Locking GPIOx_OTYPER[i] with i = 15 to 8 unduly depends on GPIOx_LCKR[i-8] instead on GPIOx_LCKR[i]. GPIOx_LCKR[i-8] locks both GPIOx_OTYPER[i] and GPIOx_OTYPER[i-8]. It is not possible to lock GPIOx_OTYPER[i] with i = 15...8 without also locking GPIOx_OTYPER[i-8].

Workaround

The only way to lock GPIOx_OTYPER[i] with i=15 to 8 is to also lock GPIOx_OTYPER[i-8].

2.4 DMA

2.4.1 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.

2.4.2 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.

2.5 ADC

2.5.1 ADCAL bit is not cleared when successive calibrations are performed and system clock frequency is considerably higher than the ADC clock frequency

Description

The ADC calibration is launched by setting ADCAL bit of ADC_CR register. It can only be initiated when the ADC is disabled (ADEN cleared in ADC_CR register). ADCAL bit stays at 1 during the whole calibration sequence and is cleared by hardware as soon the calibration completes.

However, when at least two calibrations are performed in a row and the system clock frequency is considerably higher than the ADC clock, the ADCAL bit is set again after being cleared by hardware when the first calibration phase ends. The ADCAL bit remains set, waiting for the calibration to complete and hence for a hardware clear that never occurs since the ADC clock is stopped.

Workaround

Avoid performing successive calibrations.

2.5.2 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.

2.5.3 ADEN bit cannot be set immediately after the ADC calibration

Description

At the end of the ADC calibration, an internal reset of ADEN bit occurs four ADC clock cycles after the ADCAL bit is cleared by hardware. As a consequence, if the ADEN bit is set within those four ADC clock cycles, it is reset shortly after by the calibration logic and the ADC remains disabled.

Workaround

Apply one of the following measures:
- When the ADC calibration is complete (ADCAL = 0), keep setting the ADEN bit until the ADRDY flag goes high.
- After the ADCAL is cleared, wait for a minimum of four ADC clock cycles before enabling the ADC (ADEN = 1).
- Always perform the ADC calibration with ADC clock frequency = APB frequency / 2.

2.6 TIM

2.6.1 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).

2.6.2 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.

2.6.3 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.

2.6.4 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).

2.7 IWDG

2.7.1 RVU flag not reset in Stop

Description

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

Workaround

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

2.7.2 PVU flag not reset in Stop

Description

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

Workaround

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

2.7.3 WVU flag not reset in Stop

Description

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

Workaround

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

2.7.4 RVU flag not cleared at low APB clock frequency

Description

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

Workaround

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

2.7.5 PVU flag not cleared at low APB clock frequency

Description

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

Workaround

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

2.7.6 WVU flag not cleared at low APB clock frequency

Description

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

Workaround

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

2.8 RTC and TAMP

2.8.1 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.

2.8.2 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.

2.8.3 RTC interrupt can be masked by another RTC interrupt

Description

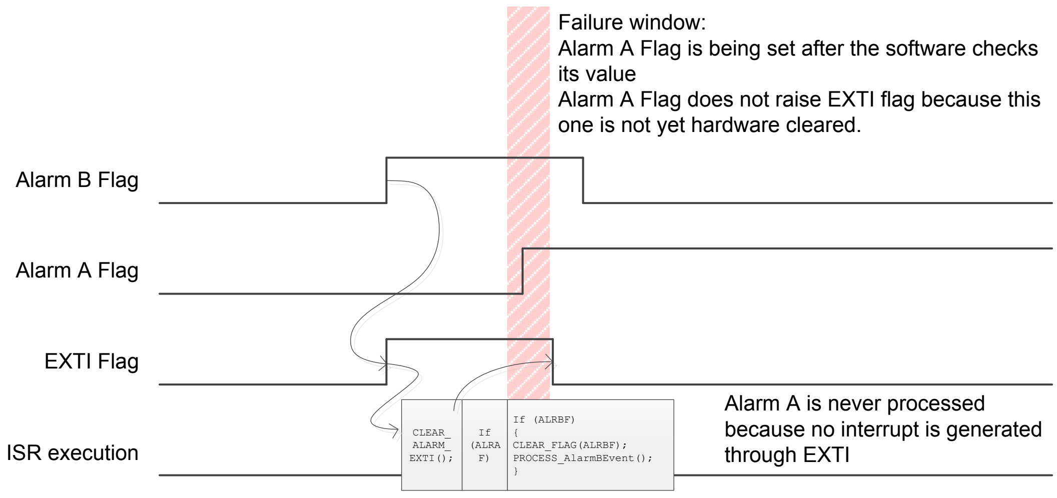
One RTC interrupt request can mask another RTC interrupt request if they share the same EXTI configurable line. For example, interrupt requests from Alarm A and Alarm B or those from tamper and timestamp events are OR-ed to the same EXTI line (refer to the *EXTI line connections* table in the *Extended 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
CLEAR_ ALARM_ EXTI();
If (ALRAF)
If (ALRBF) { CLEAR_FLAG(ALRBF); PROCESS_AlarmBEvent(); }
Alarm A is never processed because no interrupt is generated through EXTI

Workaround

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

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

2.8.4 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.

2.8.5 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.

2.8.6 A tamper event preceding the tamper detect enable not detected

Description

When the tamper detect is enabled, set in edge detection mode (TAMPFLT[1:0]=00), and

- set to active rising edge (TAMPxTRG=0): if the tamper input is already high (tamper event already occurred) at the moment of enabling the tamper detection, the tamper event may not be detected. The probability of detection increases with the APB frequency.
- set to active falling edge (TAMPxTRG=1): if the tamper input is already low (tamper event already occurred) at the moment of enabling the tamper detection, the tamper event is not detected.

Workaround

The I/O state should be checked by software in the GPIO registers after enabling the tamper detection, in order to ensure that no active edge occurred before enabling the tamper event detection.

2.9 I2C

2.9.1 10-bit slave mode: wrong direction bit value upon Read header receipt

Description

Under specific conditions, the transfer direction bit DIR (bit 16 of status register I2C_ISR) remains low upon receipt of 10-bit addressing Read header, while normally it should be set high. Nevertheless, I2C operates correctly in slave transmission mode, and data can be sent using the TXIS flag.

The failure described occurs when the following conditions are all met:

- I2C is configured in 10-bit addressing mode (OA1MODE is set in the I2C_OAR1 register).
- High LSBs of the slave address are equal to the 10-bit addressing Read header value (that is, OA1[7:3] = 11110, OA1[2] = OA1[9], OA1[1] = OA1[8], and OA1[0] = 1, in the I2C_OAR1 register).
- I2C receives 10-bit addressing Read header (0X 1111 0XX1) after repeated START condition, to enter slave transmission mode.

Workaround

Avoid using the following 10-bit slave addresses:

- OA1[9:0] = 0011110001
- OA1[9:0] = 0111110011
- OA1[9:0] = 1011110101
- OA1[9:0] = 1111110111

If the use of one of these slave addresses cannot be avoided, do not use the DIR bit in the firmware.

2.9.2 10-bit combined with 7-bit slave mode: ADDCODE may indicate wrong slave address detection

Description

Under specific conditions, the ADDCODE (address match code) bitfield in the I2C_ISR register indicates a wrong slave address.

The failure occurs when the following conditions are all met:

- A 10-bit slave address OA1 is enabled (OA1EN = 1 and OA1MODE = 1).
- A 7-bit slave address OA2 is enabled (OA2EN = 1) and it matches the non-masked bits of OA1[7:1], that is, one of the following configurations is set:
  - OA2EN = 1 and OA2MSK = 0 and OA1[7:1] = OA2[7:1]
  - OA2EN = 1 and OA2MSK = 1 and OA1[7:2] = OA2[7:2]
  - OA2EN = 1 and OA2MSK = 2 and OA1[7:3] = OA2[7:3]
  - OA2EN = 1 and OA2MSK = 3 and OA1[7:4] = OA2[7:4]
  - OA2EN = 1 and OA2MSK = 4 and OA1[7:5] = OA2[7:5]
  - OA2EN = 1 and OA2MSK = 5 and OA1[7:6] = OA2[7:6]
  - OA2EN = 1 and OA2MSK = 6 and OA1[7] = OA2[7]
  - OA2EN = 1 and OA2MSK = 7
  - GCEN = 1 and OA1[7:1] = 0000000
  - ALERTEN = 1 and OA1[7:1] = 0001100
  - SMBDEN = 1 and OA1[7:1] = 1100001
  - SMBHEN = 1 and OA1[7:1] = 0001000
- The MCU is addressed by a bus master with its 10-bit slave address OA1.

Upon the address receipt, the ADDCODE value is OA1[7:1] equal to the 7-bit slave address, instead of 0b11110 & OA1[9:8].

Workaround

None. If several slave addresses are enabled, mixing 10-bit and 7-bit addresses, the OA1 [7:1] part of the 10-bit slave address must be different than the 7-bit slave address.

2.9.3 10-bit master mode: new transfer cannot be launched if first part of the address is not acknowledged by the slave

Description

An I^{2}C-bus master generates STOP condition upon non-acknowledge of I^{2}C 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^{2}C-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^{2}C-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.

2.9.4 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.

2.9.5 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.

2.9.6 Spurious bus error detection in master mode

Description

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

Workaround

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

2.9.7 Last-received byte loss in reload mode

Description

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

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

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

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.

2.9.8 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.

2.9.9 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.

2.9.10 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.

2.10 USART

2.10.1 USART4 transmission does not work on PC11

Description

USART4_RX does not work as output on PC11.

As a consequence, single-wire half-duplex mode is not supported with pin PC11.

Workaround

Use USART4_RX mapped on PA0 instead on PC11.

2.10.2 Last byte written in TDR might not be transmitted if TE is cleared just after writing in TDR

Description

If the USART clock source is slow (for example LSE) and TE bit is cleared immediately after the last write to TDR, the last byte may not be transmitted.

Workaround

Apply one of the following measures:
- Wait until TXE flag is set before clearing TE bit.
- Wait until TC flag is set before clearing TE bit.

2.10.3 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
- D1 transmission is in progress
- D1 transmission is in progress

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.

2.10.4 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.

2.10.5 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.

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

Description

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

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

Workaround

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

2.10.7 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.

2.11 SPI

2.11.1 BSY bit may stay high when SPI is disabled

Description

The BSY flag may remain high upon disabling the SPI while operating in:
- master transmit mode and the TXE flag is low (data register full).
- master receive-only mode (simplex receive or half-duplex bidirectional receive phase) and an SCK strobing edge has not occurred since the transition of the RXNE flag from low to high.
- slave mode and NSS signal is removed during the communication.

Workaround

When the SPI operates in:
- master transmit mode, disable the SPI when TXE = 1 and BSY = 0.
- master receive-only mode, ignore the BSY flag.
- slave mode, do not remove the NSS signal during the communication.

2.11.2 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.

2.11.3 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.

Alternatively, remap SPI2 to a DMA channel not used by another peripheral.

2.11.4 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.

2.12 USB

2.12.1 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.

2.12.2 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.

2.12.3 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.

2.12.4 The 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.

2.12.5 DCD function not compliant

Description

The DCD (data contact detect) function on the device is not compliant with the *USB Battery Charging 1.2 Compliance Plan rev 1.0* specification.

Workaround

Do not use the DCD function. Instead, upon attaching a USB device, wait for at least *TDCD_TIMEOUT* amount of time before starting primary detection. This is in line with the *Battery Charging Specification rev1.2* recommendation for portable devices that do not support the DCD function.

Revision history

Table 5. **Document revision history**

Date	Version	Changes
15-Jan-2015	1	Initial release.
12-Oct-2016	2	Added errata in IWDG section: • *RVU, PVU and WVU flags are not reset in STOP mode* • *RVU, PVU and WVU flags are not reset with low-frequency APB* Modified: • document structure • cover page and Table 3 organization
09-May-2018	3	Added: • REV_ID bitfield information on the cover page • table *Summary of documentation errata* • information on workaround qualifiers in section *Summary of device errata* • the following errata: • *DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear* • *RTC interrupt can be masked by another RTC interrupt* • *Byte and half-word accesses not supported* • *Last-received byte loss in reload mode* • *Spurious master transfer upon own slave address match* Modified: • minor modifications in titles and/or text of existing limitation descriptors in I2C, SPI and USART sections • workaround description in erratum *Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period* • erratum *Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period* qualified as documentation erratum and re-written • document ID in the footer of all pages to ES0291 • renaming of introductory section on the cover page Removed redundant limitation *Wrong CRC transmitted in master mode with delay on SCK feedback* in SPI section, kept in previous versions for historical reasons.
03-Dec-2020	4	Added product revision 1 for STM32F070x6 and product revision 2 for STM32F070xB. Added errata: • System: RDP Level 1 issue • ADC: ADCAL bit is not cleared when successive calibrations are performed and system clock frequency is considerably higher than the ADC clock frequency • TIM: PWM re-enabled in automatic output enable mode despite of system break • TRGO and TRGO2 trigger output failure • Consecutive compare event missed in specific conditions • Output compare clear not working with external counter reset • IWDG: RVU flag not reset in Stop • PVU flag not reset in Stop • WVU flag not reset in Stop • RVU flag not cleared at low APB clock frequency • PVU flag not cleared at low APB clock frequency

Date	Version	Changes
		• WVU flag not cleared at low APB clock frequency • RTC: Calendar initialization may fail in case of consecutive INIT mode entry • Alarm flag may be repeatedly set when the core is stopped in debug • I2C: Spurious bus error detection in master mode • Transmission stalled after first byte transfer • USART: USART4 transmission does not work on PC11 • Anticipated end-of-transmission signaling in SPI slave mode • Data corruption due to noisy receive line Modified errata: • DMA: DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear • ADC: ADCAL bit is not cleared when successive calibrations are performed and system clock frequency is considerably higher than the ADC clock frequency workaround re-qualified to P • I2C: Spurious master transfer upon own slave address match workaround re-qualified to P • SPI: CRC error in SPI slave mode if internal NSS changes before CRC transfer rewritten as documentation erratum • USB: DCD function not compliant workaround re-qualified to P Removed two errata from Section 2.7 IWDG (superseded with the six errata added in the section).

Contents