STM32F20x and STM32F21x Errata sheet

STM32F205/207xx and STM32F215/217xx device limitations

Silicon identification

This errata sheet applies to the revision ‘Y’, ‘X’, ‘1’, ‘V’, ‘2’, ‘3’, ‘4’ and ‘5’ of STMicroelectronics STM32F205xx/STM32F207xx and STM32F215xx/STM32F217xx microcontrollers. In this document they will be referred to as STM32F20x and STM32F21x, respectively, unless otherwise specified.

The STM32F205xx/STM32F207xx and STM32F215xx/STM32F217xx feature revision r2p0-v2 of the Arm® 32-bit Cortex®-M3 core, for which an errata notice is also available (see *Section 1* for details).

The full list of part numbers is shown in *Table 2*. The products are identifiable as shown in *Table 1*:

- by the revision code marked below the order code on the device package
- by the last three digits of the Internal order code printed on the box label

Table 1. Device identification(1)

Order code	Revision code marked on device(2)
STM32F205xx, STM32F207xx	‘Y’, ‘X’, ‘1’, ‘V’, ‘2’, ‘3’, ‘4’ and ‘5’
STM32F215xx, STM32F217xx	‘Y’, ‘X’, ‘1’, ‘V’, ‘2’, ‘3’, ‘4’ and ‘5’

1. The REV_ID bits in the DBGMCU_IDCODE register show the revision code of the device (see the reference manual for details on how to find the revision code).
2. Refer to the datasheets for details on how to identify the revision code according to the packages.

Table 2. Device summary

Reference	Part number
STM32F205xx	STM32F205RB, STM32F205RC, STM32F205RE, STM32F205RF, STM32F205RG, STM32F205VB, STM32F205VC, STM32F205VE, STM32F205VF, STM32F205VG, STM32F205ZC, STM32F205ZE, STM32F205ZF, STM32F205ZG
STM32F207xx	STM32F207IC, STM32F207IE, STM32F207IF, STM32F207IG, STM32F207VC, STM32F207VE, STM32F207VF, STM32F207VG, STM32F207ZC, STM32F207ZE, STM32F207ZF, STM32F207ZG
STM32F215xx	STM32F215RG, STM32F215VG, STM32F215ZG, STM32F215RE, STM32F215VE, STM32F215ZE
STM32F217xx	STM32F217VG, STM32F217IG, STM32F217ZG, STM32F217VE, STM32F217IE, STM32F217ZE

Contents

1 Arm® 32-bit Cortex®-M3 limitations 7

1.1 Cortex-M3 limitations description for the STM32F20x and STM32F21x devices 7

1.1.1 Cortex-M3 LDRD with base in list may result in incorrect base register when interrupted or faulted 7

1.1.2 Cortex-M3 event register is not set by interrupts and debug 8

1.1.3 Cortex-M3 interrupted loads to stack pointer can cause erroneous behavior 8

1.1.4 Cortex-M3 SVC and BusFault/MemManage may occur out of order . . . 9

2 STM32F20x and STM32F21x silicon limitations 10

2.1 System limitations 13

2.1.1 ART Accelerator prefetch queue instruction is not supported 13

2.1.2 Debugging Stop mode and system tick timer 13

2.1.3 Debugging Stop mode with WFE entry 14

2.1.4 Wakeup sequence from Standby mode when using more than one wakeup source 14

2.1.5 Full JTAG configuration without NJTRST pin cannot be used 15

2.1.6 DBGMCU_CR register cannot be read by user software 15

2.1.7 Configuration of PH10 and PI10 as external interrupts is erroneous . . . 15

2.1.8 DMA2 data corruption when managing AHB and APB peripherals in a concurrent way 15

2.1.9 Slowing down APB clock during a DMA transfer 16

2.1.10 MPU attribute to RTC and IWDG registers could be managed incorrectly 16

2.1.11 Delay after an RCC peripheral clock enabling 16

2.1.12 Battery charge monitoring lower than 2.4 V 17

2.1.13 Internal noise impacting the ADC accuracy 17

2.1.14 RDP level 2 and sector write protection configuration 17

2.1.15 Possible delay in backup domain protection disabling/enabling after programming the DBP bit 18

2.2 IWDG peripheral limitation 18

2.2.1 RVU and PVU flags are not reset in STOP mode 18

2.3 RTC peripheral limitation 19

2.3.1 Setting GPIO properties of PC13 used as RTC_ALARM open-drain output 19

2.4 I2C peripheral limitations 19
2.4.1 SMBus standard not fully supported 19
2.4.2 Start cannot be generated after a misplaced Stop 19
2.4.3 Mismatch on the “Setup time for a repeated Start condition” timing parameter 20
2.4.4 Data valid time ($t_{VD;DAT}$) violated without the OVR flag being set 20
2.4.5 Both SDA and SCL maximum rise time (t_r) violated when VDD_I2C bus is higher than ((VDD + 0.3) / 0.7) V 21
2.5 I2S peripheral limitations 21
2.5.1 Wrong WS signal generation in 16-bit extended to 32-bit PCM long synchronization mode 21
2.5.2 In I2S slave mode, WS level must be set by the external master when enabling the I2S 21
2.5.3 I2S slave mode desynchronization with the master during communication 22
2.6 SPI peripheral limitations 22
2.6.1 BSY bit may stay high when SPI is disabled 22
2.6.2 BSY bit may stay high at the end of data transfer in slave mode 22
2.6.3 Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback 23
2.6.4 Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters 24
2.6.5 CRC error in SPI slave mode if internal NSS changes before CRC transfer 24
2.6.6 Anticipated communication upon SPI transit from slave receiver to master 25
2.7 USART peripheral limitations 25
2.7.1 Idle frame is not detected if receiver clock speed is deviated 25
2.7.2 In full duplex mode, the Parity Error (PE) flag can be cleared by writing to the data register 25
2.7.3 Parity Error (PE) flag is not set when receiving in Mute mode using address mark detection 26
2.7.4 Break frame is transmitted regardless of nCTS input line status 26
2.7.5 nRTS signal abnormally driven low after a protocol violation 26
2.7.6 Start bit detected too soon when sampling for NACK signal from the smartcard 27
2.7.7 Break request can prevent the Transmission Complete flag (TC) from being set 27
2.7.8 Guard time is not respected when data are sent on TXE events 27
2.7.9 nRTS is active while RE or UE = 0 28

2.8 bxCAN peripheral limitations 28
2.8.1 bxCAN time triggered communication mode not supported 28
2.9 OTG_FS peripheral limitations 28
2.9.1 Data in RxFIFO overwritten when all channels are disabled simultaneously 28
2.9.2 Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_FS registers 29
2.9.3 Host packet transmission may hang when connecting through a hub to a low-speed device 29
2.9.4 OTG host blocks the receive channel when receiving IN packets and no TxFIFO is configured 29
2.9.5 Host channel-halted interrupt not generated when the channel is disabled 29
2.9.6 Error in software-read OTG_FS_DCFG register values 30
2.9.7 Minimum AHB frequency to guarantee correct operation of USB OTG FS peripheral 30
2.10 OTG_HS peripheral limitations 30
2.10.1 Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_HS registers 30
2.10.2 Host packet transmission may hang when connecting through a hub to a low-speed device 31
2.11 Ethernet peripheral limitations 31
2.11.1 Incorrect layer 3 (L3) checksum is inserted in transmitted IPv6 packets without TCP, UDP or ICMP payloads 31
2.11.2 The Ethernet MAC processes invalid extension headers in the received IPv6 frames 31
2.11.3 MAC stuck in the Idle state on receiving the TxFIFO flush command exactly one clock cycle after a transmission completes 32
2.11.4 Transmit frame data corruption 32
2.11.5 Successive write operations to the same register might not be fully taken into account 32
2.11.6 MCO PLL clock pins not compatible with Ethernet IEEE802.3 long term jitter specifications 35
2.12 FSMC peripheral limitations 36
2.12.1 Dummy read cycles inserted when reading synchronous memories . . . 36
2.12.2 FSMC synchronous mode and NWAIT signal disabled 36
2.12.3 FSMC NOR Flash/PSRAM controller asynchronous access on bank 2 to 4 when bank 1 is in synchronous mode (CBURSTRW bit is set) 37
2.13 SDIO peripheral limitations 37
2.13.1 SDIO HW flow control 37

2.13.2 Wrong CCRCFAIL status after a response without CRC is received . . . 37
2.13.3 SDIO clock divider BYPASS mode may not work properly 38
2.13.4 Data corruption in SDIO clock dephasing (NEGEDGE) mode 38
2.13.5 CE-ATA multiple write command and card busy signal management . . 38
2.13.6 No underrun detected with wrong data transmission 38
2.14 ADC limitations 39
2.14.1 ADC sequencer modification during conversion 39
2.15 DAC limitations 39
2.15.1 DMA underrun flag management 39
2.15.2 DMA request not automatically cleared by DMAEN=0 39

3 Revision history 41

List of tables

Table 1. Device identification 1
Table 2. Device summary 1
Table 3. Cortex-M3 core limitations and impact on microcontroller behavior 7
Table 4. Summary of silicon limitations 10
Table 5. Impacted registers and bits 33
Table 6. Document revision history 41

1 Arm® 32-bit Cortex®-M3 limitations

An errata notice of the STM32F20x and STM32F21x core is available on Arm®[a] website http://infocenter.arm.com.

All the described limitations are minor and related to the revision r2p0-v2 of the Cortex-M3 core. *Table 3* summarizes these limitations and their implications on the behavior of high-density STM32F20x and STM32F21x devices.

Table 3. Cortex-M3 core limitations and impact on microcontroller behavior

Arm ID	Arm category	Arm summary of errata	Impact on STM32F20x and STM32F21x devices
752419	Cat 2	Interrupted loads to SP can cause erroneous behavior	Minor
740455	Cat 2	SVC and BusFault/MemManage may occur out of order	Minor
602117	Cat 2	LDRD with base in list may result in incorrect base register when interrupted or faulted	Minor
563915	Cat 2	Event register is not set by interrupts and debug	Minor

1.1 Cortex-M3 limitations description for the STM32F20x and STM32F21x devices

Only the limitations described below have an impact, even though minor, on the implementation of STM32F20x and STM32F21x devices.

All the other limitations described in the Arm errata notice (and summarized in *Table 3* above) have no impact and are not related to the implementation of STM32F20x and STM32F21x devices (Cortex-M3 r2p0-v2).

1.1.1 Cortex-M3 LDRD with base in list may result in incorrect base register when interrupted or faulted

Description

The Cortex-M3 Core has a limitation when executing an LDRD instruction from the system-bus area, with the base register in a list of the form LDRD Ra, Rb, [Ra, #imm]. The execution may not complete after loading the first destination register due to an interrupt before the second loading completes or due to the second loading getting a bus fault.

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

Workarounds

1. This limitation does not impact the STM32F20x and STM32F21x code execution when executing from the embedded Flash memory, which is the standard use of the microcontroller.
2. Use the latest compiler releases. As of today, they no longer generate this particular sequence. Moreover, a scanning tool is provided to detect this sequence on previous releases (refer to your preferred compiler provider).

1.1.2 Cortex-M3 event register is not set by interrupts and debug

Description

When interrupts related to a WFE occur before the WFE is executed, the event register used for WFE wakeup events is not set and the event is missed. Therefore, when the WFE is executed, the core does not wake up from WFE if no other event or interrupt occurs.

Workaround

Use STM32F20x and STM32F21x external events instead of interrupts to wake up the core from WFE by configuring an external or internal EXTI line in event mode.

1.1.3 Cortex-M3 interrupted loads to stack pointer can cause erroneous behavior

Description

An interrupt occurring during the data-phase of a single word load to the stack pointer (SP/R13) can caused an erroneous behavior of the device. In addition, returning from the interrupt results in the load instruction being executed an additional time.

For all the instructions performing an update of the base register, the base register is erroneously updated on each execution, resulting in the stack pointer being loaded from an incorrect memory location.

The instructions affected by this limitation are the following:

- LDR SP, [Rn],#imm
- LDR SP, [Rn,#imm]!
- LDR SP, [Rn,#imm]
- LDR SP, [Rn]
- LDR SP, [Rn,Rm]

Workaround

As of today, no compiler generates these particular instructions. This limitation can only occur with hand-written assembly code.

Both issues can be solved by replacing the direct load to the stack pointer by an intermediate load to a general-purpose register followed by a move to the stack pointer.

Example:

Replace LDR SP, [R0] by

LDR R2,[R0]

MOV SP,R2

1.1.4 Cortex-M3 SVC and BusFault/MemManage may occur out of order

Description

If an SVC exception is generated by executing the SVC instruction while the next instruction fetch is faulted, then the MemManage or BusFault handler may be entered even though the faulted instruction following the SVC instruction should not have been executed.

Workaround

A workaround is required only if the SVC handler does not return to the return address that has been stacked for the SVC exception, and the instruction access after the SVC will fault. In this case, insert padding (e.g. NOP instructions) between the SVC instruction and the faulting code.

2 STM32F20x and STM32F21x silicon limitations

Table 4 gives quick references to all documented limitations.

Legend for *Table 4*: A = workaround available; N = no workaround available; P = partial workaround available, '-' and grayed = fixed, 'D' documentation erratum.

Table 4. Summary of silicon limitations

Links to silicon limitations		Revision 'Y', 'X', '1', 'V', '2', '3', '4' and '5'
Section 2.1: System limitations	*Section 2.1.1: ART Accelerator prefetch queue instruction is not supported*	N
	Section 2.1.2: Debugging Stop mode and system tick timer	A
	Section 2.1.3: Debugging Stop mode with WFE entry	A
	Section 2.1.4: Wakeup sequence from Standby mode when using more than one wakeup source	A
	Section 2.1.5: Full JTAG configuration without NJTRST pin cannot be used	A
	Section 2.1.6: DBGMCU_CR register cannot be read by user software	N
	Section 2.1.7: Configuration of PH10 and PI10 as external interrupts is erroneous	N
	Section 2.1.8: DMA2 data corruption when managing AHB and APB peripherals in a concurrent way	A
	Section 2.1.9: Slowing down APB clock during a DMA transfer	A
	Section 2.1.10: MPU attribute to RTC and IWDG registers could be managed incorrectly	A
	Section 2.1.11: Delay after an RCC peripheral clock enabling	A
	Section 2.1.12: Battery charge monitoring lower than 2.4 V	P
	Section 2.1.13: Internal noise impacting the ADC accuracy	A
	Section 2.1.14: RDP level 2 and sector write protection configuration	A
	Section 2.1.15: Possible delay in backup domain protection disabling/enabling after programming the DBP bit	A
Section 2.2: IWDG peripheral limitation	*Section 2.2.1: RVU and PVU flags are not reset in STOP mode*	A
Section 2.3: RTC peripheral limitation	*Section 2.3.1: Setting GPIO properties of PC13 used as RTC_ALARM open-drain output*	D

Table 4. Summary of silicon limitations (continued)

Links to silicon limitations		Revision 'Y', 'X', '1', 'V', '2', '3', '4' and '5'
Section 2.4: I2C peripheral limitations	Section 2.4.1: SMBus standard not fully supported	A
	Section 2.4.2: Start cannot be generated after a misplaced Stop	A
	Section 2.4.3: Mismatch on the "Setup time for a repeated Start condition" timing parameter	A
	Section 2.4.4: Data valid time ($t_{VD;DAT}$) violated without the OVR flag being set	A
	Section 2.4.5: Both SDA and SCL maximum rise time (t_r) violated when VDD_I2C bus is higher than ((VDD + 0.3) / 0.7) V	A
Section 2.5: I2S peripheral limitations	Section 2.5.1: Wrong WS signal generation in 16-bit extended to 32-bit PCM long synchronization mode	A
	Section 2.5.2: In I2S slave mode, WS level must be set by the external master when enabling the I2S	A
	Section 2.5.3: I2S slave mode desynchronization with the master during communication	A
Section 2.6: SPI peripheral limitations	Section 2.6.1: BSY bit may stay high when SPI is disabled	A
	Section 2.6.2: BSY bit may stay high at the end of data transfer in slave mode	A
	Section 2.6.3: Corrupted last bit of data and/or CRC, received in master mode with delayed SCK feedback	A
	Section 2.6.4: Wrong CRC in full-duplex mode handled by DMA with imbalanced setting of data counters	A
	Section 2.6.5: CRC error in SPI slave mode if internal NSS changes before CRC transfer	A
	Section 2.6.6: Anticipated communication upon SPI transit from slave receiver to master	A
Section 2.7: USART peripheral limitations	Section 2.7.2: In full duplex mode, the Parity Error (PE) flag can be cleared by writing to the data register	A
	Section 2.7.3: Parity Error (PE) flag is not set when receiving in Mute mode using address mark detection	N
	Section 2.7.4: Break frame is transmitted regardless of nCTS input line status	N
	Section 2.7.5: nRTS signal abnormally driven low after a protocol violation	A
	Section 2.7.6: Start bit detected too soon when sampling for NACK signal from the smartcard	N
	Section 2.7.7: Break request can prevent the Transmission Complete flag (TC) from being set	A
	Section 2.7.8: Guard time is not respected when data are sent on TXE events	A
	Section 2.7.9: nRTS is active while RE or UE = 0	A

Table 4. Summary of silicon limitations (continued)

Links to silicon limitations		Revision 'Y', 'X', '1', 'V', '2', '3', '4' and '5'
Section 2.8: bxCAN peripheral limitations	*Section 2.8.1: bxCAN time triggered communication mode not supported*	N
Section 2.9: OTG_FS peripheral limitations	*Section 2.9.1: Data in RxFIFO overwritten when all channels are disabled simultaneously*	A
	Section 2.9.2: Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_FS registers	A
	Section 2.9.3: Host packet transmission may hang when connecting through a hub to a low-speed device	N
	Section 2.9.4: OTG host blocks the receive channel when receiving IN packets and no TxFIFO is configured	A
	Section 2.9.5: Host channel-halted interrupt not generated when the channel is disabled	A
	Section 2.9.6: Error in software-read OTG_FS_DCFG register values	A
	Section 2.9.7: Minimum AHB frequency to guarantee correct operation of USB OTG FS peripheral	N
Section 2.10: OTG_HS peripheral limitations	*Section 2.10.1: Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_HS registers*	A
	Section 2.10.2: Host packet transmission may hang when connecting through a hub to a low-speed device	N
Section 2.11: Ethernet peripheral limitations	*Section 2.11.1: Incorrect layer 3 (L3) checksum is inserted in transmitted IPv6 packets without TCP, UDP or ICMP payloads*	A
	Section 2.11.2: The Ethernet MAC processes invalid extension headers in the received IPv6 frames	N
	Section 2.11.3: MAC stuck in the Idle state on receiving the TxFIFO flush command exactly one clock cycle after a transmission completes	A
	Section 2.11.4: Transmit frame data corruption	A
	Section 2.11.5: Successive write operations to the same register might not be fully taken into account	A
	Section 2.11.6: MCO PLL clock pins not compatible with Ethernet IEEE802.3 long term jitter specifications	A
Section 2.12: FSMC peripheral limitations	*Section 2.12.1: Dummy read cycles inserted when reading synchronous memories*	N
	Section 2.12.2: FSMC synchronous mode and NWAIT signal disabled	A
	Section 2.12.3: FSMC NOR Flash/PSRAM controller asynchronous access on bank 2 to 4 when bank 1 is in synchronous mode (CBURSTRW bit is set)	A

Table 4. Summary of silicon limitations (continued)

Links to silicon limitations		Revision 'Y', 'X', '1', 'V', '2', '3', '4' and '5'
Section 2.13: SDIO peripheral limitations	*Section 2.13.1: SDIO HW flow control*	N
	Section 2.13.2: Wrong CCRCFAIL status after a response without CRC is received	N
	Section 2.13.3: SDIO clock divider BYPASS mode may not work properly	A
	Section 2.13.4: Data corruption in SDIO clock dephasing (NEGEDGE) mode	N
	Section 2.13.5: CE-ATA multiple write command and card busy signal management	A
	Section 2.13.6: No underrun detected with wrong data transmission	A
Section 2.14: ADC limitations	*Section 2.14.1: ADC sequencer modification during conversion*	A
Section 2.15: DAC limitations	*Section 2.15.1: DMA underrun flag management*	A
	Section 2.15.2: DMA request not automatically cleared by DMAEN=0	A

Note: Revisions "2", "V", "1" and "X" differ from revision "Y" as they improve the LSE (low speed external oscillator) power consumption, without fixing limitations. Refer to STM32F20xx and STM32F21xx datasheets.

2.1 System limitations

2.1.1 ART Accelerator prefetch queue instruction is not supported

Description

The ART Accelerator prefetch queue instruction is not supported when V_{DD} is lower than 2.1 V.
This limitation does not prevent the ART Accelerator from using the cache enable/disable capability and the selection of the number of wait states according to the system frequency.

Workaround

None. Refer to application note AN3430 (available on *www.st.com*) for details on how to adjust performance and power consumption.

2.1.2 Debugging Stop mode and system tick timer

Description

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

Workaround

To debug the Stop mode, disable the system tick timer interrupt.

2.1.3 Debugging Stop mode with WFE entry

Description

When the Stop debug mode is enabled (DBG_STOP bit set in the DBGMCU_CR register), this allows software debugging during Stop mode.

However, if the application software uses the WFE instruction to enter Stop mode, after wakeup some instructions could be missed if the WFE is followed by sequential instructions. This affects only Stop debug mode with WFE entry.

Workaround

To debug Stop mode with WFE entry, the WFE instruction must be inside a dedicated function with 1 instruction (NOP) between the execution of the WFE and the Bx LR.

Example:

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

2.1.4 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 generating 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

To avoid this problem, the following sequence must be applied before entering Standby mode:

- Disable all used wakeup sources,
- Clear all related wakeup flags,
- Re-enable all used wakeup sources,
- 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.

2.1.5 Full JTAG configuration without NJTRST pin cannot be used

Description

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

Workaround

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

2.1.6 DBGMCU_CR register cannot be read by user software

Description

The DBGMCU_CR debug register is accessible only in Debug mode (not accessible by the user software). When this register is read in user mode, the returned value is 0x00.

Workaround

None.

2.1.7 Configuration of PH10 and PI10 as external interrupts is erroneous

Description

PH10 or PI10 are selected as the source for EXTI10 external interrupt by setting bits EXTI10[3:0] of SYSCFG_EXTICR3 register to 0x0111 or 0x1000, respectively. However, this erroneous operation enables PH2 and PI2 as external interrupt inputs.

As a result, it is not possible to use PH10/PI10 as interrupt sources if PH2/PI2 are not selected as interrupt sources as well. This means that bits EXTI10[3:0] of SYSCFG_EXTICR3 register and bits EXTI2[3:0] of SYSCFG_EXTICR1 must be programmed to the same value:

- 0x0111 to select PH10/PH2
- 0x1000 to select PI10/PI2

Workaround

None.

2.1.8 DMA2 data corruption when managing AHB and APB peripherals in a concurrent way

Description

When the DMA2 is managing AHB Peripherals (only peripherals embedding FIFOs) and also APB transfers in a concurrent way, this generates a data corruption (multiple DMA access).

When this condition occurs:

- The data transferred by the DMA to the AHB peripherals could be corrupted in case of a FIFO target.
- For memories, it will result in multiple access (not visible by the Software) and the data is not corrupted.
- For the DCMI, a multiple unacknowledged request could be generated, which implies an unknown behavior of the DMA.

AHB peripherals embedding FIFO are DCMI, CRYPTO, and HASH. On sales types without CRYPTO, only the DCMI is impacted. External FIFO controlled by the FSMC is also impacted.

Workaround

Avoid concurrent AHB (DCMI, CRYPTO, HASH, FSMC with external FIFO) and APB transfer management using the DMA2.

2.1.9 Slowing down APB clock during a DMA transfer

Description

When the CPU modifies the APB clock (slows down the clock: changes AHB/APB prescaler from 1 to 2, 1 to 4, 1 to 8 or 1 to 16) while the DMA is performing a write access to the same APB peripherals, the current DMA transfer is blocked. Only system reset can recover.

Workaround

Before slowing down the APB clock, wait until the end of the DMA transfer on this APB.

2.1.10 MPU attribute to RTC and IWDG registers could be managed incorrectly

Description

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

Workaround

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

2.1.11 Delay after an RCC peripheral clock enabling

Description

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

This delay depends on the peripheral mapping:

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

Workarounds

1. Use the DSB instruction to stall the Cortex-M CPU pipeline until the instruction is completed.
2. Insert “n” NOPs between the RCC enable bit write and the peripheral register writes (n = 2 for AHB peripherals, n = 1 + AHB/APB prescaler in case of APB peripherals).

2.1.12 Battery charge monitoring lower than 2.4 V

Description

If (V_{DD} = V_{DDA}) is lower than or equal to 2.4 V, the V_{BAT} conversion correctness is not guaranteed over the full temperature and voltage ranges. When V_{BAT} is set, the voltage divider bridge is enabled and V_{BAT}/2 is connected to the ADC input. In order to monitor the battery charge correctly, the input of the ADC must not be higher than (V_{DDA} - 0.6 V).

Thus, $V_{BAT}/2 < V_{DD} – 0.6$ V implies that $V_{DD} > 2.4$ V.

Workaround

None. V_{DD} (= V_{DDA}) must be greater than 2.4 V.

2.1.13 Internal noise impacting the ADC accuracy

Description

An internal noise generated on V_{DD} supplies and propagated internally may impact the ADC accuracy.

This noise is always active whatever the power mode of the MCU (RUN or Sleep).

Workarounds

Two steps could be followed to adapt the accuracy level to the application requirements:

1. Configure the Flash ART as Prefetch OFF and (Data + Instruction) cache ON.
2. Use averaging and filtering algorithms on ADC output codes.

For more workaround details of this limitation refer to AN4073 (available on *www.st.com*).

2.1.14 RDP level 2 and sector write protection configuration

Description

When the MCU is protected with RDP level2, the configuration of the sector write protection remains changeable by the user code.

Workarounds

Protect sensitive sectors and FLASH_OPTCR register using the Cortex-M MPU (memory protection unit) taking special care of ISR management.

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

Description

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

The higher APB1 prescaler value, the higher the delay.

Workaround

Apply one of the following countermeasures:

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

2.2 IWDG peripheral limitation

2.2.1 RVU and PVU flags are not reset in STOP mode

Description

The RVU and PVU flags of the IWDG_SR register are set by hardware after a write access to the IWDG_RLR and the IWDG_PR registers, respectively. If the Stop mode is entered immediately after the write access, the RVU and PVU flags are not reset by hardware.

Before performing a second write operation to the IWDG_RLR or the IWDG_PR register, the application software must wait for the RVU or PVU flag to be reset. However, since the RVU/PVU bit is not reset after exiting the Stop mode, the software goes into an infinite loop and the independent watchdog (IWDG) generates a reset after the programmed timeout period.

Workaround

Wait until the RVU or PVU flag of the IWDG_SR register is reset before entering the Stop mode.

2.3 RTC peripheral limitation

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

Description

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

Note: *Enabling the internal pull-up function through the PC13 GPIO settings allows sparing an external pull-up resistor.*

This is a documentation issue rather than a product limitation.

Workaround

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

2.4 I2C peripheral limitations

2.4.1 SMBus standard not fully supported

Description

The I^2C peripheral is not fully compliant with the SMBus v2.0 standard since It does not support the capability to NACK an invalid byte/command.

Workarounds

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

1. Using the SMBAL pin if supported by the host
2. the alert response address (ARA) protocol
3. the Host notify protocol

2.4.2 Start cannot be generated after a misplaced Stop

Description

If a master generates a misplaced Stop on the bus (bus error), the peripheral cannot generate a Start anymore.

Workaround

In the I^2C standard, it is allowed to send a Stop only at the end of the full byte (8 bits + acknowledge), so this scenario is not allowed. Other derived protocols like CBUS allow it, but they are not supported by the I^2C peripheral.

A software workaround consists in asserting the software reset using the SWRST bit in the I2C_CR1 control register.

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

Description

In case of a repeated Start, the “Setup time for a repeated Start condition” (named Tsu;sta in the I²C specification) can be slightly violated when the I²C operates in Master Standard mode at a frequency between 88 kHz and 100 kHz.

The issue can occur only in the following configuration:

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

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

Workaround

Reduce the frequency down to 88 kHz or use the I²C Fast-mode, if supported by the slave.

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

Description

The data valid time ($t_{VD;DAT}$, $t_{VD;ACK}$) described by the I²C standard can be violated (as well as the maximum data hold time of the current data ($t_{HD;DAT}$)) under the conditions described below. This violation cannot be detected because the OVR flag is not set (no transmit buffer underrun is detected).

This issue can occur only under the following conditions:

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

Workaround

If the master device allows it, use the clock stretching mechanism by programming the bit NOSTRETCH=0 in the I2C_CR1 register.

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

2.4.5 Both SDA and SCL maximum rise time (t_r) violated when V_{DD_I2C} bus is higher than ((V_{DD} + 0.3) / 0.7) V

Description

When an external legacy I²C bus voltage (V_{DD_I2C}) is set to 5 V while the MCU is powered from V_{DD}, the internal 5-Volt tolerant circuitry is activated as soon the input voltage (V_{IN}) reaches the V_{DD} + diode threshold level. An additional internal large capacitance then prevents the external pull-up resistor (R_P) from rising the SDA and SCL signals within the maximum timing (t_r), which is 300 ns in fast mode and 1000 ns in Standard mode.

The rise time (t_r) is measured from V_{IL} and V_{IH} with levels set at 0.3VDD_I2C and 0.7 * V_{DD_I2C}.

Workaround

The external V_{DD_I2C} bus voltage must be limited to a maximum value of ((V_{DD} + 0.3) / 0.7) V.

As a result, when the MCU is powered from V_{DD}=3.3 V, V_{DD_I2C} must not exceed 5.14 V to be compliant with I²C specifications.

2.5 I2S peripheral limitations

2.5.1 Wrong WS signal generation in 16-bit extended to 32-bit PCM long synchronization mode

Description

When I2S is master with PCM long synchronization is selected as16-bit data frame extended to 32-bit, the WS signal is generated every 16 rather than every 32 bits.

Workaround

Only the 16-bit mode with no data extension can be used when the I2S is master and when the selected mode has to be PCM long synchronization mode.

2.5.2 In I2S slave mode, WS level must be set by the external master when enabling the I2S

Description

In slave mode, the WS signal level is used only to start the communication. If the I2S (in slave mode) is enabled while the master is already sending the clock and the WS signal level is low (for I2S protocol) or is high (for the LSB or MSB-justified mode), the slave starts communicating data immediately. In this case, the master and slave will be desynchronized throughout the whole communication.

Workaround

The I2S peripheral must be enabled when the external master sets the WS line at:

- High level when the I2S protocol is selected.
- Low level when the LSB or MSB-justified mode is selected.

2.5.3 I2S slave mode desynchronization with the master during communication

Description

In I2S slave mode, if glitches on SCK or WS signals are generated at an unexpected time, a desynchronization of the master and the slave occurs. No error is reported to allow audio system to re-synchronize.

Workaround

The following workarounds can be applied in order to detect and react after a desynchronization by disabling and enabling I2S peripheral in order to resynchronize with the master.

1. Monitoring the I2S WS signal through an external Interrupt to check the I2S WS signal status.
2. Monitoring the I2S clock signal through an input capture interrupt to check the I2S clock signal status.
3. Monitoring the I2S clock signal through an input capture interrupt and the I2S WS signal via an external interrupt to check the I2S clock and I2S WS signals status.

2.6 SPI peripheral limitations

2.6.1 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.

2.6.2 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 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.

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

Description

In receive transaction, in both I2S 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 received 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 I2S 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 clockperiod.

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

Workaround

The following measures can be adopted, jointly or individually:

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

Note: All the GPIOx_OSPEEDR[1:0] bitfield settings for the SCK pin are safe up to maximum allowed APB bus frequencies except the lowest speed setting (00), when maximum achievable ABP frequency is 28 MHz for SPI and 16 MHz for I2S (with a 30 pF load).

2.6.4 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.

2.6.5 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.

2.6.6 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.

2.7 USART peripheral limitations

2.7.1 Idle frame is not detected if receiver clock speed is deviated

Description

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

Workaround

None.

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

Description

In full duplex mode, when the Parity Error flag is set by the receiver at the end of a reception, it may be cleared while transmitting by reading the USART_SR register to check the TXE or TC flags and writing data to the data register.

Consequently, the software receiver can read the PE flag as '0' even if a parity error occurred.

Workaround

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

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

Description

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

Workaround

None.

2.7.4 Break frame is transmitted regardless of nCTS input line status

Description

When CTS hardware flow control is enabled (CTSE = 1) and the Send Break bit (SBK) is set, the transmitter sends a break frame at the end of the current transmission regardless of nCTS input line status.

Consequently, if an external receiver device is not ready to accept a frame, the transmitted break frame is lost.

Workaround

None.

2.7.5 nRTS signal abnormally driven low after a protocol violation

Description

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

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

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

Workaround

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

Two workarounds can be used:

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

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

Description

According to ISO/IEC 7816-3 standard, when a character parity error is detected, the receiver transmits a NACK error signal 10.5 ± 0.2 ETUs after the character START bit falling edge. In this case, the transmitter is able to detect correctly the NACK signal until 11 ± 0.2 ETUs after the character START bit falling edge.

In Smartcard mode, the USART peripheral monitors the NACK signal during the receiver time frame (10.5 ± 0.2 ETUs), while it must wait for it during the transmitter one (11 ± 0.2 ETUs). In real cases, this would not be a problem as the card itself needs to respect a 10.7 ETU period when sending the NACK signal. However this may be an issue to undertake a certification.

Workaround

None

2.7.7 Break request can prevent the Transmission Complete flag (TC) from being set

Description

After the end of transmission of a data (D1), the Transmission Complete (TC) flag will not be set if the following conditions are met:

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

Workaround

If the application needs to detect the end of a data transfer, the break request must be issued after checking that the TC flag is set.

2.7.8 Guard time is not respected when data are sent on TXE events

Description

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

Workaround

Write the data after TC is set because in Smartcard mode, the TC flag is set at the end of the guard time duration.

2.7.9 nRTS is active while RE or UE = 0

Description

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

Workaround

Configure the I/O used for nRTS as an alternate function after setting the UE and RE bits.

2.8 bxCAN peripheral limitations

2.8.1 bxCAN time triggered communication mode not supported

Description

The time triggered communication mode described in the RM0033 (available on *www.st.com*) is not supported. As a consequence, time stamp values are not available. TTCM bit must be kept cleared in the CAN_MCR register (time triggered communication mode disabled).

Workaround

None.

2.9 OTG_FS peripheral limitations

2.9.1 Data in RxFIFO overwritten when all channels are disabled simultaneously

Description

If the available RxFIFO is just large enough to host 1 packet + its data status, and is currently occupied by the last received data + its status and, at the same time, the application requests that more IN channels be disabled, the OTG_FS peripheral does not first check for available space before inserting the disabled status of the IN channels. It just inserts them by overwriting the existing data payload.

Workaround

Use one of the following recommendations:

1. Configure the RxFIFO to host a *minimum* of 2 × MPSIZ + 2 × data status entries.
2. The application has to check the RXFLVL bit (RxFIFO non-empty) in the OTG_FS_GINTSTS register before disabling each IN channel. If this bit is not set, then the application can disable an IN channel at a time. Each time the application disables an IN channel, however, it first has to check that the RXFLVL bit = 0 condition is true.

2.9.2 Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_FS registers

Description

When the USB on-the-go full-speed peripheral is in Device mode, interrupting transmit FIFO write sequence with read or write accesses to OTG_FS endpoint-specific registers (those ending in *0* or *x*) leads to corruption of the next data written to the transmit FIFO.

Workaround

Ensure that the transmit FIFO write sequence is not interrupted with accesses to the OTG_FS registers.

2.9.3 Host packet transmission may hang when connecting through a hub to a low-speed device

Description

When the USB on-the-go full-speed peripheral connects to a low-speed device via a hub, the transmitter internal state machine may hang. This leads, after a timeout expiry, to a port disconnect interrupt.

Workaround

None. However, increasing the capacitance on the data lines may reduce the occurrence.

2.9.4 OTG host blocks the receive channel when receiving IN packets and no TxFIFO is configured

Description

When receiving data, the OTG_FS core erroneously checks for available TxFIFO space when it should only check for RxFIFO space. If the OTG_FS core cannot see any space allocated for data transmission, it blocks the reception channel and no data is received.

Workaround

Set at least one TxFIFO equal to the maximum packet size. In this way, the host application, which intends to supports only IN traffic, also has to allocate some space for the TxFIFO.

Since a USB host is expected to support any kind of connected endpoint, it is good practice to always configure enough TxFIFO space for OUT endpoints.

2.9.5 Host channel-halted interrupt not generated when the channel is disabled

Description

When the application enables, then immediately disables the host channel before the OTG_FS host has had time to begin the transfer sequence, the OTG_FS core, as a host, does not generate a channel-halted interrupt. The OTG_FS core continues to operate normally.

Workaround

Do not disable the host channel immediately after enabling it.

2.9.6 Error in software-read OTG_FS_DCFG register values

Description

When the application writes to the DAD and PFIVL bitfields in the OTG_FS_DCFG register, and then reads the newly written bitfield values, the read values may not be correct.

The values written by the application, however, are correctly retained by the core, and the normal operation of the device is not affected.

Workaround

Do not read from the OTG_FS_DCFG register’s DAD and PFIVL bitfields just after programming them.

2.9.7 Minimum AHB frequency to guarantee correct operation of USB OTG FS peripheral

Description

To guarantee correct operation of the USB OTG_FS peripheral, the AHB frequency must be configured to be not less than 14.2 MHz.

Workaround

None.

2.10 OTG_HS peripheral limitations

2.10.1 Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_HS registers

Description

When the USB on-the-go high-speed peripheral is in Device mode, interrupting transmit FIFO write sequence with read or write accesses to OTG_HS endpoint-specific registers (those ending in 0 or x) leads to corruption of the next data written to the transmit FIFO.

Workaround

Ensure that the transmit FIFO write sequence is not interrupted with accesses to the OTG_HS registers. Note that enabling DMA mode guarantees this.

2.10.2 Host packet transmission may hang when connecting through a hub to a low-speed device

Description

When the USB on-the-go high-speed peripheral connects to a low-speed device via a hub, the transmitter internal state machine may hang. This leads, after a timeout expiry, to a port disconnect interrupt.

Workaround

None. However, increasing the capacitance on the data lines may reduce the occurrence

2.11 Ethernet peripheral limitations

2.11.1 Incorrect layer 3 (L3) checksum is inserted in transmitted IPv6 packets without TCP, UDP or ICMP payloads

Description

The application provides the per-frame control to instruct the MAC to insert the L3 checksums for TCP, UDP and ICMP packets. When automatic checksum insertion is enabled and the input packet is an IPv6 packet without the TCP, UDP or ICMP payload, then the MAC may incorrectly insert a checksum into the packet. For IPv6 packets without a TCP, UDP or ICMP payload, the MAC core considers the next header (NH) field as the extension header and continues to parse the extension header. Sometimes, the payload data in such packets matches the NH field for TCP, UDP or ICMP and, as a result, the MAC core inserts a checksum.

Workaround

When the IPv6 packets have a TCP, UDP or ICMP payload, enable checksum insertion for transmit frames, or bypass checksum insertion by using the CIC (checksum insertion control) bits in TDES0 (bits 23:22).

2.11.2 The Ethernet MAC processes invalid extension headers in the received IPv6 frames

Description

In IPv6 frames, there can be zero or some extension headers preceding the actual IP payload. The Ethernet MAC processes the following extension headers defined in the IPv6 protocol: Hop-by-Hop Options header, Routing header and Destination Options header. All extension headers, except the Hop-by-Hop extension header, can be present multiple times and in any order before the actual IP payload. The Hop-by-Hop extension header, if present, has to come immediately after the IPv6's main header.

The Ethernet MAC processes all (valid or invalid) extension headers including the Hop-by-Hop extension headers that are present after the first extension header. For this reason, the GMAC core will accept IPv6 frames with invalid Hop-by-Hop extension headers. As a consequence, it will accept any IP payload as valid IPv6 frames with TCP, UDP or ICMP payload, and then incorrectly update the Receive status of the corresponding frame.

Workaround

None.

2.11.3 MAC stuck in the Idle state on receiving the TxFIFO flush command exactly one clock cycle after a transmission completes

Description

When the software issues a TxFIFO flush command, the transfer of frame data stops (even in the middle of a frame transfer). The TxFIFO read controller goes into the Idle state (TFRS=00 in ETH_MACDBGR) and then resumes its normal operation.

However, if the TxFIFO read controller receives the TxFIFO flush command exactly one clock cycle after receiving the status from the MAC, the controller remains stuck in the Idle state and stops transmitting frames from the TxFIFO. The system can recover from this state only with a reset (e.g. a soft reset).

Workaround

Do not use the TxFIFO flush feature.

If TXFIFO flush is really needed, wait until the TxFIFO is empty prior to using the TxFIFO flush command.

2.11.4 Transmit frame data corruption

Frame data corrupted when the TxFIFO is repeatedly transitioning from non-empty to empty and then back to non-empty.

Description

Frame data may get corrupted when the TxFIFO is repeatedly transitioning from non-empty to empty for a very short period, and then from empty to non-empty, without causing an underflow.

This transitioning from non-empty to empty and back to non-empty happens when the rate at which the data is being written to the TxFIFO is almost equal to or a little less than the rate at which the data is being read.

This corruption cannot be detected by the receiver when the CRC is inserted by the MAC, as the corrupted data is used for the CRC computation.

Workaround

Use the Store-and-Forward mode: TSF = 1 (bit 21 in ETH_DMAOMR). In this mode, the data is transmitted only when the whole packet is available in the TxFIFO.

2.11.5 Successive write operations to the same register might not be fully taken into account

Description

A write operation to a register might not be fully taken into account if a previous write to the same register is performed within a time period of four TX_CLK/RX_CLK clock cycles. When this error occurs, reading the register returns the most recently written value, but the Ethernet MAC continues to operate as if the latest write operation never occurred.

See Table 5 for the registers and bits impacted by this limitation.

Table 5. Impacted registers and bits

Register name	Bit number	Bit name
DMA registers		
ETH_DMABMR	7	EDFE
ETH_DMAOMR	26	DTCEFD
	25	RSF
	20	FTF
	7	FEF
	6	FUGF
	4:3	RTC
GMAC registers		
ETH_MACCR	25	CSTF
	23	WD
	22	JD
	19:17	IFG
	16	CSD
	14	FES
	13	ROD
	12	LM
	11	DM
	10	IPCO
	9	RD
	7	APCS
	6:5	BL
	4	DC
	3	TE
	2	RE
ETH_MACFFR	-	MAC frame filter register
ETH_MACHTHR	31:0	Hash table high register
ETH_MACHTLR	31:0	Hash table low register

Table 5. Impacted registers and bits (continued)

Register name	Bit number	Bit name
ETH_MACFCR	31:16	PT
	7	ZQPD
	5:4	PLT
	3	UPFD
	2	RFCE
	1	TFCE
	0	FCB/BPA
ETH_MACVLANTR	16	VLANTC
	15:0	VLANTI
ETH_MACRWUFFR	-	All remote wakeup registers
ETH_MACPMTCSR	31	WFFRPR
	9	GU
	2	WFE
	1	MPE
	0	PD
ETH_MACA0HR	-	MAC address 0 high register
ETH_MACA0LR	-	MAC address 0 low register
ETH_MACA1HR	-	MAC address 1 high register
ETH_MACA1LR	-	MAC address 1 low register
ETH_MACA2HR	-	MAC address 2 high register
ETH_MACA2LR	-	MAC address 2 low register
ETH_MACA3HR	-	MAC address 3 high register
ETH_MACA3LR	-	MAC address 3 low register

Table 5. Impacted registers and bits (continued)

Register name	Bit number	Bit name
IEEE 1588 time stamp registers		
ETH_PTPTSCR	18	TSPFFMAE
	17:16	TSCNT
	15	TSSMRME
	14	TSSEME
	13	TSSIPV4FE
	12	TSSIPV6FE
	11	TSSPTPOEFE
	10	TSPTPPSV2E
	9	TSSSR
	8	TSSARFE
	5	TSARU
	3	TSSTU
	2	TSSTI
	1	TSFCU
	0	TSE

Workaround

Two workarounds could be applicable:

- Ensure a delay of four TX_CLK/RX_CLK clock cycles between the successive write operations to the same register.
- Make several successive write operations without delay, then read the register when all the operations are complete, and finaly reprogram it after a delay of four TX_CLK/RX_CLK clock cycles.

2.11.6 MCO PLL clock pins not compatible with Ethernet IEEE802.3 long term jitter specifications

Description

When the clock source output by the microcontroller on the MCO pin is issued from the PLL, the MCO pin cannot be used to deliver a 50 MHz RMII clock input or a 25 MHz MII clock input to the ethernet PHY compliant with the long term jitter maximum value for 1.4 ns specified in the IEEE802.3 standard.

This limitation applies both to MCO1 and MCO2 pins and PLLs.

Workaround

- In MII mode
  Use a 25 MHz external crystal to generate the HSE clock and output the clock signal on the MCO pin to clock the PHY
- In RMII mode
  Either use an external 50 MHz oscillator to clock the PHY or select a PHY with an internal PLL that is able to generate the 50 MHz RMII clock.

2.12 FSMC peripheral limitations

2.12.1 Dummy read cycles inserted when reading synchronous memories

Description

When performing a burst read access to a synchronous memory, two dummy read accesses are performed at the end of the burst cycle whatever the type of AHB burst access. However, the extra data values which are read are not used by the FSMC and there is no functional failure.

Example

If AHB data size = 32bit and MEMSIZE= 16bit, two extra 16-bit reads will be performed.

Workaround

None.

2.12.2 FSMC synchronous mode and NWAIT signal disabled

Description

When the FSMC synchronous mode operates with the NWAIT signal disabled, if the polarity (WAITPOL in the FSMC_BCRx register) of the NWAIT signal is identical to that of the NWAIT input signal level, the system hangs and no fault is generated.

Workaround

PD6 (NWAIT signal) must not be connected to AF12 and the NWAIT polarity must be configured to active high (set WAITPOL bit to 1 in FSMC_BCRx register).

2.12.3 FSMC NOR Flash/PSRAM controller asynchronous access on bank 2 to 4 when bank 1 is in synchronous mode (CBURSTRW bit is set)

Description

If bank 1of the NOR/PSRAM controller is enabled in synchronous write mode (CBURSTRW bit set), while all other NOR/PSRAM banks (2 to 4) are enabled in asynchronous mode, two issues occur:

- The byte lane NBL[1:0] are not active (kept at ‘1’) for the first write access to the asynchronous memory.
- The system hangs without any fault generation when a write access is performed to an asynchronous memory with the extended feature enabled.

These two issues occur only when the NOR/PSRAM bank 1 is configured in synchronous write mode (CBURSTRW bit set).

Workaround

If multiple banks are enabled with mixed asynchronous and synchronous write modes, use any NOR/PSRAM bank for synchronous write access, except bank 1.

2.13 SDIO peripheral limitations

2.13.1 SDIO HW flow control

Description

When enabling the HW flow control by setting bit 14 of the SDIO_CLKCR register to ‘1’, glitches can occur on the SDIOCLK output clock resulting in wrong data to be written into the SD/MMC card or into the SDIO device. As a consequence, a CRC error will be reported to the SD/SDIO MMC host interface (DCRCFAIL bit set to ‘1’ in SDIO_STA register).

Workaround

None.

Do not use the HW flow control. Overrun errors (Rx mode) and FIFO underrun (Tx mode) must be managed by the application software.

2.13.2 Wrong CCRCFAIL status after a response without CRC is received

Description

The CRC is calculated even if the response to a command does not contain any CRC field.

As a consequence, after the SDIO command IO_SEND_OP_COND (CMD5) is sent, the CCRCFAIL bit of the SDIO_STA register is set.

Workaround

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

2.13.3 SDIO clock divider BYPASS mode may not work properly

Description

In high speed communication mode, when SDIO_CK is equal to 48 MHz (PLL48_output = 48 MHz), the BYPASS bit is equal to ‘1’ and the NEGEDGE bit is equal to ‘0’ (respectively bit 10 and bit 13 in the SDIO_CLKCR register), the hold timing at the I/O pin is not aligned with the SD/MMC 2.0 specifications.

Workaround

When not using USB nor RNG, PLL48_output (SDIOCLK) frequency can be raised up to 75 MHz, allowing to reach 37.5 MHz on SDIO_CK in high speed mode. The BYPASS bit, the CLKDIV bit and the NEGEDGE bit are equal to ‘0’.

2.13.4 Data corruption in SDIO clock dephasing (NEGEDGE) mode

Description

When NEGEDGE bit is set to ‘1’, it may lead to invalid data and command response read.

Workaround

None. A configuration with the NEGEDGE bit equal to ‘1’ must not be used.

2.13.5 CE-ATA multiple write command and card busy signal management

Description

The CE-ATA card may inform the host that it is busy by driving the SDIO_D0 line low, two cycles after the transfer of a write command (RW_MULTIPLE_REGISTER or RW_MULTIPLE_BLOCK). When the card is in a busy state, the host must not send any data until the BUSY signal is de-asserted (SDIO_D0 released by the card).

This condition is not respected if the data state machine leaves the IDLE state (Write operation programmed and started, DTEN = 1, DTDIR = 0 in SDIO_DCTRL register and TXFIFOE = 0 in SDIO_STA register).

As a consequence, the write transfer fails and the data lines are corrupted.

Workaround

After sending the write command (RW_MULTIPLE_REGISTER or RW_MULTIPLE_BLOCK), the application must check that the card is not busy by polling the BSY bit of the ATA status register using the FAST_IO (CMD39) command before enabling the data state machine.

2.13.6 No underrun detected with wrong data transmission

Description

In case there is an ongoing data transfer from the SDIO host to the SD card and the hardware flow control is disabled (bit 14 of the SDIO_CLKCR is not set), if an underrun condition occurs, the controller may transmit a corrupted data block (with wrong data word) without detecting the underrun condition when the clock frequencies have the following relationship:

[3 x period(PCLK2) + 3 x period(SDIOCLK)] >= (32 / (BusWidth)) x period(SDIO_CK)

Workaround

Avoid the above-mentioned clock frequency relationship, by:
- Incrementing the APB frequency
- or decreasing the transfer bandwidth
- or reducing SDIO_CK frequency

2.14 ADC limitations

2.14.1 ADC sequencer modification during conversion

Description

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

Workaround

When an ADC conversion sequence is started by software, a new conversion sequence can be restarted only by setting the SWSTART bit in the ADC_CR2 register.

2.15 DAC limitations

2.15.1 DMA underrun flag management

Description

If the DMA is not fast enough to input the next digital data to the DAC, as a consequence, the same digital data is converted twice. In these conditions, the DMAUDR flag is set, which usually leads to disable the DMA data transfers. This is not the case: the DMA is not disabled by DMAUDR=1, and it keeps servicing the DAC.

Workaround

To disable the DAC DMA stream, reset the EN bit (corresponding to the DAC DMA stream) in the DMA_SxCR register.

2.15.2 DMA request not automatically cleared by DMAEN=0

Description

if the application wants to stop the current DMA-to-DAC transfer, the DMA request is not automatically cleared by DMAEN=0, or by DACEN=0.

If the application stops the DAC operation while the DMA request is high, the DMA request will be pending while the DAC is reinitialized and restarted; with the risk that a spurious unwanted DMA request is serviced as soon as the DAC is re-enabled.

Workaround

To stop the current DMA-to-DAC transfer and restart, apply the following sequence:

1. Check if DMAUDR is set
2. Clear the DAC/DMAEN bit
3. Clear the EN bit of the DAC DMA/Stream
4. Reconfigure by software the DAC, DMA, triggers etc.
5. Restart the application

3 Revision history

Table 6. Document revision history

Date	Revision	Changes
03-Jun-2011	1	Initial release.
20-Dec-2011	2	Removed cut number in the whole document. Added *Section 2.1.1: ART Accelerator prefetch queue instruction is not supported*, *Section 2.2.1: RVU and PVU flags are not reset in STOP mode*, *Section 2.1.7: Configuration of PH10 and PI10 as external interrupts is erroneous*, and *Section 2.13.2: Wrong CCRCFAIL status after a response without CRC is received*. Updated *Section 2.7.5: nRTS signal abnormally driven low after a protocol violation*, *Section 2.11.6: MCO PLL clock pins not compatible with Ethernet IEEE802.3 long term jitter specifications*,
03-Aug-2012	3	Added *Section 2.1.8: DMA2 data corruption when managing AHB and APB peripherals in a concurrent way*, *Section 2.1.9: Slowing down APB clock during a DMA transfer*, *Section 2.1.10: MPU attribute to RTC and IWDG registers could be managed incorrectly*, *Section 2.1.11: Delay after an RCC peripheral clock enabling*, *Section 2.1.12: Battery charge monitoring lower than 2.4 V* and *Section 2.1.13: Internal noise impacting the ADC accuracy*. Added *Section 2.12.2: FSMC synchronous mode and NWAIT signal disabled*. Added *Section 2.13.3: SDIO clock divider BYPASS mode may not work properly*, *Section 2.13.4: Data corruption in SDIO clock dephasing (NEGEDGE) mode* and *Section 2.13.5: CE-ATA multiple write command and card busy signal management*. Added *Section 2.15: DAC limitations* with *Section 2.15.1: DMA underrun flag management* and *Section 2.15.2: DMA request not automatically cleared by DMAEN=0*.
14-May-2013	4	Added silicon revision “1”. Added *Section 2.1.4: Wakeup sequence from Standby mode when using more than one wakeup source*. Added *Section 2.11.5: Successive write operations to the same register might not be fully taken into account*. Updated description in *Section 2.12.1: Dummy read cycles inserted when reading synchronous memories*. Added *Section 2.12.3: FSMC NOR Flash/PSRAM controller asynchronous access on bank 2 to 4 when bank 1 is in synchronous mode (CBURSTRW bit is set)*. Added *Section 2.13.6: No underrun detected with wrong data transmission* and *Section 2.14.1: ADC sequencer modification during conversion*. Updated disclaimer at the end of the document.
02-Aug-2013	5	Added *Section 2.1.5: Full JTAG configuration without NJTRST pin cannot be used* and *Section 2.4.5: Both SDA and SCL maximum rise time (t_r) violated when VDD_I2C bus is higher than ((VDD + 0.3) / 0.7) V*.
03-Dec-2013	6	Added *Section 2.8.1: bxCAN time triggered communication mode not supported*

Table 6. Document revision history (continued)

Date	Revision	Changes
28-Jan-2015	7	Added reference to revisions “V” and “2”. Removed Appendix A: Revision code on device marking. Updated *Silicon identification* and *Section 1: Arm® 32-bit Cortex®-M3 limitations*. Updated *Table 1: Device identification* and *Table 4: Summary of silicon limitations*. Updated footnote *2* of *Table 1*. Added *Section 2.7.6: Start bit detected too soon when sampling for NACK signal from the smartcard*, *Section 2.7.7: Break request can prevent the Transmission Complete flag (TC) from being set*, *Section 2.7.8: Guard time is not respected when data are sent on TXE events* and *Section 2.7.9: nRTS is active while RE or UE = 0*.
24-Apr-2019	8	Updated *Table 4: Summary of silicon limitations*. Added *Section 2.1.14: RDP level 2 and sector write protection configuration*.
25-May-2020	9	Updated *Table 4: Summary of silicon limitations*. Added *Section 2.1.15: Possible delay in backup domain protection disabling/enabling after programming the DBP bit*, *Section 2.6: SPI peripheral limitations* and its subsections. Minor text edits across the whole document.
17-Dec-2020	10	Added revision 3, 4 and 5.
26-Jan-2021	11	Updated workaround in *Section 2.1.5: Full JTAG configuration without NJTRST pin cannot be used*. Added *Section 2.3: RTC peripheral limitation*. Added *Section 2.9.2: Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_FS registers* and *Section 2.9.3: Host packet transmission may hang when connecting through a hub to a low-speed device*. Added OTG_HS limitations: *Section 2.10.1: Transmit data FIFO is corrupted when a write sequence to the FIFO is interrupted with accesses to certain OTG_HS registers* and *Section 2.10.2: Host packet transmission may hang when connecting through a hub to a low-speed device*.