STM32F40x and STM32F41x Errata sheet

STM32F405/407xx and STM32F415/417xx device limitations

Silicon identification

This errata sheet applies to the revision A, Z, 1, 2 and Y of STMicroelectronics STM32F405/415 and STM32F407/417 lines. In this document, they will be referred to as STM32F40x and STM32F41x, respectively, unless otherwise specified.

The STM32F40x and STM32F41x devices feature an ARM® 32-bit Cortex®-M4 core with FPU, 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)
STM32F405/415	"A", "Z", "1", "2" and "Y"
STM32F407/417	

1. The REV_ID bits in the DBGMCU_IDCODE register show the revision code of the device (see the RM0090 STM32F4xx reference manual for details on how to find the revision code).
2. Refer to the device datasheets for details on how to identify the revision code and the date code on the different packages.

Table 2. Device summary

Reference	Part number
STM32F415xx	STM32F415OG, STM32F415RG, STM32F415VG, STM32F415ZG
STM32F405xx	STM32F405OG, STM32F405OE, STM32F405RG, STM32F405VG, STM32F405ZG
STM32F407xx	STM32F407IG, STM32F407VG, STM32F407ZG, STM32F407ZE, STM32F407IE, STM32F407VE
STM32F417xx	STM32F417VG, STM32F417IG, STM32F417ZG, STM32F417VE, STM32F417ZE, STM32F417IE

Contents

List of tables

1 ARM® 32-bit Cortex®-M4 with FPU limitations

An errata notice of the STM32F40x and STM32F41x core is available from http://infocenter.arm.com.

All the described limitations are minor and related to the revision r0p1-v1 of the Cortex®-M4 core. *Table 3* summarizes these limitations and their implications on the behavior of STM32F40x and STM32F41x devices.

Table 3. Cortex®-M4 core limitations and impact on microcontroller behavior

ARM ID	ARM category	ARM summary of errata	Impact on STM32F40x and STM32F41x
752770	Cat B	Interrupted loads to SP can cause erroneous behavior	Minor
776924	Cat B	VDIV or VSQRT instructions might not complete correctly when very short ISRs are used	Minor

1.1 Cortex®-M4 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 cause 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 limitations 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

1.2 VDIV or VSQRT instructions might not complete correctly when very short ISRs are used

Description

On Cortex®-M4 with FPU core, 14 cycles are required to execute a VDIV or VSQRT instruction.

This limitation is present when the following conditions are met:

- A VDIV or VSQRT is executed
- The destination register for VDIV or VSQRT is one of s0 - s15
- An interrupt occurs and is taken
- The ISR being executed does not contain a floating point instruction
- 14 cycles after the VDIV or VSQRT is executed, an interrupt return is executed

In this case, if there are only one or two instructions inside the interrupt service routine, then the VDIV or VQSRT instruction does not complete correctly and the register bank and FPSCR are not updated, meaning that these registers hold incorrect out-of-date data.

Workaround

Two workarounds are applicable:

- Disable lazy context save of floating point state by clearing LSPEN to 0 (bit 30 of the FPCCR at address 0xE000EF34).
- Ensure that every ISR contains more than 2 instructions in addition to the exception return instruction.

2 STM32F40x and STM32F41x 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.

Table 4. Summary of silicon limitations

Links to silicon limitations		Revision A	Revision 'Z', '1', '2' and 'Y'
Section 2.1: System limitations	*Section 2.1.1: ART Accelerator prefetch queue instruction is not supported*	N	-
	Section 2.1.2: MCU device ID is incorrect	A	-
	Section 2.1.3: Debugging Sleep/Stop mode with WFE/WFI entry	A	A
	Section 2.1.4: Debugging Stop mode and system tick timer	A	A
	Section 2.1.5: Wakeup sequence from Standby mode when using more than one wakeup source	A	A
	Section 2.1.6: Full JTAG configuration without NJTRST pin cannot be used	A	A
	Section 2.1.7: PDR_ON pin not available on LQFP100 package for revision Z devices	-	N
	Section 2.1.8: Incorrect BOR option byte when consecutively programming BOR option byte	A	A
	Section 2.1.9: Configuration of PH10 and PI10 as external interrupts is erroneous	N	N
	Section 2.1.10: DMA2 data corruption when managing AHB and APB peripherals in a concurrent way	A	A
	Section 2.1.11: Slowing down APB clock during a DMA transfer	A	A
	Section 2.1.12: MPU attribute to RTC and IWDG registers could be managed incorrectly	A	A
	Section 2.1.13: Delay after an RCC peripheral clock enabling	A	A
	Section 2.1.14: Battery charge monitoring lower than 2.4 Volts	P	P
	Section 2.1.15: Internal noise impacting the ADC accuracy	A	A
Section 2.2: IWDG peripheral limitations	*Section 2.2.1: RVU and PVU flags are not reset in Stop mode*	A	A
Section 2.3: RTC limitations	*Section 2.3.1: Spurious tamper detection when disabling the tamper channel*	N	N
	Section 2.3.2: Detection of a tamper event occurring before enabling the tamper detection is not supported in edge detection mode	A	A
	Section 2.3.3: RTC calendar registers are not locked properly	A	A

Table 4. Summary of silicon limitations (continued)

Links to silicon limitations		Revision A	Revision 'Z', '1', '2' and 'Y'
Section 2.4: I2C peripheral limitations	Section 2.4.1: SMBus standard not fully supported	A	A
	Section 2.4.2: Start cannot be generated after a misplaced Stop	A	A
	Section 2.4.3: Mismatch on the "Setup time for a repeated Start condition" timing parameter	A	A
	Section 2.4.4: Data valid time (tVD;DAT) violated without the OVR flag being set	A	A
	Section 2.4.5: Both SDA and SCL maximum rise time (tr) violated when VDD_I2C bus higher than ((VDD+0.3) / 0.7) V	A	A
	Section 2.4.6: Spurious Bus Error detection in Master mode	A	A
	Section 2.4.7: Wrong behavior related with MCU Stop mode when wakeup from Stop mode by I2C peripheral is disabled	A	A
Section 2.5: SPI peripheral limitations	Section 2.5.1: Wrong CRC calculation when the polynomial is even	A	A
	Section 2.5.2: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback	A	A
	Section 2.5.3: Wrong CRC transmitted in Master mode with delayed SCK feedback	A	A
	Section 2.5.4: BSY bit may stay high at the end of a data transfer in Slave mode	A	A
Section 2.6: I2S peripheral limitations	Section 2.6.1: In I2S Slave mode, WS level must be set by the external master when enabling the I2S	A	A
	Section 2.6.2: I2S2 in full-duplex mode may not work properly when SCK and WS signals are mapped on PI1 and PI0 respectively	A	A
	Section 2.6.3: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback	A	A
Section 2.7: USART peripheral limitations	Section 2.7.1: Idle frame is not detected if receiver clock speed is deviated	N	N
	Section 2.7.2: In full-duplex mode, the Parity Error (PE) flag can be cleared by writing to the data register	A	A
	Section 2.7.3: Parity Error (PE) flag is not set when receiving in Mute mode using address mark detection	N	N
	Section 2.7.4: Break frame is transmitted regardless of nCTS input line status	N	N
	Section 2.7.5: nRTS signal abnormally driven low after a protocol violation	A	A
	Section 2.7.6: Start bit detected too soon when sampling for NACK signal from the smartcard	N	N
	Section 2.7.7: Break request can prevent the Transmission Complete flag (TC) from being set	A	A
	Section 2.7.8: Guard time is not respected when data are sent on TXE events	A	A
	Section 2.7.9: nRTS is active while RE or UE = 0	A	A

Table 4. Summary of silicon limitations (continued)

Links to silicon limitations		Revision A	Revision 'Z', '1', '2' and 'Y'
Section 2.8: bxCAN limitations	*Section 2.8.1: bxCAN time triggered communication mode not supported*	N	N
Section 2.9: OTG_FS peripheral limitations	*Section 2.9.1: Data in RxFIFO is overwritten when all channels are disabled simultaneously*	A	A
	Section 2.9.2: OTG host blocks the receive channel when receiving IN packets and no TxFIFO is configured	A	A
	Section 2.9.3: Host channel-halted interrupt not generated when the channel is disabled	A	A
	Section 2.9.4: Error in software-read OTG_FS_DCFG register values	A	A
Section 2.10: Ethernet peripheral limitations	*Section 2.10.1: Incorrect layer 3 (L3) checksum is inserted in transmitted IPv6 packets without TCP, UDP or ICMP payloads*	A	A
	Section 2.10.2: The Ethernet MAC processes invalid extension headers in the received IPv6 frames	N	N
	Section 2.10.3: MAC stuck in the Idle state on receiving the TxFIFO flush command exactly 1 clock cycle after a transmission completes	A	A
	Section 2.10.4: Transmit frame data corruption	A	A
	Section 2.10.5: Successive write operations to the same register might not be fully taken into account	A	A
Section 2.11: FSMC peripheral limitations	*Section 2.11.1: Dummy read cycles inserted when reading synchronous memories*	N	N
	Section 2.11.2: FSMC synchronous mode and NWAIT signal disabled	A	A
	Section 2.11.3: FSMC NOR Flash/PSRAM controller asynchronous access on bank 2 to 4 when bank 1 is in synchronous mode (CBURSTRW bit is set)	A	A
Section 2.12: SDIO peripheral limitations	*Section 2.12.1: SDIO HW flow control*	N	N
	Section 2.12.2: Wrong CCRCFAIL status after a response without CRC is received	A	A
	Section 2.12.3: SDIO clock divider BYPASS mode may not work properly	A	A
	Section 2.12.4: Data corruption in SDIO clock dephasing (NEGEDGE) mode	N	N
	Section 2.12.5: CE-ATA multiple write command and card busy signal management	A	A
	Section 2.12.6: No underrun detection with wrong data transmission	A	A
Section 2.13: ADC peripheral limitations	*Section 2.13.1: ADC sequencer modification during conversion*	A	A
Section 2.14: DAC peripheral limitations	*Section 2.14.1: DMA underrun flag management*	A	A
	Section 2.14.2: DMA request not automatically cleared by DMAEN=0	A	A

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 on revision A devices.

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

- Revision A devices: none
- Revision Z and 1 devices: fixed.

2.1.2 MCU device ID is incorrect

Description

On revision A devices, the STM32F40x and STM32F41x have the same MCU device ID as the STM32F20x and STM32F21x devices. On revision A devices, when reading the Revision identifier, this will return 0x2000 instead of 0x1000. The device ID and revision ID can be read from address 0xE0042000.

Workaround

- Revision A devices

  To differentiate the STM32F4xxx from the STM32F2xxx series, read the MCU device ID and the Core Device.
  - For STM32F2xxx

    MCU device ID = STM32F2xxx device ID

    Core Device = Cortex®-M3
  - For STM32F4xxx

    MCU device ID = STM32F4xxx device ID

    Core Device = Cortex®-M4
- Revision Z and 1 devices: fixed.

2.1.3 Debugging Sleep/Stop mode with WFE/WFI entry

Description

When the Sleep debug or Stop debug mode is enabled (DBG_SLEEP bit or DBG_STOP bit are set in the DBGMCU_CR register), this allows software debugging during Sleep or Stop mode. After wakeup some unreachable instructions could be executed if the following condition are met:

- If the application software disables the Prefetch queue
- The number of wait state configured on Flash interface is higher than 0
- And Linker place WFE or WFI instructions on 4-bytes aligned addresses (0x080xx_xxx4)

Workaround

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

2.1.4 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.

2.1.5 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 which 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

To avoid this problem, the following sequence should 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.

2.1.6 Full JTAG configuration without NJTRST pin cannot be used

Description

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

Workaround

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

2.1.7 PDR_ON pin not available on LQFP100 package for revision Z devices

Description

On revision Z devices, the PDR_ON pin (pin 99) available on LQFP100 package is replaced by V_{SS}. As a consequence, the POR/PDR feature is always enabled.

Workaround

- Applications using on revision A devices with PDR_ON pin connected to V_{DD} (POR/PDR feature enabled)

  Connect the former PDR_ON pin to V_{SS} on revision Z devices.
- Applications using revision A devices with PDR_ON pin connected to V_{SS} (POR/PDR feature disabled)

  No modification is required when migrating to revision Z devices. However, it is no longer possible to supply the product from a 1.7 V V_{DD} on LQFP100 package since V_{DD} minimum value is 1.8 V when the POR/PDR feature is enabled.

2.1.8 Incorrect BOR option byte when consecutively programming BOR option byte

Description

When the AHB prescaler is greater than 2, and consecutive BOR option byte program operations are performed without resetting the device, then an incorrect value might be programmed in the BOR option byte.

Workaround

To program consecutive BOR option byte values, either configure the AHB prescaler to 1 or 2, or perform a system reset between each BOR option byte program operation.

2.1.9 Configuration of PH10 and PI10 as external interrupts is erroneous

Description

PH10 or PI10 is selected as the source for the 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 the interrupt source, as well. This means that bits EXTI10[3:0] of SYSCFG_EXTICR3 register and bits EXTI2[3:0] of SYSCFG_EXTICR1 should be programmed to the same value:

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

Workaround

None.

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

Description

When the DMA2 is managing AHB Peripherals (read- or write-sensitive devices such as peripherals embedding FIFOs or GPIOs) 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. Also we can consider external FIFO controlled by the FSMC and GPIO Output Data register as AHB write sensitive peripherals.

On sales types without CRYPTO, mainly impacted peripheral is the DCMI peripheral which embeds a FIFO. External FIFO controlled by the FSMC and GPIOs when used as parallel output are also impacted.

Workaround

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

2.1.11 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 will be blocked. Only system reset will recover.

Workaround

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

2.1.12 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.

2.1.13 Delay after an RCC peripheral clock enabling

Description

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

This delay depends on the peripheral mapping:

- If the peripheral is mapped on AHB: the delay should be equal to 2 AHB cycles.
- If the peripheral is mapped on APB: the delay should be equal to 1 + (AHB/APB prescaler) cycles.

Workarounds

1. Use the DSB instruction to stall the Cortex®-M4 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).
3. Or simply insert a dummy read operation from the corresponding register just after enabling the peripheral clock.

2.1.14 Battery charge monitoring lower than 2.4 Volts

Description

If (V_{DD} = V_{DDA}) is lower than or equal to 2.4 V, the V_{BAT} conversion correctness is not guaranteed in 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}) should be greater than 2.4 V.

2.1.15 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.

2.2 IWDG peripheral limitations

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 limitations

2.3.1 Spurious tamper detection when disabling the tamper channel

Description

If the tamper detection is configured for detection on 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

2.3.2 Detection of a tamper event occurring before enabling the tamper detection is not supported in edge detection mode

Description

When the tamper detection is enabled in edge detection mode (TAMPFLT=00):

- When TAMPxTRG=0 (rising edge detection): if the tamper input is already high before enabling the tamper detection, the tamper event may or may not be detected when enabling the tamper detection. The probability to detect it increases with the APB frequency.
- When TAMPxTRG=1 (falling edge detection): if the tamper input is already low before enabling the tamper detection, the tamper event is not detected when enabling the tamper detection.

Workaround

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

2.3.3 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, RTC_DR register can be updated after reading the RTC_TR register instead of being locked.

Workaround

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

2.4 I2C peripheral limitations

2.4.1 SMBus standard not fully supported

Description

The I^{2}C 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) while the microcontroller I2C peripheral attempts to switch to Master mode by setting the START bit, the Start condition is not properly generated.

Workaround

In the I^{2}C 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^{2}C 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 limitation can occur only in the following configuration:

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

- in Slave transmit mode
- with clock stretching disabled (NOSTRETCH=1)
- if the software is late to write 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 VDD_I2C bus higher than ((VDD+0.3) / 0.7) V

Description

When an external legacy I^2C bus voltage (VDD_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 0.7VDD_I2C.

Workaround

The external VDD_I2C bus voltage should be limited to a maximum value of ((VDD+0.3) / 0.7) V. As a result, when the MCU is powered from V_{DD}=3.3 V, VDD_I2C should not exceed 5.14 V to be compliant with I^2C specifications.

2.4.6 Spurious Bus Error detection in Master mode

Description

In Master mode, a bus error can be detected by mistake, so the BERR flag can be wrongly raised in the status register. This will generate a spurious Bus Error interrupt if the interrupt is enabled. A bus error detection has no effect on the transfer in Master mode, therefore the I2C transfer can continue 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 on-going transfer can be handled normally.

2.4.7 Wrong behavior related with MCU Stop mode when wakeup from Stop mode by I2C peripheral is disabled

Description

When wakeup from Stop mode by I2C peripheral is disabled (WUPEN = 0) and the MCU enters Stop mode while a transaction is on-going on the I²C bus, the following wrong operation may occur: 1. BUSY flag may be 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. This failure may occur in Master mode of the I2C peripheral used in multi-master I²C-bus environment. 2. If I²C-bus clock stretching is enabled in I2C peripheral (NOSTRETCH = 0), the I2C peripheral may pull SCL low as long as the MCU remains in Stop mode, suspending all I²C-bus activity during that time. This may occur when the MCU enters Stop mode during the address phase of an I²C-bus transaction, in low period of SCL. This failure may occur in Slave mode of the I2C peripheral or, in Master mode of the I2C peripheral used in multi-master I²C-bus environment. Its probability depends on the timing configuration, operating clock frequency of I2C peripheral and the I²C-bus timing.

Workaround

Disable the I2C peripheral (PE=0) before entering Stop mode and re-enable it in Run mode.

2.5 SPI peripheral limitations

2.5.1 Wrong CRC calculation when the polynomial is even

Description

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

Work-around

Use odd polynomial.

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

Description

In receive transaction, in both I^2S and SPI Master modes, the last bit of the transacted frame is not captured when the signal provided by internal feedback loop from the SCK pin exceeds a critical delay. The lastly transacted bit of the stored data then keeps the value from the pattern received previously. As a consequence, the last receive data bit may be wrong and/or the CRCERR flag can be unduly asserted in the SPI mode if any data under check sum and/or just the CRC pattern is wrongly captured.

In SPI mode, data are synchronous with the APB clock. A delay of up to two APB clock periods can thus be tolerated for the internal feedback delay. The I^2S mode is more sensitive than the SPI mode since the SCK clock is not synchronized with the APB. In this case, the margin of the internal feedback delay is lower than one APB clock period.

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

Workarounds

The following workaround can be adopted, jointly or individually:

- Decrease the APB clock speed.
- Configure the IO pad of the SCK pin to be faster.

The following table gives the maximum allowable APB frequency versus GPIOx_OSPEEDR output speed control field setting for the SCK pin, at 30 pF of capacitive load.

Table 5. Maximum allowable APB frequency at 30 pF load

OSPEEDR [1:0] for SCK pin	Max. APB frequency for SPI mode [MHz]	Max. APB frequency for I^2S mode [MHz]
11 (very high), 10 (high)	84	42
01 (medium)	75	35
00 (low)	25	16

2.5.3 Wrong CRC transmitted in Master mode with delayed SCK feedback

Description

In transmit transaction of the SPI/I^2S interface in SPI Master mode with CRC enabled, the CRC data transmission may be corrupted if the delay of an internal feedback signal derived from the SCK output (further feedback clock) is greater than two APB clock periods. While data and CRC bit shifting and transfer is based on an internal clock, the CRC progressive calculation uses the feedback clock. If the delay of the feedback clock is greater than two APB periods, the transmitted CRC value may get wrong.

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

Workaround

The following workaround can be adopted, jointly or individually:

- Decrease the APB clock speed.
- Configure the IO pad of the SCK pin to be faster.

The following table gives the maximum allowable APB frequency versus GPIOx_OSPEEDR output speed control field setting for the SCK pin, at 30 pF of capacitive load.

Table 6. Maximum allowable APB frequency at 30 pF load

OSPEEDR [1:0] for SCK pin	Max. APB frequency for SPI mode [MHz]	Max. APB frequency for I^2S mode [MHz]
11 (very high), 10 (high)	84	42
01 (medium)	75	35
00 (low)	25	16

2.5.4 BSY bit may stay high at the end of a data transfer in Slave mode

Description

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

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

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

Workaround

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

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

When upper workarounds are not applicable, the following sequence can be used to prevent the synchronization issue at SPI transmit mode.

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

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

2.6 I2S peripheral limitations

2.6.1 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.

2.6.2 I2S2 in full-duplex mode may not work properly when SCK and WS signals are mapped on PI1 and PI0 respectively

Description

When SCK and WS signals are used to support I2S full-duplex through GPIO port I: PI1 and PI0 respectively, the I2S2 peripheral cannot be able to provide internally SCK signal and WS signal to I2S2_ext interface. In this case, I2S2_ext interface will not be able to send/receive data.

Workaround

Other mapped pins for SCK and WS signals can be used on GPIO Port B as below:

- I2S2 CK signal: PB10 pin or PB13 pin.
- I2S2 WS signal: PB12 pin or PB9 pin.

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

The limitation described in *Section 2.5.2: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback* also applies to I^{2}S interface.

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 should 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 shall transmit a NACK error signal 10.5 ± 0.2 ETUs after the character START bit falling edge. In this case, the transmitter should be able to detect correctly the NACK signal until 11 ± 0.2 ETUs after the character START bit falling edge.

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

Workaround

None

2.7.7 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 should 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 guardtime 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 limitations

2.8.1 bxCAN time triggered communication mode not supported

Description

The time triggered communication mode described in the reference manual is not supported. As a result timestamp values are not available. 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 is 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 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.

2.9.3 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.

2.9.4 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.

2.10 Ethernet peripheral limitations

2.10.1 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).

2.10.2 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.

2.10.3 MAC stuck in the Idle state on receiving the TxFIFO flush command exactly 1 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.

2.10.4 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.

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

Description

A write 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 7: Impacted registers and bits* for the registers and bits impacted by this limitation.

Table 7. 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

Table 7. Impacted registers and bits (continued)

Register name	Bit number	Bit name
ETH_MACHTLR	31:0	Hash Table Low Register
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
IEEE 1588 time stamp registers		

Table 7. Impacted registers and bits (continued)

Register name	Bit number	Bit name
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 finally reprogram it after a delay of four TX_CLK/RX_CLK clock cycles.

2.11 FSMC peripheral limitations

2.11.1 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.

Workaround

None.

2.11.2 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).

2.11.3 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 1 of NOR/PSRAM controller is enabled in synchronous write mode (CBURSTRW bit set), while any other NOR/PSRAM banks (2 to 4) are enabled in asynchronous mode, two limitations 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 limitations 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 for bank 1.

2.12 SDIO peripheral limitations

2.12.1 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.

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

2.12.2 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.

2.12.3 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’.

2.12.4 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’ should not be used.

2.12.5 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.

2.12.6 No underrun detection 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

2.13 ADC peripheral limitations

2.13.1 ADC sequencer modification during conversion

Description

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

Workaround

When a software start of conversion is used, the user application must first set the SWSART bit in the ADC_CR2 register, and then restart the new conversion sequence.

2.14 DAC peripheral limitations

2.14.1 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.

2.14.2 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, the following sequence should be applied:

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 8. Document revision history

Date	Revision	Changes
19-Sep-2011	1	Initial release.
12-Dec-2011	2	Replaced STM42F4xx by STM32F4xx on cover page. Added silicon revision Z. Modified link to ARM 32-bit Cortex-M4F errata notice in *Section 1: ARM® 32-bit Cortex®-M4 with FPU limitations*. Updated status of ART Accelerator prefetch queue and MCU device ID limitations for revision Z in *Table 4: Summary of silicon limitations* Updated *Section 2.1.1: ART Accelerator prefetch queue instruction is not supported* and *Section 2.1.2: MCU device ID is incorrect* to make differentiate between revision A and revision Z devices. Added *Section 2.1.6: Full JTAG configuration without NJTRST pin cannot be used*, *Section 2.1.7: PDR_ON pin not available on LQFP100 package for revision Z devices*, *Section 2.1.8: Incorrect BOR option byte when consecutively programming BOR option byte*, and *Section 2.1.9: Configuration of PH10 and PI10 as external interrupts is erroneous*. Updated workaround for *Section 2.7.5: nRTS signal abnormally driven low after a protocol violation*. Added *Section 2.12.2: Wrong CCRCFAIL status after a response without CRC is received* and *Section 2.2.1: RVU and PVU flags are not reset in Stop mode*.
03-Aug-2012	3	Added *Section : None.*, *Section 2.1.11: Slowing down APB clock during a DMA transfer*, *Section 2.1.12: MPU attribute to RTC and IWDG registers could be managed incorrectly*, *Section 2.1.13: Delay after an RCC peripheral clock enabling*, *Section 2.1.14: Battery charge monitoring lower than 2.4 Volts* and *Appendix A: Revision code on device marking*. Added *Section 2.11.2: FSMC synchronous mode and NWAIT signal disabled*. Added *Section 2.12.3: SDIO clock divider BYPASS mode may not work properly*, *Section 2.12.4: Data corruption in SDIO clock dephasing (NEGEDGE) mode* and *Section 2.12.5: CE-ATA multiple write command and card busy signal management*. Added *Section 2.14: DAC peripheral limitations* with *Section 2.14.1: DMA underrun flag management* and *Section 2.14.2: DMA request not automatically cleared by DMAEN=0*.

Table 8. Document revision history (continued)

Date	Revision	Changes
25-Apr-2013	4	Added *Section 1.2: VDIV or VSQRT instructions might not complete correctly when very short ISRs are used* Removed the reference to 'Cortex-M4F' in the whole document. Updated *Table 2: Device summary*, *Section 2.1.2: MCU device ID is incorrect*. Added *Section 2.1.5: Wakeup sequence from Standby mode when using more than one wakeup source*. Updated *Section 2.11.1: Dummy read cycles inserted when reading synchronous memories*. Added *Section 2.2: IWDG peripheral limitations*, *Section 2.6.2: I2S2 in full-duplex mode may not work properly when SCK and WS signals are mapped on PI1 and PI0 respectively*, *Section 2.10.5: Successive write operations to the same register might not be fully taken into account* and *Section 2.11.3: FSMC NOR Flash/PSRAM controller asynchronous access on bank 2 to 4 when bank 1 is in synchronous mode (CBURSTRW bit is set)* , *Section 2.12.6: No underrun detection with wrong data transmission* and *Section 2.13.1: ADC sequencer modification during conversion*. Added *Figure 6: WLCSP90 top package view*.
11-Oct-2013	5	Added silicon revision 1. Added *Section 2.4.5: Both SDA and SCL maximum rise time (tr) violated when VDD_I2C bus higher than ((VDD+0.3) / 0.7) V*. Moved device marking to datasheets.
21-Jan-2015	6	Added: rev 2 and Y on *Table 1: Device identification*, *Section 1.2: VDIV or VSQRT instructions might not complete correctly when very short ISRs are used* *Section 2.3: RTC limitations* from *Section 2.7.6: Start bit detected too soon when sampling for NACK signal from the smartcard* to *Section 2.7.9: nRTS is active while RE or UE = 0* *Section 2.8: bxCAN limitations* Updated *Table 4: Summary of silicon limitations*

Table 8. Document revision history (continued)

Date	Revision	Changes
14-Sep-2015	7	Added:*Section 2.5: SPI peripheral limitations*, *Section 2.5.1: Wrong CRC calculation when the polynomial is even* *Section 2.4.6: Spurious Bus Error detection in Master mode*. *Section 2.4.7: Wrong behavior related with MCU Stop mode when wakeup from Stop mode by I2C peripheral is disabled*. *Section 2.5.4: BSY bit may stay high at the end of a data transfer in Slave mode*. *Section 2.5.2: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback*. Updated:*Section 2.1.10: DMA2 data corruption when managing AHB and APB peripherals in a concurrent way*. Replaced *Section 2.1.5: Wakeup sequence from Standby mode when using more than one wakeup source* with *Section 2.1.3: Debugging Sleep/Stop mode with WFE/WFI entry*.
24-Nov-2016	8	Added workaround in *Section 2.1.13: Delay after an RCC peripheral clock enabling*. Added *Section 2.3.3: RTC calendar registers are not locked properly*. Updated *Section 2.5.2: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback* and added *Section 2.5.3: Wrong CRC transmitted in Master mode with delayed SCK feedback*. Updated *Section 2.5.4: BSY bit may stay high at the end of a data transfer in Slave mode*. Updated limitation description in *Section 2.4.2: Start cannot be generated after a misplaced Stop*. Added *Section 2.6.3: Corrupted last bit of data and/or CRC, received in Master mode with delayed SCK feedback* in *Section 2.6: I2S peripheral limitations*.