STM32F469xx and STM32F479xx Errata sheet

STM32F469xx and STM32F479xx line limitations

Applicability

This document applies to the part numbers of STM32F469xx and STM32F479xx devices listed in *Table 1*, and to their variants, shown in *Table 2*.

Section 1 gives a summary and *Section 2* a description of workarounds for device limitations, with respect to the device datasheet and reference manual RM0386.

Table 1. Device summary

Reference	Part numbers
STM32F469xx	STM32F469AE, STM32F469AG, STM32F469AI STM32F469BE, STM32F469BG, STM32F469BI STM32F469IE, STM32F469IG, STM32F469II STM32F469NE, STM32F469NG, STM32F469NI STM32F469VE, STM32F469VG, STM32F469VI STM32F469ZE, STM32F469ZG, STM32F469ZI
STM32F479xx	STM32F479AI, STM32F479AG STM32F479BI, STM32F479BG STM32F479II, STM32F479IG STM32F479NI, STM32F479NG STM32F479VI, STM32F479VG STM32F479ZI, STM32F479ZG

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking[1]	REV_ID[2]
STM32F469xx, STM32F479xx	"A", "1"	0x1000

1. Refer to the device datasheet for details on how to identify this code on different types of package.
2. The REV_ID[15:0] bit field of DBGMCU_IDC register (refer to the reference manual RM0386).

Contents

List of tables

1 Summary of device limitations

Table 3 gives quick references to all documented device limitations of STM32F469xx and STM32F479xx, and their status:

- A = limitation present, workaround available
- N = limitation present, no workaround available
- P = limitation present, partial workaround available
- '-' = limitation absent

Applicability of a workaround may depend on specific conditions of the target application. Adoption of a workaround may cause restrictions to the target application. Workaround for a limitation is deemed partial if it only reduces the rate of occurrence and/or the consequences of the limitation, or if it is fully effective for only a subset of instances on the device or in only a subset of operating modes of the function concerned.

Table 3. Summary of silicon limitations

Function	Section	Limitation	Revisions "A" and "1"
Core	*2.1.1*	*Interrupted loads to stack pointer can cause erroneous behavior*	A
	2.1.2	*VDIV or VSQRT instructions might not complete correctly when very short ISRs are used*	A
	2.1.3	*Store immediate overlapping exception return operation might vector to incorrect interrupt*	A
System	*2.2.1*	*Debugging Stop mode and system tick timer*	A
	2.2.2	*Debugging Sleep/Stop mode with WFE/WFI entry*	A
	2.2.3	*Full JTAG configuration without NJTRST pin cannot be used*	A
	2.2.4	*MPU attribute to RTC and IWDG registers could be managed incorrectly*	A
	2.2.5	*Delay after an RCC peripheral clock enabling*	A
	2.2.6	*Internal noise impacting the ADC accuracy*	A
	2.2.7	*Wakeup from Standby mode with RTC*	A
	2.2.8	*Data Cache might be corrupted during Flash Read While Write operation*	A
	2.2.9	*FMC_SDNWEN alternate function on PA7*	A
FMC	*2.3.1*	*Dummy read cycles inserted when reading synchronous memories*	N
QUADSPI	*2.4.1*	*Extra data written in the FIFO at the end of a read transfer*	A
	2.4.2	*First nibble of data is not written after dummy phase*	A
	2.4.3	*Wrong data can be read in memory-mapped after an indirect mode operation*	A
	2.4.4	*Memory-mapped read operations may fail when timeout counter is enabled*	A

Table 3. Summary of silicon limitations (continued)

Function	Section	Limitation	Revisions “A” and “1”
ADC	*2.5.1*	*ADC sequencer modification during conversion*	A
DAC	*2.6.1*	*DMA underrun flag management*	A
	2.6.2	*DMA request not automatically cleared by DMAEN=0*	A
DSI Host	*2.7.1*	*When used over the DSI link, the Tearing Effect Interrupt Flag is set when an Acknowledge Trigger is received from the display*	A
	2.7.2	*The time to activate the clock between HS transmissions is not calculated correctly*	A
	2.7.3	*The immediate update procedure may fail*	A
IWDG	*2.8.1*	*RVU and PVU flags are not reset in STOP mode*	A
RTC	*2.9.1*	*Spurious tamper detection when disabling the tamper channel*	A
	2.9.2	*Detection of a tamper event occurring before enabling the tamper detection is not supported in edge detection mode*	A
I2C	*2.10.1*	*SMBus standard not fully supported*	A
	2.10.2	*Start cannot be generated after a misplaced Stop*	A
	2.10.3	*Mismatch on the “Setup time for a repeated Start condition” timing parameter*	A
	2.10.4	*Data valid time (tVD;DAT) violated without the OVR flag being set*	A
	2.10.5	*Both SDA and SCL maximum rise time (tr) violated when VDD_I2C bus is higher than ((VDD + 0.3) / 0.7) V*	A
	2.10.6	*Spurious Bus Error detection in master mode*	A
USART	*2.11.1*	*Idle frame is not detected if receiver clock speed is deviated*	N
	2.11.2	*In full duplex mode, the Parity Error (PE) flag can be cleared by writing to the data register*	A
	2.11.3	*Parity Error (PE) flag is not set when receiving in Mute mode using address mark detection*	N
	2.11.4	*Break frame is transmitted regardless of nCTS input line status*	N
	2.11.5	*nRTS signal abnormally driven low after a protocol violation*	A
	2.11.6	*Start bit detected too soon when sampling for NACK signal from the smartcard*	N
	2.11.7	*Break request can prevent the Transmission Complete flag (TC) from being set*	A
	2.11.8	*Guard time is not respected when data are sent on TXE events*	A
	2.11.9	*nRTS is active while RE or UE = 0*	A

Table 3. Summary of silicon limitations (continued)

Function	Section	Limitation	Revisions “A” and “1”
SPI	*2.12.1*	*BSY bit may stay high at the end of a data transfer in slave mode*	A
	2.12.2	*The last transacted bit of data or CRC calculation can be corrupted for the received data in master mode depending on the timing of the feedback communication clock respect to the APB clock (SPI or I2S)*	A
	2.12.3	*Wrong CRC calculation when the polynomial is even*	A
	2.12.4	*CRC can be corrupted when full duplex is handled by DMA and reception DMA channel is set to number of data frames plus CRC length*	A
I2S	*2.13.1*	*In I2S slave mode WS level must be set by the external master when enabling the I2S*	A
SDIO	*2.14.1*	*Wrong CCRCFAIL status after a response without CRC is received*	A
	2.14.2	*No underrun detection with wrong data transmission*	A
bxCAN	*2.15.1*	*bxCAN time triggered communication mode not supported*	N
Ethernet	*2.16.1*	*Incorrect layer 3 (L3) checksum is inserted in transmitted IPv6 packets without TCP, UDP or ICMP payloads*	A
	2.16.2	*The Ethernet MAC processes invalid extension headers in the received IPv6 frames*	N
	2.16.3	*MAC stuck in the Idle state on receiving the TxFIFO flush command exactly one clock cycle after a transmission completes*	A
	2.16.4	*Transmit frame data corruption*	A
	2.16.5	*Successive write operations to the same register might not be fully taken into account*	A

2 Description of device limitations

Table 3 gives quick references to all documented limitations of STM32F469xx and STM32F479xx, and their status:

- A = limitation present, workaround available
- N = limitation present, no workaround available
- P = limitation present, partial workaround available
- '-' = limitation absent

The following sections describe device limitations and provide workarounds if available. They are grouped by device function.

2.1 Core

An errata notice for the Arm®[a] Cortex®-M4 FPU core revisions r0 is available from *http://infocenter.arm.com.*

Only applicable information from the Arm® errata notice is replicated in this document. Extra information may be added for more clarity.

2.1.1 Interrupted loads to stack pointer can cause erroneous behavior

This limitation is registered under Arm® ID number 752770 and classified into "Category B". Its impact to the device is minor.

Description

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 with 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]

---

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

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] with

```
LDR R2,[R0]
MOV SP,R2
```

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

This limitation is registered under Arm® ID number 776924 and classified into “Category B”. Its impact to the device is limited.

Description

The VDIV and VSQRT instructions take 14 cycles to execute. When an interrupt is taken a VDIV or VSQRT instruction is not terminated, and completes its execution while the interrupt stacking occurs. If lazy context save of floating point state is enabled then the automatic stacking of the floating point context does not occur until a floating point instruction is executed inside the interrupt service routine.

Lazy context save is enabled by default. When it is enabled, the minimum time for the first instruction in the interrupt service routine to start executing is 12 cycles. In certain timing conditions, and if there is only one or two instructions inside the interrupt service routine, then the VDIV or VSQRT instruction might not write its result to the register bank or to theFPSCR.

The failure occurs when the following condition is met:

1. The floating point unit is enabled
2. Lazy context saving is not disabled
3. A VDIV or VSQRT is executed
4. The destination register for the VDIV or VSQRT is one of s0 - s15
5. An interrupt occurs and is taken
6. The interrupt service routine being executed does not contain a floating point instruction
7. Within 14 cycles after the VDIV or VSQRT is executed, an interrupt return is executed

A minimum of 12 of these 14 cycles are utilized for the context state stacking, which leaves two cycles for instructions inside the interrupt service routine, or two wait states applied to the entire stacking sequence (which means that it is not a constant wait state for every access).

In general, this means that if the memory system inserts wait states for stack transactions (that is, external memory is used for stack data), then this erratum cannot be observed.

The effect of this erratum is that the VDIV or VQSRT instruction does not complete correctly and the register bank and FPSCR are not updated, which means that these registers hold incorrect (out of date) data.

Workaround

A workaround is only required if the floating point unit is enabled. A workaround is not required if the stack is in external memory.

There are two possible workarounds:

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

2.1.3 Store immediate overlapping exception return operation might vector to incorrect interrupt

This limitation is registered under Arm® ID number 838869 and classified into “Category B (rare)”. Its impact to the device is minor.

Description

The core includes a write buffer that permits execution to continue while a store is waiting on the bus. Under specific timing conditions, during an exception return while this buffer is still in use by a store instruction, a late change in selection of the next interrupt to be taken might result in there being a mismatch between the interrupt acknowledged by the interrupt controller and the vector fetched by the processor.

The failure occurs when the following condition is met:

1. The handler for interrupt A is being executed.
2. Interrupt B, of the same or lower priority than interrupt A, is pending.
3. A store with immediate offset instruction is executed to a bufferable location.
   - STR/STRH/STRB <Rt>, [<Rn>,#imm]
   - STR/STRH/STRB <Rt>, [<Rn>,#imm]!
   - STR/STRH/STRB <Rt>, [<Rn>],#imm
4. Any number of additional data-processing instructions can be executed.
5. A BX instruction is executed that causes an exception return.
6. The store data has wait states applied to it such that the data is accepted at least two cycles after the BX is executed.
   - Minimally, this is two cycles if the store and the BX instruction have no additional instructions between them.
   - The number of wait states required to observe this erratum needs to be increased by the number of cycles between the store and the interrupt service routine exit instruction.
7. Before the bus accepts the buffered store data, another interrupt C is asserted which has the same or lower priority as A, but a greater priority than B.

Example:

The processor should execute interrupt handler C, and on completion of handler C should execute the handler for B. If the conditions above are met, then this erratum results in the processor erroneously clearing the pending state of interrupt C, and then executing the handler for B twice. The first time the handler for B is executed it will be at interrupt C's priority level. If interrupt C is pended by a level-based interrupt which is cleared by C's handler then interrupt C will be pended again once the handler for B has completed and the handler for C will be executed.

As the STM32 interrupt C is level based, it eventually becomes pending again and is subsequently handled.

Workaround

For software not using the memory protection unit, this erratum can be worked around by setting DISDEFWBUF in the Auxiliary Control Register.

In all other cases, the erratum can be avoided by ensuring a DSB occurs between the store and the BX instruction. For exception handlers written in C, this can be achieved by inserting the appropriate set of intrinsics or inline assembly just before the end of the interrupt function, for example:

```
ARMCC:
...
__schedule_barrier();
__asm{DSB};
__schedule_barrier();
}
GCC:
...
__asm volatile ("dsb 0xf":::"memory");
}
```

2.2 System

2.2.1 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.

2.2.2 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 conditions are met:

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

Workaround

Three workarounds are possible:

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

2.2.3 Full JTAG configuration without NJTRST pin cannot be used

Description

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

Workaround

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

2.2.4 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.

2.2.5 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 to the corresponding register just after enabling the peripheral clock.

2.2.6 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

To adapt the accuracy level to the application requirements, set one of the following options:

- Option1

  Set the ADCDC1 bit in the PWR_CR register.
- Option2

  Set the corresponding ADCxDC2 bit in the SYSCFG_PMC register.

Only one option can be set at a time.

For more details on option 1 and option2 mechanisms, refer to AN4073.

2.2.7 Wakeup from Standby mode with RTC

Description

After wakeup from Standby mode with one of the following events

- RTC alarm
- RTC TAMPER
- RTC TimeStamp
- RTC wake up timer

the wakeup flag WUPF in the PWR_CSR register will be kept pending even if the wakeup flag is cleared by setting the CWUPF bit in the PWR_CR register. This prevents the system from entering in Standby mode.

The wakeup from Standby mode with WKUP pin is not impacted.

Workaround

After wakeup from Standby when using RTC wakeup events:

- Check if the SBF flag was set, then clear it by setting the CSBF bit in the PWR_CR register.
- Generate a system reset in order to clear the pending wakeup flag, and allow the system to enter Standby mode next time and to wakeup by RTC.

2.2.8 Data Cache might be corrupted during Flash Read While Write operation

Description

When a write to the internal Flash memory is done, the Data Cache is normally updated to reflect the data value update. During this Data Cache update, a read to the other memory bank may occur; this read can corrupt the Data Cache content, and subsequent read operations at the same address (Cache hits) will be corrupted.

This limitation occurs only in dual bank mode, when reading (data access or code execution) from one Flash Bank while writing to the other Flash Bank with Data Cache enabled.

Workaround

When the application is performing data accesses in both Flash memory banks, the Data Cache must be disabled by resetting the DCEN bit before any write to the Flash. Before enabling the Data Cache again, it must be reset by setting and then resetting the DCRST bit.

Example of code

```
  /* Disable data cache  */
  __HAL_FLASH_DATA_CACHE_DISABLE();

  /* Set PG bit */
  SET_BIT(FLASH->CR, FLASH_CR_PG);

  /* Program the Flash word */
WriteFlash(Address, Data);

  /* Reset data cache */
  __HAL_FLASH_DATA_CACHE_RESET();
  /* Enable data cache */
  __HAL_FLASH_DATA_CACHE_ENABLE();
```

2.2.9 FMC_SDNWEN alternate function on PA7

Description

The FMC_SDNWEN alternate function is not available on PA7.

Workaround

Use FMC_SDNWEN alternate function on PH5 or PC0.

2.3 FMC

2.3.1 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.

2.4 QUADSPI

2.4.1 Extra data written in the FIFO at the end of a read transfer

Description

When all the conditions listed below are gathered:

- QUADSPI is used in indirect mode
- QUADSPI clock is AHB/2 (PRESCALER = 0x01 in the QUADSPI_CR)
- QUADSPI is in quad mode (DMODE = 0b11 in the QUADSPI_CCR)
- QUADSPI is in DDR mode (DDRM = 0b1 in the QUADSPI_CCR)

an extra data is incorrectly written in the FIFO when a data is read at the same time that the FIFO gets full at the end of a read transfer.

Workaround

One of the two workarounds listed below can be used:

- Read out the extra data until the BUSY flag goes low and discard it.
- Request an abort after reading out all the correct received data from FIFO in order to flush FIFO and have the busy low. Abort will keep the last register configuration (set the ABORT bit in the QUADSPI_CR).

2.4.2 First nibble of data is not written after dummy phase

Description

The first nibble of data to be written to the external Flash memory is lost when both following conditions are met:

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

Workaround

Use alternate bytes instead of dummy phases to add latency between address phase and data phase. This works only if the number of dummy cycles corresponds to a multiple of 8 bits of data.

Example: to generate

- one dummy cycle: send one alternate byte, possible only in four data lines DDR mode or Dual-flash SDR mode
- two dummy cycles: send one alternate byte in four data lines SDR mode
- four dummy cycles: send two alternate bytes in four data lines SDR mode, or send one alternate byte in two data lines SDR mode
- eight dummy cycles: send one alternate byte in one data line SDR mode.

2.4.3 Wrong data can be read in memory-mapped after an indirect mode operation

Description

Wrong data can be read with the first memory-mapped read request when QUADSPI entered memory-mapped mode with both LSB bits in the address register QUADSPI_AR[1:0] not reset.

Workaround

QUADSPI_AR register must be reset just before entering memory-mapped mode.

This can be done in two different ways, depending on the current QUADSPI operating mode:

1. Indirect read mode:
   a) Reset address register
   b) Do an abort request to stop reading and clear busy bit
   c) Enter memory-mapped mode.

Note: User should take care not to read QUADSPI_DR register after resetting address register.

2. Indirect write mode: reset the address register then enter to memory-mapped mode

Note: User should take care not to write to QUADSPI_DR register after resetting address register.

2.4.4 Memory-mapped read operations may fail when timeout counter is enabled

Description

In Memory-mapped mode when the TC is enabled, the QuadSPI peripheral can hang and memory-mapped read operations fail.

The QuadSPI hang occurs if the timeout flag TOF is set at the same clock edge of a new memory mapped read request.

Workaround

The timeout counter must be disabled.

In order to rise the chip select high, the application can do an abort at the end of each memory-mapped read operation.

2.5 ADC

2.5.1 ADC sequencer modification during conversion

Description

If an ADC conversion is started by software (writing the SWSTART bit), and if the ADC_SQRx or ADC_JSQRx registers are modified during the conversion, the current conversion is reset and the ADC does not restart a new conversion sequence automatically.

If an ADC conversion is started by hardware trigger, this limitation does not apply. The ADC restarts a new conversion sequence automatically.

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.

2.6 DAC

2.6.1 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.

2.6.2 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.

2.7 DSI Host

2.7.1 When used over the DSI link, the Tearing Effect Interrupt Flag is set when an Acknowledge Trigger is received from the display

Description

In Adapted Command Mode, when the Tearing Effect mechanism is used over the DSI link, the Tearing Effect Interrupt Flag (TEIF) of the DSI Wrapper Interrupt Status Register (DSI_WISR) is asserted when an Acknowledge Trigger is received from the display.

Acknowledges Trigger can be received from the display:

- for each packet when the Acknowledge Request Enable (ARE) bit of the DSI Host Command Mode Configuration Register (DSI_CMCR) is set
- when a response is awaited from the display

Workaround

Do not use the Tearing Effect over the link but use the dedicated TE pin.

When using the Tearing Effect over the link, do not use the Tearing Effect interrupt nor Automatic Refresh Mode, but launch the display refresh immediately after a set_tear_on or a set_scanline DCS command (as the display is driving the DSI link until the Tearing Effect occurs, the refresh will be automatically stalled until the Tearing Effect).

2.7.2 The time to activate the clock between HS transmissions is not calculated correctly

Description

In Automatic Clock Lane control mode, the DSI Host can turn off the clock lane between two High-Speed transmissions.

To do so, the DSI Host calculates the time required for the clock lane to change from High-Speed to Low-Power and from Low-Power to High-Speed.

This timings are configured by the HS2LP_TIME and LP2HS_TIME in the DSI Host Clock Lane Timer Configuration Register (DSI_CLTCR). DSI Host is not calculating LP2HS_TIME + HS2LP_TIME but 2 x HS2LP_TIME instead.

Workaround

Configure HS2LP_TIME and LP2HS_TIME with the same value as the max between HS2LP_TIME and LP2HS_TIME.

As an example, if HS2LP_TIMER = 44 and LP2HS_TIME = 113 configure the register fields as follows:

- HS2LP_TIME = 113
- LP2HS_TIME = 113

2.7.3 The immediate update procedure may fail

Description

The immediate update procedure implies that both the Update Register (UR) and the Enable (EN) bits of the DSI Host Video Shadow Control Register (DSI_VSCR) are initially cleared, and are set by the same instruction.

Because of a race condition between the two signals, this immediate update procedure may fail in few cases, leading the DSI Host to wait until the next frame end before updating the configuration.

Workaround

After an “Immediate update” procedure, verify if the configuration is updated by reading the auto-cleared bit UR.

If the UR bit is not cleared, repeat the process by writing first 0x0000 then 0x0101 in DSIHOST_VSCR.

2.8 IWDG

2.8.1 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.

2.9 RTC

2.9.1 Spurious tamper detection when disabling the tamper channel

Description

If the tamper detection is configured for detection on the falling edge event (TAMPFLT = 00 and TAMPxTRG = 1) and if the tamper event detection is disabled when the tamper pin is at high level, a false tamper event is detected.

Workaround

The false tamper event detection cannot be avoided, but the backup registers erase can be avoided by setting TAMPxNOERASE bit before clearing TAMPxE bit. The two bits must be written in two separate RTC_TAMPCR write accesses.

2.9.2 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.

2.10 I2C

2.10.1 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

2.10.2 Start cannot be generated after a misplaced Stop

Description

If a master generates a misplaced Stop on the bus (bus error), while the peripheral tries to switch to master mode by setting the START bit, the Start condition is not properly generated.

Workaround

In the I²C 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 I²C peripheral.

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

2.10.3 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.

2.10.4 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.

2.10.5 Both SDA and SCL maximum rise time (t_r) violated when VDD_I2C bus is higher than ((V_{DD} + 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 0.3 VDD_I2C and 0.7 VDD_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.

2.10.6 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.

2.11 USART

2.11.1 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.

2.11.2 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.

2.11.3 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.

2.11.4 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.

2.11.5 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.

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

Description

In the ISO7816, when a character parity error is incorrect, the Smartcard receiver shall transmit a NACK error signal at (10.5 ± 0.2) etu after the character START bit falling edge. In this case, the USART transmitter should be able to detect correctly the NACK signal by sampling at (11.0 ± 0.2) etu after the character START bit falling edge.

The USART peripheral used in Smartcard mode doesn't respect the (11 +/-0.2) etu timing, and when the NACK falling edge arrives at 10.68 etu or later, the USART might misinterpret this transition as a START bit even if the NACK is correctly detected.

Workaround

None.

2.11.7 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 in the following conditions: - 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 transfer of D1

Workaround

If the application needs to detect the end of transfer of the data, the break request should be done after making sure that the TC flag is set.

2.11.8 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.

2.11.9 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 the receiver is disabled (RE=0) i.e. not ready to receive data.

Workaround

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

2.12 SPI

2.12.1 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.

2.12.2 The last transacted bit of data or CRC calculation can be corrupted for the received data in master mode depending on the timing of the feedback communication clock respect to the APB clock (SPI or I2S)

Description

When the SPI or I2S is configured in master mode (in full duplex or receiver mode, reading back the data register or CRC enabled), the data received may have the last transacted bit corrupted if delay of internal feedback derived from SCK pin is comparable with APB clock period. The last bit value is strobed too late into the shift register, while its content has been already either copied into the data register or compared with CRC pattern calculated internally. In case of data corruption, the bit position in the data register contains the value of the last bit received during the previous data transfer and CRCERR flag is asserted in spite of all the data being received correctly.

The main factors that contribute negatively to the delay are decreased V_{DD} level, high temperature, high SPI bus capacity load and low SCK IO output speed. SPI communication speed has no impact.

Workaround

- Decrease the APB clock.
- Set the IO pad configuration for the SCK pin to be faster.

Table 4 gives an overview about maximum APB frequency vs. GPIOx_OSPEEDR output speed register setting for the SCK pin with a maximum load 30 pF.

Table 4. Maximum APB frequency vs. GPIOx_OSPEEDR setting

OSPEEDR bits [1:0] for the SCK pin	Maximum APB frequency	
	For SPI	For I2S
High (10) and Very high (11)	90 MHz	45 MHz
Medium (01)	75 MHz	35 MHz
Low (00)	26 MHz	12 MHz

2.12.3 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.

Workaround

Use odd polynomial.

2.12.4 CRC can be corrupted when full duplex is handled by DMA and reception DMA channel is set to number of data frames plus CRC length

Description

When SPI is handled by DMA and configured in full duplex master or slave mode with CRC enabled, the CRC computation can be corrupted when it is frozen temporary under specific conditions for an ongoing frame. This can happen when DMA counters for data reception and transmission are misbalanced and DMA receive counter reaches zero exactly when the transaction is completed. It makes an internal signal normally dedicated to receive only mode not properly cleared and pending. This happens when DMA reception counter includes an additional CRC pattern. Consequently, during the next transaction session, whenever the DMA TXE event service comes too late (due to some other BUS matrix activity e.g. when the DMA is servicing request from another channel) the pending internal signal is wrongly propagated into the unexpected CRC freezing even in full duplex mode.

Workaround

Ensure clearing of the internal CRC freezing signal dedicated to receive only mode before a transaction starts. This can be done by:

- Setting the DMA data counters for reception and transmission equal to assure proper clearing of the signal at the end of every transaction {while CRC pattern is performed). In this case, the received CRC data reading is not handled by DMA and user has to handle reading out this information from data register by software.
- Performing an HW reset of the SPI before a transaction starts via peripheral reset register (if applicable).

2.13 I2S

2.13.1 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.

2.14 SDIO

2.14.1 Wrong CCRCFAIL status after a response without CRC is received

Description

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

2.14.2 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 are such that:

$[3 \times \text{period(PCLK2)} + 3 \times \text{period(SDIOCLK)}] \geq (32 / (\text{BusWidth})) \times \text{period(SDIO\_CK)}$.

Workaround

Avoid the above-mentioned clock frequency relationship, by:

- incrementing the APB frequency;
- or decreasing the transfer bandwidth;
- or reducing SDIO_CK frequency.

2.15 bxCAN

2.15.1 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.

2.16 Ethernet

2.16.1 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).

2.16.2 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.

2.16.3 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.

2.16.4 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.

2.16.5 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 5* for 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

Table 5. 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 5. 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 are 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 completed, and finally reprogram it after a delay of four TX_CLK/RX_CLK clock cycles.

3 Revision history

Table 6. Document revision history

Date	Revision	Changes
27-Aug-2015	1	Initial release.
26-Oct-2015	2	Updated *Table 3: Summary of silicon limitations*. Updated *Section 2.4.1: Extra data written in the FIFO at the end of a read transfer*. Added *Section 2.7: DSI Host* and its subsections.
14-Jan-2016	3	Updated *Table 3: Summary of silicon limitations*. Added *Section 2.2.7: Wakeup from Standby mode with RTC*.
13-Oct-2016	4	Updated *Table 3: Summary of silicon limitations*. Updated *Section 2.2.5: Delay after an RCC peripheral clock enabling*, *Section 2.10.2: Start cannot be generated after a misplaced Stop*, *Section 2.12.1: BSY bit may stay high at the end of a data transfer in slave mode* and *Section 2.12.2: The last transacted bit of data or CRC calculation can be corrupted for the received data in master mode depending on the timing of the feedback communication clock respect to the APB clock (SPI or I2S)*. Added *Section 2.2.8: Data Cache might be corrupted during Flash Read While Write operation*, *Section 2.12.3: Wrong CRC calculation when the polynomial is even*, *Section 2.4.2: First nibble of data is not written after dummy phase* and *Section 2.4.3: Wrong data can be read in memory-mapped after an indirect mode operation*.
28-Jul-2017	5	Replaced former *Silicon identification* with *Applicability*, and former *Table 1: Device identification* with *Table 2: Device variants*. Updated *Section 2: Description of device limitations*. Updated *Table 3: Summary of silicon limitations*. Added *Section 2.2.9: FMC_SDNWEN alternate function on PA7*.
06-Jun-2018	6	Updated *Applicability*. Updated *Table 1: Device summary*, *Table 2: Device variants* and *Table 3: Summary of silicon limitations*. Rearranged description of limitations according to order of RM0386. Replaced former *Section 1: ARM® 32-bit Cortex®-M4 with FPU limitations* with *Section 2.1: Core* and its subsections. Added *Section 2.1.3: Store immediate overlapping exception return operation might vector to incorrect interrupt*, *Section 2.12.4: CRC can be corrupted when full duplex is handled by DMA and reception DMA channel is set to number of data frames plus CRC length* and *Section 2.4.4: Memory-mapped read operations may fail when timeout counter is enabled*.