# STM32MP151x STM32MP153x STM32MP157x

Errata sheet

## STM32MP151x/3x/7x device errata

## Applicability

This document applies to the part numbers of STM32MP151x/3x/7x devices and the device variants as stated in this page.

It gives a summary and a description of the device errata, with respect to the device datasheet and reference manual RM0441 for STM32MP151x, RM0442 for STM32MP153x, and RM0436 for STM32MP157x.

Deviation of the real device behavior from the intended device behavior is considered to be a device limitation. Deviation of the description in the reference manual or the datasheet from the intended device behavior is considered to be a documentation erratum. The term *"errata"* applies both to limitations and documentation errata.

Table 1. **Device summary**

| Reference | Part numbers |
|---|---|
| STM32MP151x | STM32MP151A, STM32MP151C |
| STM32MP153x | STM32MP153A, STM32MP153C |
| STM32MP157x | STM32MP157A, STM32MP157C |

Table 2. **Device variants**

| Reference | Silicon revision codes | |
|---|---|---|
| | Device marking(1) | REV_ID(2) |
| STM32MP151x/3x/7x | B | 0x2000 |

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

# 1 Summary of device errata

The following table gives a quick reference to the STM32MP151x/3x/7x device limitations and their status:

A = workaround available

N = no workaround available

P = partial workaround available

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

**Table 3. Summary of device limitations**

| Function | Section | Limitation | Status |
|---|---|---|---|
| | | | Rev. B |
| Arm Cortex-A7 core | 2.1.1 | Memory locations might be accessed speculatively due to instruction fetches when HCR.VM is set | A |
| | 2.1.2 | Cache maintenance by set/way operations can execute out of order | A |
| | 2.1.3 | PMU events 0x07, 0x0C, and 0x0E do not increment correctly | N |
| | 2.1.4 | PMU event counter 0x14 does not increment correctly | A |
| | 2.1.5 | Exception mask bits are cleared when an exception is taken in Hyp mode | N |
| Arm Cortex-M4 core | 2.2.1 | Interrupted loads to SP can cause erroneous behavior | A |
| | 2.2.2 | VDIV or VSQRT instructions might not complete correctly when very short ISRs are used | A |
| | 2.2.3 | Store immediate overlapping exception return operation might vector to incorrect interrupt | A |
| System | 2.3.1 | TPIU fails to output sync after the pattern generator is disabled in Normal mode | A |
| | 2.3.2 | Serial-wire multi-drop debug not supported | N |
| | 2.3.3 | HSE external oscillator required in some LTDC use cases | P |
| | 2.3.4 | RCC cannot exit Stop and LP-Stop modes | A |
| | 2.3.5 | Incorrect reset of glitch-free kernel clock switch | P |
| | 2.3.6 | Limitation of aclk/hclk5/hclk6 to 200 MHz when used as SDMMC1/2 kernel clock | P |
| DDRPHYC | 2.4.1 | DDRPHYC overconsumption upon reset or Standby mode exit | A |
| DMAMUX | 2.5.1 | SOFx not asserted when writing into DMAMUX_CFR register | N |
| | 2.5.2 | OFx not asserted for trigger event coinciding with last DMAMUX request | N |
| | 2.5.3 | OFx not asserted when writing into DMAMUX_RGCFR register | N |
| | 2.5.4 | Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event | A |
| QUADSPI | 2.6.1 | Memory-mapped read of last memory byte fails | P |
| ADC | 2.7.1 | ADC ANA0/ANA1 resolution limited when Gigabit Ethernet is used | P |
| | 2.7.2 | ADC missing codes in differential 16-bit static acquisition | P |
| DTS | 2.8.1 | Mode using PCLK & LSE (REFCLK_SEL = 1) should not be used | P |
| DSI | 2.9.1 | DSI PHY compliant with MIPI DPHY v0.81 specification, not with DSI PHY v1.0 | N |

| Function | Section | Limitation | Status |
|---|---|---|---|
| | | | Rev. B |
| TIM | 2.10.1 | One-pulse mode trigger not detected in master-slave reset + trigger configuration | P |
| LPTIM | 2.11.1 | MCU may remain stuck in LPTIM interrupt when entering Stop mode | A |
| | 2.11.2 | MCU may remain stuck in LPTIM interrupt when clearing event flag | P |
| RTC and TAMP | 2.12.1 | Incorrect version register | N |
| | 2.12.2 | Calendar initialization may fail in case of consecutive INIT mode entry | A |
| | 2.12.3 | Alarm flag may be repeatedly set when the core is stopped in debug | N |
| I2C | 2.13.1 | Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period | P |
| | 2.13.2 | Spurious bus error detection in master mode | A |
| | 2.13.3 | Spurious master transfer upon own slave address match | P |
| SPI | 2.14.1 | Master data transfer stall at system clock much faster than SCK | A |
| | 2.14.2 | Corrupted CRC return at non-zero UDRDET setting | P |
| | 2.14.3 | TXP interrupt occurring while SPI disabled | A |
| ETH | 2.15.1 | Incorrect L4 inverse filtering results for corrupted packets | N |
| | 2.15.2 | Rx DMA may fail to recover upon DMA restart following a bus error, with Rx timestamping enabled | A |
| | 2.15.3 | Tx DMA may halt while fetching TSO header under specific conditions | A |
| | 2.15.4 | Spurious receive watchdog timeout interrupt | A |
| | 2.15.5 | Incorrect flexible PPS output interval under specific conditions | A |
| | 2.15.6 | Packets dropped in RMII 10Mbps mode due to fake dribble and CRC error | A |
| | 2.15.7 | ARP offload function not effective | A |

# 2 Description of device errata

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

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

## 2.1 Arm Cortex-A7 core

Errata notice for the Arm® Cortex®-A7 core revision r0p5-REVIDR=0x01 is available from http://infocenter.arm.com.

### 2.1.1 Memory locations might be accessed speculatively due to instruction fetches when HCR.VM is set

This limitation is registered under Arm ID number 844169 and classified into “Category B”. Its impact to the device is minor.

#### Description

The ARMv7 architecture requires that when all associated stages of translation are disabled for the current privilege level, memory locations are only accessed due to instruction fetches within the same or next 4 KB region as an instruction which has been or will be fetched due to sequential execution. In the conditions detailed below, the Cortex-A7 processor might access other locations speculatively due to instruction fetches.

The unwanted speculative instruction fetches may occur when the following conditions are all met:

1. The processor must be executing at PL2 or Secure PL1.
2. Address translation is disabled for the current exception level (by clearing the appropriate SCTLR.M or HSCTLR.M bit).
3. The HCR.VM bit is set.

As the HCR.VM is reset low, this issue will not manifest during boot code.

#### Workaround

This issue is most likely to arise in powerdown code, if PL2 or secure PL1 software disables address translation before the core is powered down.

To work around this erratum, software should ensure that HCR.VM is cleared before disabling address translation at PL2 or Secure PL1.

### 2.1.2 Cache maintenance by set/way operations can execute out of order

This limitation is registered under Arm ID number 814220 and classified into “Category B”. Its impact to the device is limited.

#### Description

The ARM v7 architecture states that all cache and branch predictor maintenance operations that do not specify an address execute in program order, relative to each other. However, because of this erratum, an L2 set/way cache maintenance operation can overtake an L1 set/way cache maintenance operation.

Code that intends to clean dirty data from L1 to L2 and then from L2 to L3 using set/way operations might not behave as expected. The L2 to L3 operation might happen first and result in dirty data remaining in L2 after the L1 to L2 operation has completed.

If dirty data remains in L2 then an external agent, such as a DMA agent, might observe stale data.

If the processor is reset or powered-down while dirty data remains in L2 then the dirty data will be lost.

The failure occurs when the following conditions are all met:

1. A CPU performs an L1 DCCSW or DCCISW operation.

2. The targeted L1 set/way contains dirty data.
3. Before the next DSB, the same CPU executes an L2 DCCSW or DCCISW operation while the L1 set/way operation is in progress.
4. The targeted L2 set/way is within the group of L2 set/ways that the dirty data from L1 can be allocated to.

If the above conditions are met then the L2 set/way operation can take effect before the dirty data from L1 has been written to L2.

*Note: Conditions (3) and (4) are not likely to be met concurrently when performing set/way operations on the entire L1 and L2 caches. This is because cache maintenance code is likely to iterate through sets and ways in a consistent ascending or consistent descending manner across cache levels, and to perform all operations on one cache level before moving on to the next cache level. This means that, for example, cache maintenance operations on L1 set 0 and L2 set 0 will be separated by cache maintenance operations for all other sets in the L1 cache. This creates a large window for the cache maintenance operations on L1 set 0 to complete.*

#### Workaround

Correct ordering between set/way cache maintenance operations can be forced by executing a DSB before changing cache levels.

### 2.1.3 PMU events 0x07, 0x0C, and 0x0E do not increment correctly

This limitation is registered under Arm ID number 809719 and classified into “Category C”. Its impact to the device is minor.

#### Description

The Cortex-A7 processor implements version 2 of the Performance Monitor Unit architecture (PMUv2). The PMU can gather statistics on the operation of the processor and memory system during runtime. This event information can be used when debugging or profiling code.

The PMU can be programmed to count architecturally executed stores (event 0x07), software changes of the PC (event 0x0C), and procedure returns (event 0x0E). However, because of this erratum, these events do not fully adhere to the descriptions in the PMUv2 architecture.

As a result of this limitation, the information returned by PMU counters that are programmed to count events 0x07, 0x0C, or 0x0E might be misleading when debugging or profiling code executed on the processor.

The error occurs when the following conditions are met:

Either:

1. A PMU counter is enabled and programmed to count event 0x07. That is: instruction architecturally executed, condition code check pass, store.
2. A PLDW instruction is executed.

If the above conditions are met, the PMUv2 architecture specifies that the counter for event 0x07 does not increment. However, the counter does increment.

Or:

1. A PMU counter is enabled and programmed to count event 0x0C. That is: instruction architecturally executed, condition code check pass, software change of the PC.
2. An SVC, HVC, or SMC instruction is executed.

If the above conditions are met, the PMUv2 architecture specifies that the counter for event 0x0C increments. However, the counter does not increment.

Or:

1. A PMU counter is enabled and programmed to count event 0x0E. That is: instruction architecturally executed, condition code check pass, procedure return.
2. One of the following instructions is executed:
   a. `MOV PC, LR`
   b. `ThumbEE LDMIA R9!, {?, PC}`
   c. `ThumbEE LDR PC, [R9], #offset`
   d. `BX Rm, where Rm != R14`
   e. `LDM SP, {?, PC}`

If the above conditions are met, the PMUv2 architecture specifies that the counter for event 0x0E increments for (a), (b), (c) and does not increment for (d) and (e). However, the counter does not increment for (a), (b), (c) and increments for (d) and (e).

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, then this interrupt will eventually become re-pending and subsequently be handled.

#### Workaround

None.

### 2.1.4 PMU event counter 0x14 does not increment correctly

This limitation is registered under Arm ID number 805420 and classified into “Category C”. Its impact to the device is minor.

#### Description

The Cortex-A7 MPCore processor implements version 2 of the Performance Monitor Unit architecture (PMUv2). The PMU can gather statistics on the operation of the processor and memory system during runtime. This event information can be used when debugging or profiling code. When a PMU counter is programmed to count L1 instruction cache accesses (event 0x14), the counter should increment on all L1 instruction cache accesses.

This limitation has the following implications:

1. If SCR.{AW, FW} is set to 0 then the clearing of corresponding bit CPSR.{A, F} to 0 has no effect. The value of CPSR.{A, F} is ignored.
2. A PMU counter is enabled and programmed to count L1 instruction cache accesses (event 0x14).
3. Cacheable instruction fetches miss in the L1 instruction cache.

If the above conditions are met, the event counter will not increment.

#### Workaround

To obtain a better approximation for the number of L1 instruction cache accesses, enable a second PMU counter and program it to count instruction fetches that cause linefills (event 0x01). Add the value returned by this counter to the value returned by the L1 instruction access counter (event 0x14). The result of the addition is a better indication of the number of L1 instruction cache accesses.

### 2.1.5 Exception mask bits are cleared when an exception is taken in Hyp mode

This limitation is registered under Arm ID number 804069 and classified into “Category C”. Its impact to the device is minor.

#### Description

The Cortex-A7 processor implements the ARM Virtualization Extensions and the ARM Security Extensions. Exceptions can be routed to Monitor mode by setting SCR.{EA, FIQ, IRQ} to 1. Exceptions can be masked by setting corresponding bit CPSR.{A, I, F} to 1.

The ARMv7-A architecture states that an exception taken in Hyp mode does not change the value of the mask bits for exceptions routed to Monitor mode. However, because of this erratum, the corresponding mask bits will be cleared to 0.

The error occurs when the following conditions are met:

1. One or more exception types are routed to Monitor mode by setting one or more of SCR.{EA, FIQ, IRQ} to 1.
2. The corresponding exception types are masked by setting the corresponding CPSR.{A, F, I} bits to 1.
3. Any exception is taken in Hyp mode.

If the above conditions are met then the exception mask bit CPSR.{A, F, I} is cleared to 0 for each exception type that meets conditions (1) and (2). The affected mask bits are cleared together regardless of the exception type in condition (3).

The implications of this erratum are:

- If SCR.{AW, FW} is set to 0 then the clearing of corresponding bit CPSR.{A, F} to 0 has no effect. The value of CPSR.{A, F} is ignored.
- Otherwise, when CPSR.{A, F, I} is set to 1, Secure code cannot rely on CPSR.{A, F, I} remaining set to 1. An exception that should be masked might be routed to Monitor mode.

The impact of this limitation is considered to be minor as it is expected that the users will:

1. set SCR.{AW, FW} to 0 when SCR.{EA, FIQ} is set to 1.
2. set SCR.IRQ to 0.

#### Workaround

None.

## 2.2 Arm Cortex-M4 core

Errata notice for the Arm® Cortex®-M4 core revision r0p1 is available from http://infocenter.arm.com.

### 2.2.1 Interrupted loads to SP 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

If an interrupt occurs during the data-phase of a single word load to the stack-pointer (SP/R13), erroneous behavior can occur. In all cases, returning from the interrupt will result in the load instruction being executed an additional time. For all instructions performing an update to the base register, the base register will be erroneously updated on each execution, resulting in the stack-pointer being loaded from an incorrect memory location.

The affected instructions that can result in the load transaction being repeated are:

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

The affected instructions that can result in the stack-pointer being loaded from an incorrect memory address are:

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

As compilers do not generate these particular instructions, the limitation is only likely to occur with hand-written assembly code.

#### Workaround

Both issues may be worked around by replacing the direct load to the stack-pointer, with an intermediate load to a general-purpose register followed by a move to the stack-pointer.

### 2.2.2 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 the FPSCR.

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 2 cycles for instructions inside the interrupt service routine, or 2 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:

- Disable lazy context save of floating point state by clearing LSPEN to 0 (bit 30 of the FPCCR at address 0xE000EF34).
- Disable lazy context save of floating point state by clearing LSPEN to 0 (bit 30 of the FPCCR at address 0xE000EF34).

### 2.2.3 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");
}
```

## 2.3 System

### 2.3.1 TPIU fails to output sync after the pattern generator is disabled in Normal mode

#### Description

The TPIU includes a pattern generator that can be used by external tool to determine the operating behavior of the trace port and timing characteristics. This pattern generator includes a mode that transmits the test pattern for a specified number of cycles, and then reverts to transmitting normal trace data.

When the TPIU is configured to operate in Normal mode (FFCR.EnFCont=0), the synchronization sequence that is required between the test pattern and the trace data is not generated. Synchronization will be generated at a later time, as determined by the synchronization counter.

Conditions:

The following conditions must all occur:

- The TPIU is configured in normal mode, `FFCR.EnFCont==0`
- The TPIU is configured with the formatter enabled, `FFCR.EnFTC==1`
- The pattern generator is enabled in timed mode, `Current_test_pattern_mode.PTIMEEN==1`

Implications:

The timed mode of the TPIU is intended to permit the TPIU to transition between an initial synchronization sequence using the pattern generator and functional mode without any further programming intervention. If the synchronization sequence is not generated at the end of the test pattern, the trace port analyzer is unlikely to be able to capture the start of the trace stream correctly. Synchronization will be correctly inserted based on the value configured in the FSCR, once the specified number of frames of trace data have been output.

#### Workaround

Workaround requires software interaction to detect the completion of the test pattern sequence. In addition, any trace data present at the input to the TPIU is lost whilst the pattern generator is active. Any trace data present in the input to the TPIU before the formatter is re-enabled (and synchronization generated) will not be de-compressible.

1. After enabling the pattern generator, set `FFCR.StopOnFl==1` and `FFCR.FOnMan==1`.
2. Poll `FFSR.FtStopped` until 1 is read
3. Set `FFCR.EnFTC==1`

### 2.3.2 Serial-wire multi-drop debug not supported

#### Description

The target instance bitfield TINSTANCE[3:0] of the DP_DLPIDR debug port data link protocol identification register is frozen to 0, which prevents the debug of multiple instances of the same function connected on same SWD bus.

#### Workaround

None.

### 2.3.3 HSE external oscillator required in some LTDC use cases

#### Description

Due to capacitive load on the LTDC pins, the LTDC used at high or very high speed (OSPEEDR ≥ 1) may impact the on-chip HSE crystal oscillator clock, which then could lead to a deterioration of USB HS eye diagram.

*Note: This does not apply when the LTDC is internally connected to the DSI host, even at high clock frequencies up to 90 MHz, as it does not drive pins.*

#### Workaround

If using the USB HS interface, avoid using the on-chip HSE oscillator in the use cases summarized in the table. Instead, get the clock from an external oscillator connected to the HSE pins, as indicated in the table (mandatory or recommended).

Table 4. **Use of external oscillator on HSE pins**

| LTDC connected to | Conditions | | External oscillator on HSE |
|---|---|---|---|
| HDMI bridge | HDMI bridge close to the device<br>load ~15 pf | 3 V < $V_{DD}$< 3.6 V<br>$f_{pixel}$ up to 74.25 MHz (1280x720 at 60 fps)<br>OSPEEDR = 1 for all LTDC signals<br>USB HS used | Mandatory |
| | | 1.7 V < $V_{DD}$< 2 V<br>$f_{pixel}$ up to 74.25 MHz<br>Duty cycle = 40 %<br>0.1 $V_{DD}$ < $V_{IN}$ < 0.9 $V_{DD}$<br>OSPEEDR = 3 for LCD_CLK<br>OSPEEDR = 2 for all other LTDC signals | Recommended |

| LTDC connected to | Conditions | | External oscillator on HSE |
|---|---|---|---|
| Parallel LCD | load < 30 pf | 3 V < $V_{DD}$< 3.6 V<br>$f_{pixel}$ up to 90 MHz (1366x768 at 60 fps)<br>OSPEEDR = 2 for LCD_CLK<br>OSPEEDR = 1 for all other LTDC signals<br>USB HS used | Mandatory |
| | | 3 V < $V_{DD}$< 3.6 V<br>$f_{pixel}$ up to 48 MHz (800x600 at 60 fps)<br>OSPEEDR = 1 for LCD_CLK<br>OSPEEDR = 0 for all other LTDC signals | Recommended |
| | | 1.7 V < $V_{DD}$< 2 V<br>$f_{pixel}$ up to 69 MHz (1024x768 at 60 fps)<br>Duty cycle = 40 %<br>0.1 $V_{DD}$ < $V_{IN}$ < 0.9 $V_{DD}$<br>OSPEEDR = 3 for LCD_CLK<br>OSPEEDR = 2 for all other LTDC signals | |

### 2.3.4 RCC cannot exit Stop and LP-Stop modes

#### Description

When trying to exit Stop or LP-Stop mode, the handshake mechanism between PWRCTRL and RCC can be wrong due to a too short internal state within the RCC state machine. In extreme case, the PWRCTRL does not see the RCC signal, thus causing the whole system going to Stop or LP-Stop mode again, respectively.

#### Workaround

Set the PWRLP_DLY[21:0] bitfield of the RCC_PWRLPDLYCR register to a value equal to or greater than 0x4.

*Note: This register is designed to handle LP-Stop mode, but it can also be used in the present case for Stop mode.*

### 2.3.5 Incorrect reset of glitch-free kernel clock switch

#### Description

The activation of NRST (by external signal, internal watchdog or software) may not properly reset the glitch-free clock switches inside the RCC.
As a consequence, the peripheral kernel clock previously used by the application remains selected. If not enabled again (by default or by SW), there is no kernel clock present on the related peripheral and the BootROM hangs.

*Note: USB boot is usually not affected as the application always uses the same clocking scheme depending on HSE crystal frequency choice. For example, there is no issue for HSE = 24 MHz as hse_ker_ck is used for OTG clocking. Other peripherals, such as LPTIM, USART, and I2C, should work without care if the previous application clock is enabled again to ensure that the clock switch is not stuck.*

#### Workaround

Use one of the following measures:

1. By hardware, ensure that the VDDCORE logic is reset on NRST activation:
   – With STPMIC1
     By default, a power cycle on VDDCORE (and VDD) is issued upon activating NRST.
   – Without STPMIC1
     Connect NRST to NRST_CORE or ensure that the NRST activation causes a VDDCORE power cycle.

The drawback is that, the debug logic also being reset, it is not possible to debug the boot chain (TF-A and U-Boot) as any breakpoints set are lost during the power cycle or NRST_CORE activation.

2. By software:
   – For interfaces required during boot, whether Flash memory peripherals (SDMMC1/2, QUADSPI, or FMC) or USART/UART (USART2/3/6 or UART4/5/7/8), use the same clock as the one used during the BootROM execution:
     ◦ If BOOT[2:0] = 000 or 110 (UART boot), set UARTxxSRC[2:0] to 0 (pclk) or 2 (hsi_ker_ck), for all UART instances not disabled via uart_intances_disabled bitfield of the BSEC OTP WORD 3.
     ◦ If BOOT[2:0] = 001 or 111 (QUADSPI boot), set QSPISRC[2:0] to 0 (aclk) or 3 (per_ck).
     ◦ If BOOT[2:0] = 010 or 101 (SDMMC boot), set SDMMC12SRC[2:0] to 0 (hclk6) or 3 (hsi_ker_ck).
   – For SD card, the use of HSI (64 MHz) causes raw bandwidth performance penalty as only 32 or 64 MHz could be used instead of respectively 50 MHz (SDR25/DDR50) and 100 MHz (SDR50)
   – For eMMC, the use of HSI (64 MHz) causes raw bandwidth performance penalty as only 32 or 64 MHz could be used instead of respectively 52 MHz (SDR/DDR) or over 100 MHz (HS200). Note that hclk6/hclk5/aclk are limited to 200 MHz when hclk6 is used as SDMMC1/SDMMC2 kernel clock
     ◦ If BOOT[2:0] = 011 (FMC boot), set FMCSRC[2:0] to 0 (aclk) or 3 (per_ck)

### 2.3.6 Limitation of aclk/hclk5/hclk6 to 200 MHz when used as SDMMC1/2 kernel clock

#### Description

The SDMMC1 and SDMMC2 kernel clock is limited to 200 MHz whereas hclk6 maximum frequency is 266 MHz. As a consequence, when SDMMC12SRC[2:0] = 0 (hclk6), the AXI bus clock (aclk), AHB5 bus clock (hclk5), and AHB6 bus clock (hclk6) cannot exceed 200 MHz.

#### Workaround

Apply one of the following measures:
- When SDMMC12SRC[2:0] = 0, limit aclk/hclk5/hclk6 to 200 MHz.
- Use another SDMMC1/SDMMC2 kernel clock source, that is, set SDMMC12SRC[2:0] to 1, 2, or 3.

## 2.4 DDRPHYC

### 2.4.1 DDRPHYC overconsumption upon reset or Standby mode exit

#### Description

Upon reset and upon Standby mode exit, DDRPHY DLLs are not properly disabled, which leads to leakage on the DDR_ZQ pin causing excessive consumption from $V_{DDCORE}$. Once DDRPHYC is initialized for normal DDR usage, the consumption becomes as specified and remains as specified during Stop modes.

There is no DDRPHYC overconsumption in Standby mode.

#### Workaround

Prevent the leakage on the DDR_ZQ pin, by first enabling DDRPHYC in the RCC, then setting the ZQPD bit of the DDRPHYC_ZQ0CR0 register.

## 2.5 DMAMUX

### 2.5.1 SOFx not asserted when writing into DMAMUX_CFR register

#### Description

The SOFx flag of the DMAMUX_CSR status register is not asserted if overrun from another DMAMUX channel occurs when the software writes into the DMAMUX_CFR register.

This can happen when multiple DMA channels operate in synchronization mode, and when overrun can occur from more than one channel. As the SOFx flag clear requires a write into the DMAMUX_CFR register (to set the corresponding CSOFx bit), overrun occurring from another DMAMUX channel operating during that write operation fails to raise its corresponding SOFx flag.

#### Workaround

None. Avoid the use of synchronization mode for concurrent DMAMUX channels, if at least two of them potentially generate synchronization overrun.

### 2.5.2 OFx not asserted for trigger event coinciding with last DMAMUX request

#### Description

In the DMAMUX request generator, a trigger event detected in a critical instant of the last-generated DMAMUX request being served by the DMA controller does not assert the corresponding trigger overrun flag OFx. The critical instant is the clock cycle at the very end of the trigger overrun condition.

Additionally, upon the following trigger event, one single DMA request is issued by the DMAMUX request generator, regardless of the programmed number of DMA requests to generate.

The failure only occurs if the number of requests to generate is set to more than two (GNBREQ[4:0] > 00001).

#### Workaround

Make the trigger period longer than the duration required for serving the programmed number of DMA requests, so as to avoid the trigger overrun condition from occurring on the very last DMA data transfer.

### 2.5.3 OFx not asserted when writing into DMAMUX_RGCFR register

#### Description

The OFx flag of the DMAMUX_RGSR status register is not asserted if an overrun from another DMAMUX request generator channel occurs when the software writes into the DMAMUX_RGCFR register. This can happen when multiple DMA channels operate with the DMAMUX request generator, and when an overrun can occur from more than one request generator channel. As the OFx flag clear requires a write into the DMAMUX_RGCFR register (to set the corresponding COFx bit), an overrun occurring in another DMAMUX channel operating with another request generator channel during that write operation fails to raise the corresponding OFx flag.

#### Workaround

None. Avoid the use of request generator mode for concurrent DMAMUX channels, if at least two channels are potentially generating a request generator overrun.

### 2.5.4 Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event

#### Description

If a write access into the DMAMUX_CxCR register having the SE bit at zero and SPOL[1:0] bitfield at a value other than 00:

- sets the SE bit (enables synchronization),
- modifies the values of the DMAREQ_ID[5:0] and SYNC_ID[4:0] bitfields, and
- does not modify the SPOL[1:0] bitfield,

and if a synchronization event occurs on the previously selected synchronization input exactly two AHB clock cycles before this DMAMUX_CxCR write, then the input DMA request selected by the DMAREQ_ID[5:0] value before that write is routed.

#### Workaround

Ensure that the SPOL[1:0] bitfield is at 00 whenever the SE bit is 0. When enabling synchronization by setting the SE bit, always set the SPOL[1:0] bitfield to a value other than 00 with the same write operation into the DMAMUX_CxCR register.

## 2.6 QUADSPI

### 2.6.1 Memory-mapped read of last memory byte fails

#### Description

Regardless of the number of I/O lines used (1, 2 or 4), a memory-mapped read of the last byte of the memory region defined through the FSIZE[4:0] bitfield of the QUADSPI_DCR register always yields 0x00, whatever the memory byte content is. A repeated attempt to read that last byte causes the AXI bus to stall.

#### Workaround

Apply one of the following measures:

- Avoid reading the last byte of the memory region defined through FSIZE, for example by taking margin in FSIZE bitfield setting.
- If the last byte is read, ignore its value and abort the ongoing process so as to prevent the AXI bus from stalling.
- For reading the last byte of the memory region defined through FSIZE, use indirect read.

## 2.7 ADC

### 2.7.1 ADC ANA0/ANA1 resolution limited when Gigabit Ethernet is used

#### Description

The following ADC1 and ADC2 input pins might be impacted by adjacent PA0 and PG13 activity when used for ETH in Gigabit mode.

#### Workaround

16-bit and 14-bit data resolutions are not recommended on these pins when Gigabit Ethernet is used. This limits data resolution configuration to 8 bits, 10 bits or 12 bits.

### 2.7.2 ADC missing codes in differential 16-bit static acquisition

In differential mode, static signal acquisition shows missing codes in 16-bit mode. Dynamic signal acquisition is not affected as the energy of missing codes is negligible.

#### Workaround

In applications where missing codes matter, avoid using 16-bit data resolutions on static acquisition.

## 2.8 DTS

### 2.8.1 Mode using PCLK & LSE (REFCLK_SEL = 1) should not be used

#### Description

The DTS cannot be used with PCLK enabled while REFCLK is set to LSE (REFCLK_SEL = 1).

#### Workaround

Only use mode with REFCLK = PCLK (REFCLK_SEL = 0).

## 2.9 DSI

### 2.9.1 DSI PHY compliant with MIPI DPHY v0.81 specification, not with DSI PHY v1.0

#### Description

The DSI PHY does not comply with the DSI PHY v1.0 specification. Instead, it operates in compliance with the MIPI DPHY v0.81 specification.

#### Workaround

None.

## 2.10 TIM

### 2.10.1 One-pulse mode trigger not detected in master-slave reset + trigger configuration

#### Description

The failure occurs when several timers configured in one-pulse mode are cascaded, and the master timer is configured in combined reset + trigger mode with the MSM bit set:

OPM=1 in TIMx_CR1, SMS[3:0]=1000 and MSM=1 in TIMx_SMCR.

The MSM delays the reaction of the master timer to the trigger event, so as to have the slave timers cycle-accurately synchronized.

If the trigger arrives when the counter value is equal to the period value set in the TIMx_ARR register, the one-pulse mode of the master timer does not work and no pulse is generated on the output.

#### Workaround

None. However, unless a cycle-level synchronization is mandatory, it is advised to keep the MSM bit reset, in which case the problem is not present. The MSM=0 configuration also allows to decrease the timer latency to external trigger events.

## 2.11 LPTIM

### 2.11.1 MCU may remain stuck in LPTIM interrupt when entering Stop mode

#### Description

This limitation occurs when disabling the low-power timer (LPTIM).

When the user application clears the ENABLE bit in the LPTIM_CR register within a small time window around one LPTIM interrupt occurrence, then the LPTIM interrupt signal used to wake up the MCU from Stop mode may be frozen in active state. Consequently, when trying to enter Stop mode, this limitation prevents the MCU from entering low-power mode and the firmware remains stuck in the LPTIM interrupt routine.

This limitation applies to all Stop modes and to all instances of the LPTIM. Note that the occurrence of this issue is very low.

#### Workaround

In order to disable a low power timer (LPTIMx) peripheral, do not clear its ENABLE bit in its respective LPTIM_CR register. Instead, reset the whole LPTIMx peripheral via the RCC controller by setting and resetting its respective LPTIMxRST bit in RCC_APByRSTRz register.

### 2.11.2 MCU may remain stuck in LPTIM interrupt when clearing event flag

#### Description

This limitation occurs when the LPTIM is configured in interrupt mode (at least one interrupt is enabled) and the software clears any flag by writing the LPTIM_ICR bit in the LPTIM_ISR register. If the interrupt status flag corresponding to a disabled interrupt is cleared simultaneously with a new event detection, the set and clear commands might reach the APB domain at the same time, leading to an asynchronous interrupt signal permanently stuck at ‘1’.

This issue can occur either during an interrupt subroutine execution (where the flag clearing is usually done), or outside an interrupt subroutine.

Consequently, the firmware remains stuck in the LPTIM interrupt routine, and the MCU cannot enter Stop mode.

#### Workaround

To avoid this issue, it is strongly advised to follow the recommendations listed below:

- Clear the flag only when its corresponding interrupt is enabled in the interrupt enable register.
- If for specific reasons, it is required to clear some flags that have corresponding interrupt lines disabled in the interrupt enable register, it is recommended to clear them during the current subroutine prior to those which have corresponding interrupt line enabled in the interrupt enable register.
- Flags must not be cleared outside the interrupt subroutine.

*Note: The proper clear sequence is already implemented in the HAL_LPTIM_IRQHandler in the STM32Cube™.*

## 2.12 RTC and TAMP

### 2.12.1 Incorrect version register

#### Description

The RTC_VERR and the TAMP_VERR registers do not provide the correct IP revision information.

#### Workaround

None.

### 2.12.2 Calendar initialization may fail in case of consecutive INIT mode entry

#### Description

If the INIT bit of the RTC_ICSR register is set between one and two RTCCLK cycles after being cleared, the INITF flag is set immediately instead of waiting for synchronization delay (which should be between one and two RTCCLK cycles), and the initialization of registers may fail. Depending on the INIT bit clearing and setting instants versus the RTCCLK edges, it can happen that, after being immediately set, the INITF flag is cleared during one RTCCLK period then set again. As writes to calendar registers are ignored when INITF is low, a write occurring during this critical period might result in the corruption of one or more calendar registers.

#### Workaround

After existing the initialization mode, clear the BYPSHAD bit (if set) then wait for RSF to rise, before entering the initialization mode again.

*Note: It is recommended to write all registers in a single initialization session to avoid accumulating synchronization delays.*

### 2.12.3 Alarm flag may be repeatedly set when the core is stopped in debug

#### Description

When the core is stopped in debug mode, the clock is supplied to subsecond RTC alarm downcounter even though the device is configured to stop the RTC in debug.

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

#### Workaround

None.

## 2.13 I2C

### 2.13.1 Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period

**Description**

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

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

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

**Workaround**

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

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

### 2.13.2 Spurious bus error detection in master mode

**Description**

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

**Workaround**

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

### 2.13.3 Spurious master transfer upon own slave address match

**Description**

When the device is configured to operate at the same time as master and slave (in a multi- master I$^2$C-bus application), a spurious master transfer may occur under the following condition:

- Another master on the bus is in process of sending the slave address of the device (the bus is busy).
- The device initiates a master transfer by bit set before the slave address match event (the ADDR flag set in the I2C_ISR register) occurs.
- After the ADDR flag is set:
  - the device does not write I2C_CR2 before clearing the ADDR flag, or
  - the device writes I2C_CR2 earlier than three I2C kernel clock cycles before clearing the ADDR flag

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

#### Workaround

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

Normal mode (SBC = 0):

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

Slave byte control mode (SBC = 1):

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

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

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

## 2.14 SPI

### 2.14.1 Master data transfer stall at system clock much faster than SCK

#### Description

With the system clock (spi_pclk) substantially faster than SCK (spi_ker_ck divided by a prescaler), SPI master data transfer can stall upon setting the CSTART bit within one SCK cycle after the EOT event (EOT flag raise) signaling the end of the previous transfer.

#### Workaround

Apply one of the following measures:

- Disable then enable SPI after each EOT event.
- Upon EOT event, wait for at least one SCK cycle before setting CSTART.
- Prevent EOT events from occurring, by setting transfer size to undefined (TSIZE = 0) and by triggering transmission exclusively by TXFIFO writes.

### 2.14.2 Corrupted CRC return at non-zero UDRDET setting

#### Description

With non-zero setting of UDRDET[1:0] bitfield, the SPI slave can transmit the first bit of CRC pattern corrupted, coming wrongly from the UDRCFG register instead of SPI_TXCRC. All other CRC bits come from the SPI_TXCRC register, as expected.

#### Workaround

Keep TXFIFO non-empty at the end of transfer.

### 2.14.3 TXP interrupt occurring while SPI disabled

#### Description

SPI peripheral is set to its default state when disabled (SPE = 0). This flushes the FIFO buffers and resets their occupancy flags. TXP and TXC flags become set (the latter if the TSIZE field contains zero value), triggering interrupt if enabled with TXPIE or EOTIE bit, respectively. The resulting interrupt service can be spurious if it tries to write data into TXFIFO to clear the TXP and TXC flags, while both FIFO buffers are inaccessible (as the peripheral is disabled).

#### Workaround

Keep TXP and TXC (the latter if the TSIZE field contains zero value) interrupt disabled whenever the SPI peripheral is disabled.

## 2.15 ETH

### 2.15.1 Incorrect L4 inverse filtering results for corrupted packets

#### Description

Received corrupted IP packets with payload (for IPv4) or total (IPv6) length of less than two bytes for L4 source port (SP) filtering or less than four bytes for L4 destination port (DP) filtering are expected to cause a mismatch. However, the inverse filtering unduly flags a match and the corrupted packets are forwarded to the software application. The L4 stack gets incomplete packet and drops it.

*Note: The perfect filtering correctly reports a mismatch.*

#### Workaround

None.

### 2.15.2 Rx DMA may fail to recover upon DMA restart following a bus error, with Rx timestamping enabled

#### Description

When the timestamping of Rx packets is enabled, some or all of the received packets can have Rx timestamp which is written into a descriptor upon the completion of the Rx packet/status transfer.

However, due to a defect, when bus error occurs during the descriptor read (that is subsequently used as context descriptor to update the Rx timestamp), the context descriptor write is skipped by DMA. Also, Rx DMA does not flush the Rx timestamp stored in the intermediate buffers during the error recovery process and enters Stop state. Due to this residual timestamp in the intermediate buffer, Rx DMA, after being restarted, does not transfer packets.

#### Workaround

Issue a soft reset to drop all Tx packets and Rx packets present inside the controller at the time of bus error. After the soft reset, reconfigure the controller and re-create the descriptors.

*Note: The workaround introduces additional latency.*

### 2.15.3 Tx DMA may halt while fetching TSO header under specific conditions

#### Description

When fetching header bytes from the system memory by issuing a one-beat request in TSO mode, Tx DMA may get into a deadlock state if

- address-aligned beats are enabled (AAL bit of the ETH_DMASBMR register is set),
- DMA start address of a burst transfer request (BUF1AP of TDES0 descriptor) is not aligned on a boundary of the configured data width (64 bits), and
- DMA start address of a burst transfer request is not aligned on a boundary of the programmed burst length (TxPBL bit of the ETH_DMACiTXCR (i = 0 or 1) register).

#### Workaround

Ensure that Tx DMA initiates a burst of at least two beats when fetching header bytes from the system memory, through one of the following measures:

- Disable address-aligned beats by clearing the AAL bit of the ETH_DMASBMR register.
- Align the buffer address pointing to the packet start to a data width boundary (set the bits[2:0] of the address to zero, for 64-bit data width).

- Set the buffer address pointing to the packet start to a value that ensures a burst of minimum two beats when AAL = 1.

### 2.15.4 Spurious receive watchdog timeout interrupt

#### Description

Setting the RWTU[1:0] bitfield of the ETH_DMAC0RXIWTR register to a non-zero value while the RWT[7:0] bitfield is at zero leads to a spurious receive watchdog timeout interrupt (if enabled) and, as a consequence, to executing an unnecessary interrupt service routine with no packets to process.

#### Workaround

Ensure that the RWTU[1:0] bitfield is not set to a non-zero value while the RWT[7:0] bitfield is at zero. For setting RWT[7:0] and RWTU[1:0] bitfields each to a non-zero value, perform two successive writes. The first is either a byte-wide write to the byte containing the RWT[7:0] bitfield, or a 32-bit write that only sets the RWT[7:0] bitfield and keeps the RWTU[1:0] bitfield at zero. The second is either a byte-wide write to the RWTU[1:0] bitfield or a 32-bit write that sets the RWTU[1:0] bitfield while keeping the RWT[7:0] bitfield unchanged.

### 2.15.5 Incorrect flexible PPS output interval under specific conditions

#### Description

The use of the fine correction method for correcting the IEEE 1588 internal time reference, combined with a large frequency drift of the driving clock from the grandmaster source clock, leads to an incorrect interval of the flexible PPS output used in Pulse train mode. As a consequence, external devices synchronized with the flexible PPS output of the device can go out of synchronization.

#### Workaround

Use the coarse method for correcting the IEEE 1588 internal time reference.

### 2.15.6 Packets dropped in RMII 10Mbps mode due to fake dribble and CRC error

#### Description

When operating with the RMII interface at 10 Mbps, the Ethernet peripheral may generate a fake extra nibble of data repeating the last packet (nibble) of the data received from the PHY interface. This results in an odd number of nibbles and is flagged as a dribble error. As the RMII only forwards to the system completed bytes of data, the fake nibble would be ignored and the issue would have no consequence. However, as the CRC error is also flagged when this occurs, the error-packet drop mechanism (if enabled) discards the packets.

*Note: Real dribble errors are rare. They may result from synchronization issues due to faulty clock recovery.*

#### Workaround

When using the RMII 10 MHz mode, disable the error-packet drop mechanism by setting the FEP bit of the ETH_MTLRXQiOMR (i = 0 or 1) register. Accept packets of transactions flagging both dribble and CRC errors.

### 2.15.7 ARP offload function not effective

#### Description

When the Target Protocol Address of a received ARP request packet matches the device IP address set in the ETH_MACARPAR register, the source MAC address in the SHA field of the ARP request packet is compared with the device MAC address in ETH_MACA0LR and ETH_MACA0HR registers (Address0), to filter out ARP packets that are looping back.

Instead, a byte-swapped comparison is performed by the device. As a consequence, the packet is forwarded to the application as a normal packet with no ARP indication in the packet status, and the device does not generate an ARP response.

For example, with the Address0 set to 0x665544332211:

- If the SHA field of the received ARP packet is 0x665544332211, the ARP response is generated while it should not.

- If the SHA field of the received ARP packet is 0x112233445566, the ARP response not is generated while it should.

**Workaround**

Parse the received frame by software and send the ARP response if the source MAC address matches the byte-swapped Address0.

## Revision history

Table 5. **Document revision history**

| Date | Version | Changes |
|---|---|---|
| 1-Feb-2019 | 1 | Initial release. |

## Contents

1 **Summary of device errata** . . . . . 2

2 **Description of device errata** . . . . . 4

2.1 Arm Cortex-A7 core . . . . . 4

2.1.1 Memory locations might be accessed speculatively due to instruction fetches when HCR.VM is set . . . . . 4

2.1.2 Cache maintenance by set/way operations can execute out of order . . . . . 4

2.1.3 PMU events 0x07, 0x0C, and 0x0E do not increment correctly . . . . . 5

2.1.4 PMU event counter 0x14 does not increment correctly . . . . . 6

2.1.5 Exception mask bits are cleared when an exception is taken in Hyp mode . . . . . 6

2.2 Arm Cortex-M4 core . . . . . 7

2.2.1 Interrupted loads to SP can cause erroneous behavior . . . . . 7

2.2.2 VDIV or VSQRT instructions might not complete correctly when very short ISRs are used . 7

2.2.3 Store immediate overlapping exception return operation might vector to incorrect interrupt 8

2.3 System . . . . . 9

2.3.1 TPIU fails to output sync after the pattern generator is disabled in Normal mode . . . . . 9

2.3.2 Serial-wire multi-drop debug not supported . . . . . 10

2.3.3 HSE external oscillator required in some LTDC use cases . . . . . 10

2.3.4 RCC cannot exit Stop and LP-Stop modes . . . . . 11

2.3.5 Incorrect reset of glitch-free kernel clock switch . . . . . 11

2.3.6 Limitation of aclk/hclk5/hclk6 to 200 MHz when used as SDMMC1/2 kernel clock . . . . . 12

2.4 DDRPHYC . . . . . 12

2.4.1 DDRPHYC overconsumption upon reset or Standby mode exit . . . . . 12

2.5 DMAMUX . . . . . 12

2.5.1 SOFx not asserted when writing into DMAMUX_CFR register . . . . . 12

2.5.2 OFx not asserted for trigger event coinciding with last DMAMUX request . . . . . 13

2.5.3 OFx not asserted when writing into DMAMUX_RGCFR register . . . . . 13

2.5.4 Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event . . . . . 13

2.6 QUADSPI . . . . . 13

2.6.1 Memory-mapped read of last memory byte fails . . . . . 13

2.7 ADC . . . . . 14

2.7.1 ADC ANA0/ANA1 resolution limited when Gigabit Ethernet is used . . . . . . . . . . . . . . . . . 14
2.7.2 ADC missing codes in differential 16-bit static acquisition . . . . . . . . . . . . . . . . . . . . . . . . . 14
2.8 DTS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
2.8.1 Mode using PCLK & LSE (REFCLK_SEL = 1) should not be used . . . . . . . . . . . . . . . . . . . 14
2.9 DSI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
2.9.1 DSI PHY compliant with MIPI DPHY v0.81 specification, not with DSI PHY v1.0 . . . . . . . 14
2.10 TIM . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
2.10.1 One-pulse mode trigger not detected in master-slave reset + trigger configuration . . . . . . 15
2.11 LPTIM . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
2.11.1 MCU may remain stuck in LPTIM interrupt when entering Stop mode. . . . . . . . . . . . . . . . 15
2.11.2 MCU may remain stuck in LPTIM interrupt when clearing event flag. . . . . . . . . . . . . . . . . 15
2.12 RTC and TAMP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.12.1 Incorrect version register. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.12.2 Calendar initialization may fail in case of consecutive INIT mode entry . . . . . . . . . . . . . . . 16
2.12.3 Alarm flag may be repeatedly set when the core is stopped in debug . . . . . . . . . . . . . . . . 16
2.13 I2C . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.13.1 Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.13.2 Spurious bus error detection in master mode . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
2.13.3 Spurious master transfer upon own slave address match . . . . . . . . . . . . . . . . . . . . . . . . 17
2.14 SPI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.14.1 Master data transfer stall at system clock much faster than SCK . . . . . . . . . . . . . . . . . . . 18
2.14.2 Corrupted CRC return at non-zero UDRDET setting . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.14.3 TXP interrupt occurring while SPI disabled . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.15 ETH . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
2.15.1 Incorrect L4 inverse filtering results for corrupted packets. . . . . . . . . . . . . . . . . . . . . . . . 19
2.15.2 Rx DMA may fail to recover upon DMA restart following a bus error, with Rx timestamping enabled . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
2.15.3 Tx DMA may halt while fetching TSO header under specific conditions . . . . . . . . . . . . . . 19
2.15.4 Spurious receive watchdog timeout interrupt. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.15.5 Incorrect flexible PPS output interval under specific conditions . . . . . . . . . . . . . . . . . . . . 20
2.15.6 Packets dropped in RMII 10Mbps mode due to fake dribble and CRC error . . . . . . . . . . . 20
2.15.7 ARP offload function not effective . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

**Revision history . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .22**