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, STM32MP151D, STM32MP151F
STM32MP153x	STM32MP153A, STM32MP153C, STM32MP153D, STM32MP153F
STM32MP157x	STM32MP157A, STM32MP157C, STM32MP157D, STM32MP157F

Table 2. Device variants

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32MP151x/3x/7x	B	0x2000
STM32MP151x/3x/7x	Z	0x2001

1. *Refer to the device datasheet for how to identify this code on different types of package.*
2. *REV_ID[15:0] bitfield of DBGMCU_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 = limitation present, workaround available

N = limitation present, no workaround available

P = limitation present, partial workaround available

“-” = limitation absent

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

Table 3. Summary of device limitations

Function	Section	Limitation	Status	
			Rev. B	Rev. Z
Arm Cortex-A7 core	2.1.1	Memory locations might be accessed speculatively due to instruction fetches when HCR.VM is set	A	A
	2.1.2	Cache maintenance by set/way operations can execute out of order	A	A
	2.1.3	PMU events 0x07, 0x0C, and 0x0E do not increment correctly	N	N
	2.1.4	PMU event counter 0x14 does not increment correctly	A	A
	2.1.5	Exception mask bits are cleared when an exception is taken in Hyp mode	N	N
Arm Cortex-M4 core	2.2.1	Interrupted loads to SP can cause erroneous behavior	A	A
	2.2.2	VDIV or VSQRT instructions might not complete correctly when very short ISRs are used	A	A
	2.2.3	Store immediate overlapping exception return operation might vector to incorrect interrupt	A	A
System	2.3.1	TPIU fails to output sync after the pattern generator is disabled in Normal mode	A	A
	2.3.2	Serial-wire multi-drop debug not supported	N	N
	2.3.3	HSE external oscillator required in some LTDC use cases	P	P
	2.3.4	RCC cannot exit Stop and LP-Stop modes	A	A
	2.3.5	Incorrect reset of glitch-free kernel clock switch	P	P
	2.3.6	Limitation of aclk/hclk5/hclk6 to 200 MHz when used as SDMMC1/2 kernel clock	P	P
	2.3.7	Overconsumption in Standby with on-board 1.8 V regulator bypassed	A	-
	2.3.8	eMMC boot timeout too short	A	-
	2.3.9	Cortex-M4 cannot use I/O compensation on Standby mode exit	A	A
	2.3.10	Missed wake-up events in CSTANDBY	N	-
	2.3.12	RCC security settings for Cortex-M4 access are not kept upon wake up from Standby	N	N
	2.3.13	Wakeup pin flags cleared at Standby mode exit with MCU_BEN high and MPU_BEN low	N	N
	2.3.14	Boundary scan data unduly sampled on TCK falling edge	A	A
	2.3.15	Boundary scan SAMPLE/PRELOAD behavior not compliant with IEE1149.1	A	A

Function	Section	Limitation	Status	
			Rev. B	Rev. Z
System	2.3.16	Boot with a specific combination of NAND Flash memories fails	N	-
	2.3.17	Boot with 1-bit error correction on SLC NAND Flash memory fails	A	-
	2.3.18	Boot fails if eMMC does not answer the first command	N	-
	2.3.19	DLYB limits SDMMC throughput	N	N
	2.3.20	LSE CSS cannot be used in VBAT state	N	N
	2.3.21	Wrong value in Coresight M4 ROM table	A	A
DDRPHYC	2.4.1	DDRPHYC overconsumption upon reset or Standby mode exit	A	-
	2.4.2	DDR_CLK jitter out of JEDEC requirement for 32-bit LPDDR2/LPDDR3 at low device Tj	A	A
	2.4.3	Data corruption at low device Tj combined with low 32-bit LPDDR2/ LPDDR3 I/O supply voltage	A	A
GPIO	2.5.1	GPIO assigned to DAC cannot be used in output mode when the DAC output is connected to on-chip peripheral	N	N
DMA	2.6.1	USART/UART/LPUART DMA transfer abort	D	D
DMAMUX	2.7.1	SOFx not asserted when writing into DMAMUX_CFR register	N	N
	2.7.2	OFx not asserted for trigger event coinciding with last DMAMUX request	N	N
	2.7.3	OFx not asserted when writing into DMAMUX_RGCFR register	N	N
	2.7.4	Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event	A	A
QUADSPI	2.8.1	Memory-mapped read of last memory byte fails	P	P
ADC	2.9.1	New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0	A	A
	2.9.2	Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0	A	A
	2.9.3	Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode	A	A
	2.9.4	ADC_AWDy_OUT reset by non-guarded channels	A	A
	2.9.5	ADC ANA0/ANA1 resolution limited when Gigabit Ethernet is used	P	P
	2.9.6	ADC missing codes in differential 16-bit static acquisition	P	P
VREFBUF	2.11.1	VREFBUF Hold mode cannot be used	N	N
	2.11.2	Overshoot on VREFBUF output	A	A
DAC	2.10.1	Invalid DAC channel analog output if the DAC channel MODE bitfield is programmed before DAC initialization	A	A
	2.10.2	DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge	N	N
DTS	2.12.1	Mode using PCLK & LSE (REFCLK_SEL = 1) should not be used	P	P
DSI	2.13.1	DSI PHY compliant with MIPI DPHY v0.81 specification, not with DSI PHY v1.0	N	-
TIM	2.15.1	One-pulse mode trigger not detected in master-slave reset + trigger configuration	P	P
	2.15.2	Consecutive compare event missed in specific conditions	N	N
	2.15.3	Output compare clear not working with external counter reset	P	P
LPTIM	2.16.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	A	A

Function	Section	Limitation	Status	
			Rev. B	Rev. Z
LPTIM	2.16.2	Device may remain stuck in LPTIM interrupt when clearing event flag	P	P
	2.16.3	LPTIM events and PWM output are delayed by 1 kernel clock cycle		
RTC and TAMP	2.17.1	Incorrect version register	N	N
	2.17.2	Calendar initialization may fail in case of consecutive INIT mode entry	A	A
	2.17.3	Alarm flag may be repeatedly set when the core is stopped in debug	N	N
	2.17.4	A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF	N	N
	2.17.5	REFCKON write protection associated to INIT KEY instead of CAL KEY	A	A
	2.17.6	Tamper flag not set on LSE failure detection	N	N
I2C	2.18.1	Wrong data sampling when data setup time (tSU;DAT) is shorter than one I2C kernel clock period	P	P
	2.18.2	Spurious bus error detection in master mode	A	A
	2.18.3	Spurious master transfer upon own slave address match	P	P
	2.18.4	OVR flag not set in underrun condition	N	N
	2.18.5	Transmission stalled after first byte transfer	A	A
USART	2.19.1	Anticipated end-of-transmission signaling in SPI slave mode	A	A
	2.19.2	Data corruption due to noisy receive line	N	N
SPI	2.20.1	Master data transfer stall at system clock much faster than SCK	A	A
	2.20.2	Corrupted CRC return at non-zero UDRDET setting	P	P
	2.20.3	TXP interrupt occurring while SPI disabled	A	A
	2.20.4	Possible corruption of last-received data depending on CRCSIZE setting	A	A
FDCAN	2.21.1	Desynchronization under specific condition with edge filtering enabled	A	A
	2.21.2	Tx FIFO messages inverted under specific buffer usage and priority setting	A	A
	2.21.3	DAR mode transmission failure due to lost arbitration	A	A
OTG_HS	2.22.1	Host packet transmission may hang when connecting through a hub to a low-speed device	N	N
ETH	2.23.1	Incorrect L4 inverse filtering results for corrupted packets	N	N
	2.23.2	Rx DMA may fail to recover upon DMA restart following a bus error, with Rx timestamping enabled	A	A
	2.23.3	Spurious receive watchdog timeout interrupt	A	A
	2.23.4	Incorrect flexible PPS output interval under specific conditions	A	A
	2.23.5	Packets dropped in RMII 10Mbps mode due to fake dribble and CRC error	A	A
	2.23.6	ARP offload function not effective	A	A
	2.23.7	Slight imbalance of AV traffic CBS bandwidth allocation	N	N
	2.23.8	Spurious checksum error upon MTL pending Tx queue flush	N	N
	2.23.9	Bus error coinciding with start-of-packet corrupts MAC-generated packet transmission	N	N
	2.23.10	DMA spurious state upon AXI DMA slave bus error	P	P

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

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

Function	Section	Documentation erratum
System	2.3.11	SSCG option is not available on PLL3 and PLL4
HASH	2.14.1	Superseded suspend sequence for data loaded by DMA
	2.14.2	Superseded suspend sequence for data loaded by the CPU

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

Reference manual and 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).
- Ensure that every interrupt service routine contains more than 2 instructions in addition to the exception return instruction.

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 5. Use of external oscillator on HSE pins

<table>
<tr><th>LTDC connected to</th><th colspan="2">Conditions</th><th>External oscillator on HSE</th></tr>
<tr><td rowspan="2">HDMI bridge</td><td rowspan="2">HDMI bridge close to the device
load ~15 pf</td><td>3 V < V_{DD}< 3.6 V
f_{pixel} up to 74.25 MHz (1280x720 at 60 fps)
OSPEEDR = 1 for all LTDC signals
USB HS used</td><td>Mandatory</td></tr>
<tr><td>1.7 V < V_{DD}< 2 V
f_{pixel} up to 74.25 MHz
Duty cycle = 40 %
0.1 V_{DD} < V_{IN} < 0.9 V_{DD}
OSPEEDR = 3 for LCD_CLK
OSPEEDR = 2 for all other LTDC signals</td><td>Recommended</td></tr>
<tr><td rowspan="3">Parallel LCD</td><td rowspan="3">load < 30 pf</td><td>3 V < V_{DD}< 3.6 V
f_{pixel} up to 90 MHz (1366x768 at 60 fps)
OSPEEDR = 2 for LCD_CLK
OSPEEDR = 1 for all other LTDC signals
USB HS used</td><td>Mandatory</td></tr>
<tr><td>3 V < V_{DD}< 3.6 V
f_{pixel} up to 48 MHz (800x600 at 60 fps)
OSPEEDR = 1 for LCD_CLK
OSPEEDR = 0 for all other LTDC signals</td><td rowspan="2">Recommended</td></tr>
<tr><td>1.7 V < V_{DD}< 2 V
f_{pixel} up to 69 MHz (1024x768 at 60 fps)
Duty cycle = 40 %
0.1 V_{DD} < V_{IN} < 0.9 V_{DD}
OSPEEDR = 3 for LCD_CLK
OSPEEDR = 2 for all other LTDC signals</td></tr>
</table>

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 V_{DDCORE} 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 (in case V_{DD} is below 2.745V, connection should be done using a capacitor with a value of 1/10th of the capacitor value present between NRST and GND, ensure also that potential capacitors between NRST_CORE to GND are removed).

   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.

2.3.7 Overconsumption in Standby with on-board 1.8 V regulator bypassed

Description

The device directly supplied from 1.8 V applied on the VDDA1V8_REG pin (with the on-board regulator bypassed by connecting BYPASS_REG1V8 to V_{DD}) exhibits an overconsumption of about 900 µA in Standby mode (with the V_{DDCORE} supply shutdown) or when NRST_CORE is active, due to an excessive leakage through the VDDA1V8_REG pin.

Workaround

Shut the VDDA1V8_REG pin supply down during Standby, either using a power switch between VDD and VDDA1V8_REG pins or a dedicated voltage regulator with shutdown command connected to an OR combination (for example using diodes) of PWR_ON and NRST signals.

2.3.8 eMMC boot timeout too short

Description

Direct boot from eMMC device (BootROM code execution directly followed by eMMC boot code execution) may fail, due to eMMC timeout detected during the BootROM code execution, causing a fallback to serial/USB boot selection.

The issue occurs if after the reset command (CMD0 followed with 0xF0F0F0F0), the JESD84-B51 boot operation (CMD kept low for over 74 clock cycles), and ACK receipt, the eMMC device does not respond with boot partition data within 10 ms (time allowed by the BootROM code).

Workaround

For direct boot, use an eMMC device that meets the required timing. Consult the eMMC device vendor if it is necessary for assessing the robustness of the solution with their eMMC. As an indication, Toshiba THGBMNG5D1LBAIL, THGBMDG5D1LBAIL, and Kingston EMMC04G-M627-X03U (all 4 Gbyte devices) appear to operate correctly on the STM32MP157C-EV1 board.

Alternatively, use another kind of Flash memory device, such as SD-card, serial-NOR, serial-NAND, or SLC-NAND, to house a part of the boot code to execute directly after the BootROM code, and to be followed by a second part of the boot code housed in the eMMC device.

2.3.9 Cortex-M4 cannot use I/O compensation on Standby mode exit

Description

As the CSION bit of the RCC_OCENSETR register is secured upon exiting Standby mode, the Cortex-M4 core cannot enable the CSI oscillator required for activating the I/O compensation function and thus for enabling high-speed I/O operation of Cortex-M4 peripherals. The Cortex-A7 secure core can set the CSION bit but it is not systematically woken up upon exiting Standby mode.

Workaround

For I/Os of Cortex-M4 peripherals required on Standby exit and until CSI is started by the Cortex-A7 core, only use IOSPEEDR[1:0] settings 00 and 01 (as they do not require I/O compensation).

2.3.10 Missed wake-up events in CSTANDBY

Description

While the Cortex-A7 core is in CSTANDBY mode, WWDG (EXTI event 68) or Cortex-M4 reset (EXTI event 73) wake-up event may cause the RCC state to change faster than the state of PWR. As a consequence, the Cortex-A7 core does not wake up.

Workaround

None.

2.3.11 SSCG option is not available on PLL3 and PLL4

Description

Some reference manual revisions may omit the information that the spread-spectrum clock generation feature is not available on PLL3 and PLL4. The bits RCC_PLL3CR and RCC_PLL4CR of the SSCG_CTRL register must be kept at zero. The registers RCC_PLL3CSGR and RCC_PLL4CSGR must not be used.

This is a documentation issue rather than a device limitation.

Workaround

Assuming that this instruction is followed, no application workaround is required.

2.3.12 RCC security settings for Cortex-M4 access are not kept upon wake up from Standby

Description

When the Cortex-M4 core wakes up from Standby, the TZEN and MCKPROT bits of the RCC_TZCR register are set high as by default, regardless their previous settings upon entering Standby. Until the Cortex-A7 core is started, the operation of the device is limited as follows:

- Some RCC settings are not accessible.
- None of HSE, CSI, LSI, and PLL3 can be started.
- The only possible Cortex-M4 core clock source is HSI (64MHz).
- The only possible per_clk is HSI.
- HSI cannot be set to keep running when Cortex-M4 core goes in CSTOP. (The HSIKERON bit is not accessible.)
- Peripherals secured by default cannot be accessed by the Cortex-M4 core.
- Only PLL4 on HSI is usable for peripherals having a PLL output as clock source option.

Workaround

None.

2.3.13 Wakeup pin flags cleared at Standby mode exit with MCU_BEN high and MPU_BEN low

Description

Upon Standby mode exit while the MCU_BEN bit is high and the MPU_BEN bit low, all wakeup flags in the PWR_WKUPFR register are unduly cleared, which prevents the detectability of the wakeup source.

Workaround

None.

2.3.14 Boundary scan data unduly sampled on TCK falling edge

Description

In Boundary scan mode, the TDI input is sampled on the falling edge of TCK, which is contrary to the IEEE1149.1 requirements and to the device JTAG timing specifications. Depending on the timing of the signal received on the TDI input, this leads to a risk of data corruption/shift. Other JTAG or SWD modes, such as debug, are exempt of this limitation. The TDO timing is not affected, either.

Workaround

Ensure that the input signal on the TDI pin holds for at least 1 ns after the TCK falling edge.

Note: The BSDL description file provided takes into account this design issue and adds a dummy cycle.

2.3.15 Boundary scan SAMPLE/PRELOAD behavior not compliant with IEE1149.1

Description

IEEE1149.1 expects the SAMPLE/PRELOAD instruction to be non-invasive. In the device however, it behaves as an EXTEST instruction, instead. This potentially causes the scanned pins to toggle to levels unsafe from the application perspective.

Workaround

Do not use SAMPLE/PRELOAD instruction. Instead, use the specific CUSTOM_HIGHZ instruction (00011) that allows safe sampling of the pins, by setting them into high-Z state during the sampling phase.

Note: *During the board reset phase (for example during the power-up, when NRST is driven low), all the scannable pins are set to high-Z state. This requires a safe board design ensuring defined pin levels where applicable, for example through the use of pull-up or pull-down resistors.*

2.3.16 Boot with a specific combination of NAND Flash memories fails

Description

The boot process fails in the following configuration:

- SLC NAND Flash memory on FMC is set as the primary boot source (OTP WORD3[29:27] = 001).
- An SLC NAND Flash memory is present and empty (or with only corrupted binary headers).
- Serial NAND Flash memory is set as secondary boot source (OTP WORD3[26:24] = 101).

Workaround

None.

2.3.17 Boot with 1-bit error correction on SLC NAND Flash memory fails

Description

With an SLC NAND Flash memory with 1-bit Hamming ECC (either ONFI-compliant Flash memory or 1-bit Hamming ECC set via the OTP WORD9), connected through the FMC, the BootROM code unduly reads the three *Hamming 1-bit ECC* bytes in a reverse order with respect to their programming order. As a result, the ECC signals data corruption and the boot systematically fails.

Workaround

Apply one of the following measures:

- In the OTP WORD9, set the fmc_ecc_bit_nb[2:0] bitfield to 010 (BCH4) instead to 001 (Hamming). Set all the other bitfields in accordance with the parameters of the SLC NAND Flash memory used.
- Program the three ECC bytes of the SLC NAND Flash memory spare array in the reverse order.

Caution: If applied to a device version not impacted by the issue, the latter measure causes the boot process to fail.

2.3.18 Boot fails if eMMC does not answer the first command

Description

If an eMMC does not answer (for example, because it is not yet powered) the first command sent to CMD line, the boot routine remains in an infinite loop with no timeout, which causes the boot process to fail.

Workaround

None.

2.3.19 DLYB limits SDMMC throughput

Description

When DLYB is used SDMMC could exhibit FIFO underrun or overrun due to internal resynchronization and buffering limitation. This is seen on eMMC HS200 mode, on SD-Card SDR50, and SDR104 mode.

Workaround

Do not use DLYB. With eMMC, use up to DDR mode (52 MHz clock, 104 MBytes/s), with SD-Card, use up to DDR50 (50 MHz clock, 50 MBytes/s).

2.3.20 LSE CSS cannot be used in VBAT state

Description

Under certain circumstances, switching V_{SW} from VDD to VBAT and vice-versa, could falsely trigger the LSE CSS. This would stop the RTC clock, generate false tamper detection and result in the unwanted deletion of the BKPSRAM or Backup REG.

Workaround

Do not enable LSE CSS when VBAT mode is planned to be used.

2.3.21 Wrong value in Coresight M4 ROM table

Description

CM4ROM_PIDR4 and CM4ROM_PIDR0 ROM tables are wrongly defined to 0x04 and 0x50 respectively whereas they should be set to 0x05 and 0x00 respectively. This only impacts the debugging tools which normally identify the target using this ROM table.

Workaround

Use DBGMCU_IDC to identify the device.

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.

2.4.2 DDR_CLK jitter out of JEDEC requirement for 32-bit LPDDR2/LPDDR3 at low device Tj

Description

At extreme low junction temperatures close to -40°C, the JEDEC JESD209-2B and JESD209-3F (for LPDDR2 and LPDDR3, resp.) maximum allowed jitter specification $t_{JIT(per)}$ of ±90 ps at 533 MHz may not be respected when operating with 32-bit LPDDR2 or LPDDR3 memory devices, despite respecting the design rules as per the AN5122 application note and despite optimizing the PCB routing for signal integrity.

Workaround

Use one of the following measures or their combination:

- Increase $V_{DDQ\_DDR}/V_{DD2}/V_{DDQ}/V_{DDCA}$ memory I/O supply (to, for example, 1.25 V).
- Increase the drive output impedance Z0 to 53 Ω, by setting the ZPROG[3:0] bitfield of the DDRPHYC_ZQ0CR1 register to 0x8.

2.4.3 Data corruption at low device T_j combined with low 32-bit LPDDR2/LPDDR3 I/O supply voltage

Description

Very low device junction temperature (such as below -20°C) increases the risk of data corruption for accesses to 32-bit LPDDR2/LPDDR3 memory with low I/O supply voltage ($V_{DDQ\_DDR}/V_{DD2}/V_{DDQ}/V_{DDCA}$), such as 1.2 V or less. The lower the device junction temperature and the memory I/O supply voltage, the higher the risk.

Note: The use of 16-bit LPDDR2/LPDDR3 memory devices prevents the issue.

Workaround

Ensure that the operating conditions of the end volume product are never worse, in terms of the device junction temperature and the LPDDR2/LPDDR3 memory I/O supply voltage, than the conditions applied in the (successful) product engineering and qualification stages. In particular, ensure that the memory I/O supply voltage always exceeds 1.2 V. Alternatively, use a 16-bit LPDDR2/LPDDR3 memory device.

2.5 GPIO

2.5.1 GPIO assigned to DAC cannot be used in output mode when the DAC output is connected to on-chip peripheral

Description

When a DAC output is connected only to an on-chip peripheral, the corresponding GPIO is expected to be available as an output for any other functions.

However, when the DAC output is configured for on-chip peripheral connection only, the GPIO output buffer remains disabled and cannot be used in output mode (GPIO or alternate function). It can still be used in input or analog mode.

This limitation applies to DAC1_OUT1 and DAC1_OUT2 connected to PA4 and PA5, respectively.

Workaround

None.

2.6 DMA

2.6.1 USART/UART/LPUART DMA transfer abort

Description

Some reference manual revisions may unduly present the bit 20 (TRBUFF in the corrected revisions) of the DMA_SxCR register as reserved, to be kept at reset value (low). This bit must be set to ensure the completion of USART/UART/LPUART DMA transfer when another DMA transfer is requested concurrently. Otherwise, it may occur that the other DMA transfer request is not served and that it leads to aborting the ongoing USART/UART/LPUART DMA transfer.

This is a documentation issue rather than a device limitation.

Workaround

No application workaround is required if the TRBUFF bit is used as indicated.

2.7 DMAMUX

2.7.1 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.

2.7.2 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.

2.7.3 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.

2.7.4 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.

2.8 QUADSPI

2.8.1 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.

2.9 ADC

2.9.1 New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0

Description

Once an injected conversion sequence is complete, the queue is consumed and the context changes according to the new ADC_JSQR parameters stored in the queue. This new context is applied for the next injected sequence of conversions.

However, the programming of the new context in ADC_JSQR (change of injected trigger selection and/or trigger polarity) may launch the execution of this context without waiting for the trigger if:

- the queue of context is enabled (JQDIS cleared to 0 in ADC_CFGR), and
- the queue is never empty (JQM cleared to 0 in ADC_CFGR), and
- the injected conversion sequence is complete and no conversion from previous context is ongoing

Workaround

Apply one of the following measures:
- Ignore the first conversion.
- Use a queue of context with JQM = 1.
- Use a queue of context with JQM = 0, only change the conversion sequence but never the trigger selection and the polarity.

2.9.2 Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0

Description

When an injected conversion sequence is complete and the queue is consumed, writing a new context in ADC_JSQR just after the completion of the previous context and with a length longer that the previous context, may cause both contexts to fail. The two contexts are considered as one single context. As an example, if the first context contains element 1 and the second context elements 2 and 3, the first context is consumed followed by elements 2 and 3 and element 1 is not executed.

This issue may happen if:
- the queue of context is enabled (JQDIS cleared to 0 in ADC_CFGR), and
- the queue is never empty (JQM cleared to 0 in ADC_CFGR), and
- the length of the new context is longer than the previous one

Workaround

If possible, synchronize the writing of the new context with the reception of the new trigger.

2.9.3 Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode

Description

In Dual ADC mode, an unexpected regular conversion may start at the end of the second injected conversion without a regular trigger being received, if the second injected conversion starts exactly at the same time than the end of the first injected conversion. This issue may happen in the following conditions:
- two consecutive injected conversions performed in Interleaved simultaneous mode (DUAL[4:0] of ADC_CCR = 0b00011), or
- two consecutive injected conversions from master or slave ADC performed in Interleaved mode (DUAL[4:0]of ADC_CCR = 0b00111)

Workaround

- In Interleaved simultaneous injected mode: make sure the time between two injected conversion triggers is longer than the injected conversion time.
- In Interleaved only mode: perform injected conversions from one single ADC (master or slave), making sure the time between two injected triggers is longer than the injected conversion time.

2.9.4 ADC_AWDy_OUT reset by non-guarded channels

Description

ADC_AWDy_OUT is set when a guarded conversion of a regular or injected channel is outside the programmed thresholds. It is reset after the end of the next guarded conversion that is inside the programmed thresholds.

However, the ADC_AWDy_OUT signal is also reset at the end of conversion of non-guarded channels, both regular and injected.

Workaround

When ADC_AWDy_OUT is enabled, it is recommended to use only the ADC channels that are guarded by a watchdog.

If ADC_AWDy_OUT is used with ADC channels that are not guarded by a watchdog, take only ADC_AWDy_OUT rising edge into account.

2.9.5 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.

2.9.6 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.

2.10 DAC

2.10.1 Invalid DAC channel analog output if the DAC channel MODE bitfield is programmed before DAC initialization

Description

When the DAC operates in Normal mode and the DAC enable bit is cleared, writing a value different from 000 to the DAC channel MODE bitfield of the DAC_MCR register before performing data initialization causes the corresponding DAC channel analog output to be invalid.

Workaround

Apply the following sequence:

1. Perform one write access to any data register.
2. Program the MODE bitfield of the DAC_MCR register.

2.10.2 DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge

Description

When the DAC channel operates in DMA mode (DMAEN of DAC_CR register set), the DMA channel underrun flag (DMAUDR of DAC_SR register) fails to rise upon an internal trigger detection if that detection occurs during the same clock cycle as a DMA request acknowledge. As a result, the user application is not informed that an underrun error occurred.

This issue occurs when software and hardware triggers are used concurrently to trigger DMA transfers.

Workaround

None.

2.11 VREFBUF

2.11.1 VREFBUF Hold mode cannot be used

Description

VREFBUF can be configured to operate in Hold mode to reduce current consumption.

When VREFBUF enters Hold mode (by setting both HIZ and ENVR bits of the VREFBUF_CSR register), the VREF+ I/O transits to high impedance mode. If not discharged externally, the capacitor on the VREF+ pin keeps its charge and voltage. Exiting VREFBUF Hold mode (by clearing the HIZ bit) in this condition might lead to a voltage overshoot on the VREF+ output.

Workaround

None.

2.11.2 Overshoot on VREFBUF output

Description

An overshoot might occur on VREFBUF output if VREF+ pin has residual voltage when VREFBUF is enabled (ENVR is set in VREFBUF_CSR register).

Workaround

Let the voltage on the VREF+ pin drop to 1 V under the target V_{REFBUF_OUT}. This can be achieved by switching VREFBUF buffer off (ENVR is cleared and HIZ is cleared in VREFBUF_CSR register) allowing sufficient time to discharge the capacitor on the VREF+ pin through the VREFBUF pull-down resistor.

2.12 DTS

2.12.1 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).

2.13 DSI

2.13.1 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.

2.14 HASH

2.14.1 Superseded suspend sequence for data loaded by DMA

Description

The section *HASH / Context swapping / Data loaded by DMA / Current context saving* of some reference manual revisions may suggest the following suspend sequence for using HASH with DMA:

1. Clear the DMAE bit to disable the DMA interface.
2. Wait until the current DMA transfer is complete (wait for DMAS = 0 in the HASH_SR register).

This recommendation is obsolete and superseded with the following sequence that suspends then resumes the secure digest computing in order to swap the context:

Suspend:

1. In Polling mode, wait for BUSY = 0. If the DCIS bit of the HASH_SR register is set, the hash result is available and the context swapping is useless. Otherwise, go to step 2.
2. In Polling mode, wait for BUSY = 1.
3. Disable the DMA channel. Then clear the DMAE bit of the HASH_CR register.
4. In Polling mode, wait for BUSY = 0. If the DCIS bit of the HASH_SR register is set, the hash result is available and the context swapping is useless. Otherwise, go to step 5.
5. Save the HASH_IMR, HASH_STR, HASH_CR, and HASH_CSR0 to HASH_CSR37 registers. The HASH_CSR38 to HASH_CSR53 registers must also be saved if an HMAC operation is ongoing.

Resume:

1. Reconfigure the DMA controller so that it proceeds with the transfer of the message up to the end if it is not interrupted again. Do not forget to take into account the words already pushed into the FIFO if NBW[3:0] is higher than 0x0.
2. Program the values saved in memory to the HASH_IMR, HASH_STR, and HASH_CR registers.
3. Initialize the hash processor by setting the INIT bit of the HASH_CR register.
4. Program the values saved in memory to the HASH_CSRx registers.
5. Restart the processing from the point of interruption, by setting the DMAE bit.

Note: *To optimize the resume process when NBW[3:0] = 0x0, HASH_CSR22 to HASH_CSR37 registers do not need to be saved then restored as the FIFO is empty.*

This is a documentation issue rather than a product limitation.

Workaround

No application workaround is required as long as the new sequence is applied.

2.14.2 Superseded suspend sequence for data loaded by the CPU

Description

The section *HASH / Context swapping / Data loaded by software* of some reference manual revisions may instruct that *"the user application must wait until DINIS ≠ 1 (last block processed and input FIFO empty) or NBW 0 (FIFO not full and no processing ongoing)"*.

This instruction is obsolete and superseded with the following:

When the DMA is not used to load the message into the hash processor, the context can be saved only when no block processing is ongoing.

To suspend the processing of a message, proceed as follows after writing 16 words 32-bit (plus one if it is the first block):

1. In Polling mode, wait for BUSY = 0, then poll if the DINIS status bit is set to 1. In Interrupt mode, implement the next step in DINIS interrupt handler (recommended).

2. Store the contents of the following registers into memory:
   – HASH_IMR
   – HASH_STR
   – HASH_CR
   – HASH_CSR0 to HASH_CSR37 and, if an HMAC operation is ongoing, also HASH_CSR38 to HASH_CSR53

To resume the processing of a message, proceed as follows:

1. Write the HASH_IMR, HASH_STR, and HASH_CR registers with the values saved in memory.
2. Initialize the hash processor by setting the INIT bit of the HASH_CR register.
3. Write the HASH_CSRx registers with the values saved in memory.
4. Restart the processing from the point of interruption.

Note: To optimize the resume process when NBW[3:0]=0x0, HASH_CSR22 to HASH_CSR37 registers do not need to be saved then restored as the FIFO is empty.

This is a documentation issue rather than a product limitation.

Workaround

No application workaround is required as long as the new sequence is applied.

2.15 TIM

2.15.1 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 decreasing the timer latency to external trigger events.

2.15.2 Consecutive compare event missed in specific conditions

Description

Every match of the counter (CNT) value with the compare register (CCR) value is expected to trigger a compare event. However, if such matches occur in two consecutive counter clock cycles (as consequence of the CCR value change between the two cycles), the second compare event is missed for the following CCR value changes:

- in edge-aligned mode, from ARR to 0:
  – first compare event: CNT = CCR = ARR
  – second (missed) compare event: CNT = CCR = 0
- in center-aligned mode while up-counting, from ARR-1 to ARR (possibly a new ARR value if the period is also changed) at the crest (that is, when TIMx_RCR = 0):
  – first compare event: CNT = CCR = (ARR-1)
  – second (missed) compare event: CNT = CCR = ARR

- in center-aligned mode while down-counting, from 1 to 0 at the valley (that is, when TIMx_RCR = 0):
  – first compare event: CNT = CCR = 1
  – second (missed) compare event: CNT = CCR = 0

This typically corresponds to an abrupt change of compare value aiming at creating a timer clock single-cycle-wide pulse in toggle mode.

As a consequence:

- In toggle mode, the output only toggles once per counter period (squared waveform), whereas it is expected to toggle twice within two consecutive counter cycles (and so exhibit a short pulse per counter period).
- In center mode, the compare interrupt flag does note rise and the interrupt is not generated.

Note: The timer output operates as expected in modes other than the toggle mode.

Workaround

None.

2.15.3 Output compare clear not working with external counter reset

Description

The output compare clear event (ocref_clr) is not correctly generated when the timer is configured in the following slave modes: Reset mode, Combined reset + trigger mode, and Combined gated + reset mode.

The PWM output remains inactive during one extra PWM cycle if the following sequence occurs:

1. The output is cleared by the ocref_clr event.
2. The timer reset occurs before the programmed compare event.

Workaround

Apply one of the following measures:

- Use BKIN (or BKIN2 if available) input for clearing the output, selecting the Automatic output enable mode (AOE = 1).
- Mask the timer reset during the PWM ON time to prevent it from occurring before the compare event (for example with a spare timer compare channel open-drain output connected with the reset signal, pulling the timer reset line down).

2.16 LPTIM

2.16.1 Device 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 device from Stop mode may be frozen in active state. Consequently, when trying to enter Stop mode, this limitation prevents the device 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.

2.16.2 Device 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 in LPTIM_ISR register by writing its corresponding bit in LPTIM_ICR 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 high.

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 device 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.

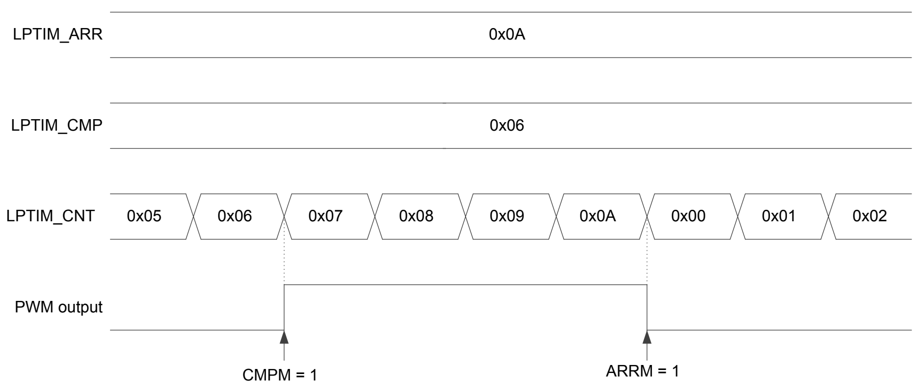
2.16.3 LPTIM events and PWM output are delayed by 1 kernel clock cycle

Description

The compare match event (CMPM), auto reload match event (ARRM), PWM output level and interrupts are updated with a delay of one kernel clock cycle.

Consequently, it is not possible to generate PWM with a duty cycle of 0% or 100%.

The following waveform gives the example of PWM output mode and the effect of the delay:



Workaround

Set the compare value to the desired value minus 1. For instance in order to generate a compare match when LPTM_CNT = 0x08, set the compare value to 0x07.

2.17 RTC and TAMP

2.17.1 Incorrect version register

Description

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

Workaround

None.

2.17.2 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.

2.17.3 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.

2.17.4 A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF

Description

With the timestamp on tamper event enabled (TAMPTS bit of the RTC_CR register set), a tamper event is ignored if it occurs:

- within four APB clock cycles after setting the CTSF bit of the RTC_SCR register to clear the TSF flag, while the TSF flag is not yet effectively cleared (it fails to set the TSOVF flag)
- within two ck_apre cycles after setting the CTSF bit of the RTC_SCR register to clear the TSF flag, when the TSF flag is effectively cleared (it fails to set the TSF flag and timestamp the calendar registers)

Workaround

None.

2.17.5 REFCKON write protection associated to INIT KEY instead of CAL KEY

Description

The write protection of the REFCKON bit is unlocked if the key sequence is written in RTC_WPR with the privilege and security rights set by the INITPRIV and INITSEC bits, instead of being set by the CALPRIV and CALSEC bits.

Workaround

Unlock the INIT KEY before writing REFCKON.

2.17.6 Tamper flag not set on LSE failure detection

Description

With the timestamp on tamper event enabled (the TAMPTS bit of the RTC_CR register set), the LSE failure detection (LSE clock stopped) event connected to the internal tamper 3 fails to raise the ITAMP3F and ITAMP3MF flags, although it duly erases or blocks (depending on the internal tamper 3 configuration) the backup registers and other device secrets, and the RTC and TAMP peripherals resume normally upon the LSE restart.

Note: As expected in this particular case, the TSF and TSMF flags remain low as long as LSE is stopped as they require running RTCCLK clock to operate.

Workaround

None.

2.18 I2C

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

Description

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

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

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

Workaround

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

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

2.18.2 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^2C-bus transfer in master mode and any such transfer continues normally.

Workaround

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

2.18.3 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^2C-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.

2.18.4 OVR flag not set in underrun condition

Description

In slave transmission with clock stretching disabled (NOSTRETCH = 1 in the I2C_CR1 register), an underrun condition occurs if the current byte transmission is completed on the I2C bus, and the next data is not yet written in the TXDATA[7:0] bitfield. In this condition, the device is expected to set the OVR flag of the I2C_ISR register and send 0xFF on the bus.

However, if the I2C_TXDR is written within the interval between two I2C kernel clock cycles before and three APB clock cycles after the start of the next data transmission, the OVR flag is not set, although the transmitted value is 0xFF.

Workaround

None.

2.18.5 Transmission stalled after first byte transfer

Description

When the first byte to transmit is not prepared in the TXDATA register, two bytes are required successively, through TXIS status flag setting or through a DMA request. If the first of the two bytes is written in the I2C_TXDR register in less than two I2C kernel clock cycles after the TXIS/DMA request, and the ratio between APB clock and I2C kernel clock frequencies is between 1.5 and 3, the second byte written in the I2C_TXDR is not internally detected. This causes a state in which the I2C peripheral is stalled in master mode or in slave mode, with clock stretching enabled (NOSTRETCH = 0). This state can only be released by disabling the peripheral (PE = 0) or by resetting it.

Workaround

Apply one of the following measures:
- Write the first data in I2C_TXDR before the transmission starts.
- Set the APB clock frequency so that its ratio with respect to the I2C kernel clock frequency is lower than 1.5 or higher than 3.

2.19 USART

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

Description

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

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

Workaround

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

2.19.2 Data corruption due to noisy receive line

Description

In UART mode with oversampling by 8 or 16 and with 1 or 2 stop bits, the received data may be corrupted if a glitch to zero shorter than the half-bit occurs on the receive line within the second half of the stop bit.

Workaround

None.

2.20 SPI

2.20.1 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.

2.20.2 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.

2.20.3 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.

2.20.4 Possible corruption of last-received data depending on CRCSIZE setting

Description

With the CRC calculation disabled (CRCEN = 0), the transfer size bitfield set to a value greater than zero (TSIZE[15:0] > 0), and the length of CRC frame set to less than 8 bits (CRCSIZE[4:0] < 00111), the last data received in the RxFIFO may be corrupted.

Workaround

Keep the CRCSIZE[4:0] bitfield at its default setting (00111) during the data reception if CRCEN = 0 and TSIZE[15:0] > 0.

2.21 FDCAN

2.21.1 Desynchronization under specific condition with edge filtering enabled

Description

FDCAN may desynchronize and incorrectly receive the first bit of the frame if:
- the edge filtering is enabled (the EFBI bit of the FDCAN_CCCR register is set), and
- the end of the integration phase coincides with a falling edge detected on the FDCAN_Rx input pin

If this occurs, the CRC detects that the first bit of the received frame is incorrect, flags the received frame as faulty and responds with an error frame.

Note: This issue does not affect the reception of standard frames.

Workaround

Disable edge filtering or wait for frame retransmission.

2.21.2 Tx FIFO messages inverted under specific buffer usage and priority setting

Description

Two consecutive messages from the Tx FIFO may be inverted in the transmit sequence if:
- FDCAN uses both a dedicated Tx buffer and a Tx FIFO (the TFQM bit of the FDCAN_TXBC register is cleared), and
- the messages contained in the Tx buffer have a higher internal CAN priority than the messages in the Tx FIFO.

Workaround

Apply one of the following measures:
- Ensure that only one Tx FIFO element is pending for transmission at any time:

  The Tx FIFO elements may be filled at any time with messages to be transmitted, but their transmission requests are handled separately. Each time a Tx FIFO transmission has completed and the Tx FIFO gets empty (TFE bit of FDACN_IR set to 1) the next Tx FIFO element is requested.
- Use only a Tx FIFO:

  Send both messages from a Tx FIFO, including the message with the higher priority. This message has to wait until the preceding messages in the Tx FIFO have been sent.
- Use two dedicated Tx buffers (for example, use Tx buffer 4 and 5 instead of the Tx FIFO). The following pseudo-code replaces the function in charge of filling the Tx FIFO:

```
Write message to Tx Buffer 4
Transmit Loop:
    Request Tx Buffer 4 - write AR4 bit in FDCAN_TXBAR
    Write message to Tx Buffer 5
    Wait until transmission of Tx Buffer 4 complete (IR bit in FDCAN_IR),
    read TO4 bit in FDCAN_TXBTO
    Request Tx Buffer 5 - write AR5 bit of FDCAN_TXBAR
    Write message to Tx Buffer 4
    Wait until transmission of Tx Buffer 5 complete (IR bit in FDCAN_IR),
    read TO5 bit in FDCAN_TXBTO
```

2.21.3 DAR mode transmission failure due to lost arbitration

Description

In DAR mode, the transmission may fail due to lost arbitration at the first two identifier bits.

Workaround

Upon failure, clear the corresponding Tx buffer transmission request bit TRPx of the FDCAN_TXBRP register and set the corresponding cancellation finished bit CFx of the FDCAN_TXBCF register, then restart the transmission.

2.22 OTG_HS

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

Description

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

Workaround

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

2.23 ETH

2.23.1 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.

2.23.2 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.

2.23.3 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.

2.23.4 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.

2.23.5 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_MTLRXQ0OMR or ETH_MTLRXQ1OMR register. Accept packets of transactions flagging both dribble and CRC errors.

2.23.6 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.

2.23.7 Slight imbalance of AV traffic CBS bandwidth allocation

Description

When a queue is enabled for AV traffic (TXQEN[1:0] = 10 in ETH_ MTLTXQ0OMR or ETH_ MTLTXQ1OMR register) and CBS algorithm selected (AVALG = 1 and CC = 0 in the ETH_MTLTXQ1ECR register), the expected behavior is as follows:

- A queue is eligible for transmission when its credit is positive. Higher credit gives more chance for a queue to be served.
- Credit of any non-empty pending (not currently served) queue is incremented at a programmable rate during the data transmission from another queue.
- Credit of the queue transmitting data is decremented at a programmable rate.
- Credit of any empty queue is nullified.

However, during the transmission from one queue, pending empty queues unduly keep their credits, instead of being nullified.

As a consequence, when an empty queue with undue non-zero credit receives data to transmit, it has a slightly higher chance to be served and thus wins more continuous bandwidth than expected from the CBS algorithm specification.

This non-conformity is negligible with most applications.

Workaround

None.

2.23.8 Spurious checksum error upon MTL pending Tx queue flush

Description

Transmit checksum engine signals packet transmission errors via IHE (IP header error) and PCE (payload checksum error) flags.

When a flush of a pending (not currently served) MTL Tx FIFO queue 0 or 1 is initiated (by setting the FTQ bit of the ETH_MTLTXQ0OMR or ETH_MTLTXQ1OMR register, respectively), the MTL sends a dummy Tx status indicating the flushed packets. The checksum engine is expected to ignore this dummy Tx status.

However, when multiple transmit queues with checksum offload engines are enabled and the drop Tx status disabled (DTXSTS = 0 in the ETH_MTLOMR register), the checksum engine unduly takes into account the dummy Tx status. The MAC Tx status of the packet under transmission then returns zeros in its checksum field, instead of the due value.

As a consequence, the checksum engine may spuriously signal transmission error for the packet under transmission and for the following packet.

Note: The defect is expected to have no impact to the user application as the checksum error flags are intended for debug purposes only.

Workaround

None.

2.23.9 Bus error coinciding with start-of-packet corrupts MAC-generated packet transmission

Description

A bus error coinciding with the start of a new packet unduly aborts the transmission of any internally MAC-generated packet (ARP, PTO, Pause). As a consequence, the packet is sent on the line as a runt frame with corrupted FCS.

The aborted packet is not retransmitted and can cause:

- flow control failure in case of Pause/PFC packet corruption
- delay in ARP handshake from ARP offload engine (the ARP stack recovers because it sends ARP requests periodically)
- delay in PTP response/SYNC packets generated by the PTP offload engine (the PTP stack recovers because it sends request packets periodically)

The occurrence rate of this failure is very low.

Workaround

None.

2.23.10 DMA spurious state upon AXI DMA slave bus error

Description

Normally, upon an AXI DMA slave bus error, the ETH controller triggers a fatal bus error interrupt (by setting the FBE bit of the ETH_DMACSR register) and stops the corresponding DMA channel by clearing the ST bit of the ETH_DMAC0TXCR or ETH_DMAC1TXCR register. The software can then recreate the list of descriptors and restart the Tx DMA channel (set the ST bit).

However, in the following cases, the DMA controller fails to recover from the bus error or it corrupts the TSO/USO header data:

- when the bus error occurs for a packet transfer initiated by a Tx DMA channel 1 and the OSF bit of the ETH_DMAC0TXCR or ETH_DMAC1TXCR register is set
- when the TSO/USO segmentation is enabled in the Tx descriptor

Workaround

Reset and reconfigure the ETH controller by software, then re-create the descriptors. This restores the normal DMA controller operation, although it causes the loss of all latent Tx and Rx packets in the ETH controller and adds processing latency.

Revision history

Table 6. Document revision history

Date	Version	Changes
1-Feb-2019	1	Initial release.
17-Sep-2019	2	Added errata in Section 1 Summary of device errata and Section 2 Description of device errata: • Overconsumption in Standby with on-board 1.8 V regulator bypassed • eMMC boot timeout too short • Cortex-M4 cannot use I/O compensation on Standby mode exit • Missed wake-up events in CSTANDBY • DDR_CLK jitter out of JEDEC requirement for 32-bit LPDDR2/LPDDR3 at low device Tj • Data corruption at low device Tj combined with low 32-bit LPDDR2/ LPDDR3 I/O supply voltage • Invalid DAC channel analog output if the DAC channel MODE bitfield is programmed before DAC initialization • DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge • OVR flag not set in underrun condition • Transmission stalled after first byte transfer • Possible corruption of last-received data depending on CRCSIZE setting • Desynchronization under specific condition with edge filtering enabled • Tx FIFO messages inverted under specific buffer usage and priority setting • DAR mode transmission failure due to lost arbitration
25-Nov-2019	3	Added: • Table 4. Summary of device documentation errata • SSCG option is not available on PLL3 and PLL4 erratum in Table 4. Summary of device documentation errata and Section 2 Description of device errata
21-Jan-2020	4	Added errata in Section 1 Summary of device errata and Section 2 Description of device errata: • RCC security settings for Cortex-M4 access are not kept upon wake up from Standby • GPIO assigned to DAC cannot be used in output mode when the DAC output is connected to on-chip peripheral • Wakeup pin flags cleared at Standby mode exit with MCU_BEN high and MPU_BEN low • Boundary scan data unduly sampled on TCK falling edge • Boundary scan SAMPLE/PRELOAD behavior not compliant with IEE1149.1 • New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0 • Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0 • Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode • Consecutive compare event missed in specific conditions • Output compare clear not working with external counter reset • Anticipated end-of-transmission signaling in SPI slave mode • Data corruption due to noisy receive line • Host packet transmission may hang when connecting through a hub to a low-speed device

Date	Version	Changes
		Added silicon revision Z.
28-Apr-2020	5	Added errata in Section 1 Summary of device errata and Section 2 Description of device errata: • Boot with a specific combination of NAND Flash memories fails • Boot with 1-bit error correction on SLC NAND Flash memory fails • Boot fails if eMMC does not answer the first command • VREFBUF Hold mode cannot be used • Overshoot on VREFBUF output • Superseded suspend sequence for data loaded by DMA • Superseded suspend sequence for data loaded by the CPU • A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF • REFCKON write protection associated to INIT KEY instead of CAL KEY • Tamper flag not set on LSE failure detection
24-Feb-2021	6	Added errata in Section 1 Summary of device errata and Section 2 Description of device errata: • DLYB limits SDMMC throughput • USART/UART/LPUART DMA transfer abort • LSE CSS cannot be used in VBAT state • Wrong value in Coresight M4 ROM table • ADC_AWDy_OUT reset by non-guarded channels • USART/UART/LPUART DMA transfer abort • LPTIM events and PWM output are delayed by 1 kernel clock cycle • Slight imbalance of AV traffic CBS bandwidth allocation • Spurious checksum error upon MTL pending Tx queue flush • Bus error coinciding with start-of-packet corrupts MAC-generated packet transmission • DMA spurious state upon AXI DMA slave bus error Modified: • Incorrect reset of glitch-free kernel clock switch Removed: • Tx DMA may halt while fetching TSO header under specific conditions

Contents