

Errata

MSPM0G321x, MSPM0G320x, MSPM0G321x-Q1, MSPM0G320x-Q1, MSPM0G121x, MSPM0G120x Mixed- Signal Microcontrollers



Abstract

This document describes the known exceptions to the functional specifications (advisories).

Table of Contents

1 Errata notes.....	1
2 Functional Advisories.....	1
3 Preprogrammed Software Advisories.....	3
4 Debug Only Advisories.....	3
5 Fixed by Compiler Advisories.....	3
6 Device Nomenclature.....	3
6.1 Device Symbolization and Revision Identification.....	3
7 Advisory Descriptions.....	5
8 Revision History.....	24

1 Errata notes

The following notes should be considered before reading the erratas list and entries:

- GPIO_ERR_05 and GPIO_ERR_06 no longer affect the device
- SYSOSC_ERR_01 no longer affects the device
- PMCU_ERR_13 no longer affects the device
- RST_ERR_01 no longer affects the device
- SYSCTL_ERR_01, SYSCTL_ERR_05, and SYSCTL_ERR_06 no longer affect the device
- SYSPLL_ERR_01 no longer affects the device
- SYSOSC_ERR_04 no longer affects the device
- VREF_ERR_05 no longer affects the device

2 Functional Advisories

Advisories that affect the device's operation, function, or parametrics.

✓ The check mark indicates that the issue is present in the specified revision.

Errata Number	Rev A	Rev B
AES_ERR_01	✓	✓
BSL_ERR_01	✓	✓
COMP_ERR_06	✓	✓
CPU_ERR_02	✓	✓
CPU_ERR_03	✓	✓
FCC_ERR_01	✓	✓
FLASH_ERR_03	✓	✓
FLASH_ERR_04	✓	✓

Errata Number	Rev A	Rev B
FLASH_ERR_05	✓	✓
FLASH_ERR_08	✓	✓
GPIO_ERR_05	✓	
GPIO_ERR_06	✓	
KEystore_ERR_01	✓	✓
MATHACL_ERR_01	✓	✓
PMCU_ERR_13	✓	
RST_ERR_01	✓	
RST_ERR_03	✓	✓
SPI_ERR_10	✓	✓
SYSCTL_ERR_01	✓	
SYSCTL_ERR_02	✓	✓
SYSCTL_ERR_03	✓	✓
SYSCTL_ERR_04	✓	✓
SYSCTL_ERR_05	✓	
SYSCTL_ERR_06	✓	
SYSCTL_ERR_11	✓	✓
SYSCTL_ERR_12	✓	✓
SYSOSC_ERR_01	✓	✓
SYSOSC_ERR_02	✓	✓
SYSOSC_ERR_04	✓	✓
SYSOSC_ERR_05	✓	✓
SYSOSC_ERR_06	✓	✓
SYSPLL_ERR_01	✓	
TIMER_ERR_04	✓	✓
TIMER_ERR_06	✓	✓
TIMER_ERR_07	✓	✓
UNICOMMI2CC_ERR_01	✓	✓
UNICOMMI2CC_ERR_02	✓	✓
UNICOMMI2CC_ERR_03	✓	✓
UNICOMMSPI_ERR_02	✓	✓
UNICOMMSPI_ERR_03	✓	✓
UNICOMMSPI_ERR_04	✓	✓
UNICOMMUART_ERR_06	✓	✓
UNICOMMUART_ERR_07	✓	✓
UNICOMMUART_ERR_09	✓	✓
UNICOMMUART_ERR_10	✓	✓
UNICOMMUART_ERR_12	✓	✓
UNICOMMUART_ERR_13	✓	✓
UNICOMMUART_ERR_14	✓	✓
UNICOMMUART_ERR_15	✓	✓

Errata Number	Rev A	Rev B
VREF_ERR_05	✓	

3 Preprogrammed Software Advisories

Advisories that affect factory-programmed software.

✓ The check mark indicates that the issue is present in the specified revision.

4 Debug Only Advisories

Advisories that affect only debug operation.

✓ The check mark indicates that the issue is present in the specified revision.

5 Fixed by Compiler Advisories

Advisories that are resolved by compiler workaround. Refer to each advisory for the IDE and compiler versions with a workaround.

✓ The check mark indicates that the issue is present in the specified revision.

6 Device Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all MSP MCU devices. Each MSP MCU commercial family member has one of two prefixes: MSP or XMS. These prefixes represent evolutionary stages of product development from engineering prototypes (XMS) through fully qualified production devices (MSP).

XMS – Experimental device that is not necessarily representative of the final device's electrical specifications

MSP – Fully qualified production device

Support tool naming prefixes:

X: Development-support product that has not yet completed Texas Instruments internal qualification testing.

null: Fully-qualified development-support product.

XMS devices and X development-support tools are shipped against the following disclaimer:

"Developmental product is intended for internal evaluation purposes."

MSP devices have been characterized fully, and the quality and reliability of the device have been demonstrated fully. TI's standard warranty applies.

Predictions show that prototype devices (XMS) have a greater failure rate than the standard production devices. TI recommends that these devices not be used in any production system because their expected end-use failure rate still is undefined. Only qualified production devices are to be used.

TI device nomenclature also includes a suffix with the device family name. This suffix indicates the temperature range, package type, and distribution format.

6.1 Device Symbolization and Revision Identification

The package diagrams below indicate the package symbolization scheme, and [Table 5-1](#) defines the device revision to version ID mapping.

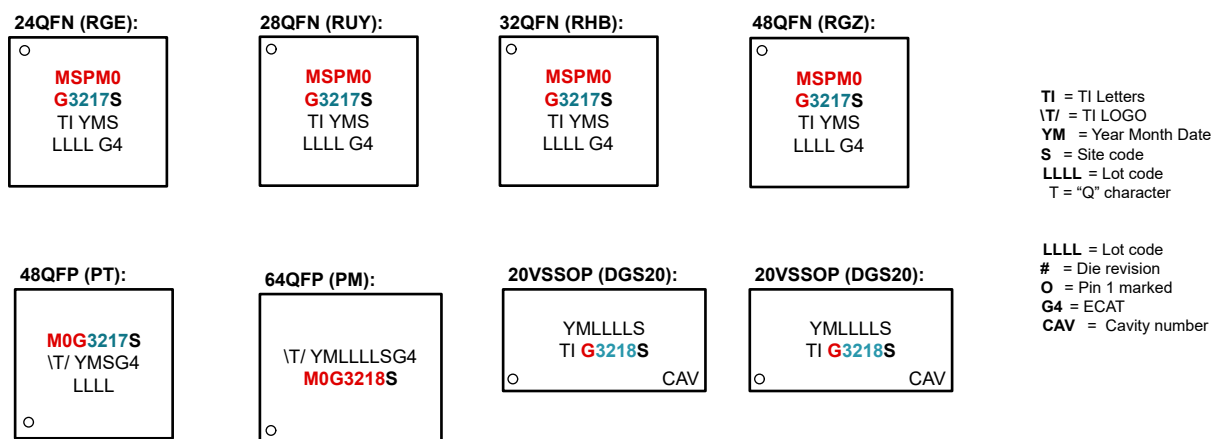


Figure 6-1. Package Symbolization

Table 6-1. Die Revisions

Revision Letter	Product Development Status	DEVICEID.VERSION
A	Prototype (X-Marked)	0
B	RTM	0

The revision letter indicates the product hardware revision. Advisories in this document are marked as applicable or not applicable for a given device based on the revision letter. This letter maps to an integer stored in the factory constants memory of the device (DEVICEID.VERSION field), which can be used to look up the revision using application software or a connected debug probe.

Note

The DEVICEID.VERSION field is incremented with post RTM (Release to Market) product hardware revisions. Within the same VERSION value, users can identify specific variants by the 'X' character on the silkscreen.

7 Advisory Descriptions

AES_ERR_01 ***AES Module***

Category

Functional

Function

AES Saved Context Ready interrupt is not generating as expected

Description

Saved Context Ready interrupt is not getting generated. The interrupt is generated if an access (read or write) is made to any AES register.

Workaround

Use polling based mechanism to check the status bit for Saved Context Ready in CTRL register instead of interrupt.

BSL_ERR_01 ***BSL Module***

Category

Functional

Function

Invoking the BSL through application software will fail under certain conditions

Description

BSL fails to start through invocation within an application (BSL Software Invoke) due to SRAM error code. After a reset, the device will go back to the application instead of invoking BSL due to the error.

This errata does not apply to BSL invocation through hardware invoke methods

Workaround

For applications needing a software invocation of the BSL, clear the entirety of SRAM with assembly code before resetting device. The MSPM0 bsl_software_invoke example within the MSPM0 SDK v 1.20.01.xx or higher contains an example fix within the "bsl_software_invoke" example.

COMP_ERR_06 ***COMP Module***

Category

Functional

Function

COMP output ready signal and interrupt generation will be delayed if SYSOSC is disabled before output is ready

Description

Enabling the COMP or changing any of its input configurations (IPSEL/IMSEL) will trigger the OUTRDYIFG bit to be set once the COMPs output is ready. OUTRDYIFG gates all COMP interrupts from firing until the output is ready.

The COMP OUTRDYIFG bit requires SYSOSC to be at least 4MHz to become set. If SYSOSC is disabled, OUTRDYIFG will be delayed by 2 LFCLK cycles (~60us).

COMP_ERR_06

(continued)

COMP Module**Workaround**

To avoid the 2 LFCLK cycle delay, wait for OUTRDYIFG bit to be set before entering a LPM that disables SYSOSC or changing the clock to LFCLK.

1. Enable Comparator
2. Configure the IPSEL/IMSEL, as required
3. Wait for OUTRDYIFG interrupt
4. Enter Low Power Mode (LPM) or change clock to LFCLK, if required

CPU_ERR_02**CPU Module****Category**

Functional

Function

Limitation of disabling prefetch/cache for CPUSS

Description

CPU prefetch/cache disable will not take effect if there is a pending flash memory access

Workaround

Disable the cache and the prefetcher (order is not mandatory), and then issue a memory access to the shutdown memory (SHUTDNSTORE) in SYSTCL (please check the SHUTDNSTORE registers in the devices TRM for more reference).

After the memory access completes, the prefetcher/cache will be disabled.

Example:

```

CPUSS->CTL = ( CPUSS_CTL_PREFETCH_DISABLE |
CPUSS_CTL_ICACHE_DISABLE); //disables instruction caching and pre-fetching
DL_SYSCTL_setShutdownStorageByte(DL_SYSCTL_SHUTDOWN_STORAGE_BYTE_X
, data) // Save a byte to SHUTDOWN memory

```

CPU_ERR_03**CPU Module****Category**

Functional

Function

Prefetcher can fetch wrong instructions when transitioning into Low power modes

Description

When transitioning into low power modes and there is a pending prefetch, the prefetcher can erroneously fetch incorrect data (all 0's). When the device wakes up, if the prefetcher and cache do not get overwritten by ISR code, then the main code execution from flash can get corrupted. For example, if the ISR is in the SRAM, then the incorrect data that was prefetched from Flash does not get overwritten. When the ISR returns the corrupted data in the prefetcher can be fetched by the CPU resulting in incorrect instructions. A HW Event wake is another example of a process that will wake the device, but not flush the prefetcher.

Workaround

Disable prefetcher before entering low power modes.

Example:

CPU_ERR_03 (continued)

CPU Module

CPUSS->CTL &= 0x6; // disables prefetcher, maintains other settings
 SYSCTL.SOCLOCK.SHUTDNSTORE0 // Read from SHUTDOWN Memory
 __WFI(); // or __WFE(); this function calls the transition into low power mode
 CPUSS->CTL |= 0x1; // enables prefetcher

FCC_ERR_01

FCC Module

Category

Functional

Function

FCC behaves incorrectly when BUSCLK is sourced from LFCLK

Description

When BUSCLK is sourced from LFCLK, FCC does not operate as expected. Either the FCC done signal does not assert, or an incorrect FCC value is reported.

Workaround

No workaround

FLASH_ERR_03

FLASH Module

Category

Functional

Function

Flash access with 2 wait states followed by invalid bootcode access will cause next flash access to also throw a violation

Description

Doing a Flash access followed by a access to BOOTCODE when you have 2 wait states will cause the next flash access to also cause a violation.

Workaround

Do not attempt to access boot-code region post-boot phase. Otherwise, there will need to be 4 clock cycles in between the bootcode violation and next correct flash access.

FLASH_ERR_04

FLASH Module

Category

Functional

Function

Wrong Address gets reported in the SYSCTL->DEDERRADDR

Description

When a FLASHDED error appears the data truncates the most significant byte. In the memory limits of the device, the most significant byte does not have an impact to the return address for MAIN flash. For NONMAIN flash or Factory region the MSB can be listed as 0x41xx.xxxx

Workaround

If the return address of the SYSCTL_DEDERRADDR returns a 0x00Cxxxxx, do an OR operation with 0x41000000 to get the proper address for the NONMAIN or Factory

FLASH_ERR_04
(continued)

FLASH Module

region return address. For example, if SYSCTL_DEDERRADDR = 0x00C4013C, the real address is 0x41C4013C.

For MAIN Flash DED, the SYSCTL_DEDERRADDR can be used as is.

FLASH_ERR_05

FLASH Module

Category

Functional

Function

DEDERRADDR can have incorrect reset value

Description

The reset value of the SYSCTL->DEDERRADDR can return a 0x00C4013C instead of the correct 0x00000000. The location of the error is in the Factory Trim region and is not indicative of a failure, it can be properly ignored. The reset value tends to change once NONMAIN has been programmed on the device.

Workaround

Accept 0x00C4013C as another reset value, so the default value from boot can be 0x00000000 or 0x00C4013C. The return value is outside of the range of the MAIN flash on the device so there is no potential of this return coming from an actual FLASH DED status.

FLASH_ERR_08

FLASH Module

Category

Functional

Function

Hard fault isn't generated for typical invalid memory region

Description

Hard fault isn't generated while trying to access illegal memory address space as shown below: 1. 0x010053FF - 0x20000000 2. 0x40BFFFFFF - 0x41C00000 3. 0x41C007FF - 0x41C40000

Workaround

No

GPIO_ERR_05

GPIO Module

Category

Functional

Function

Writing to GPIO DOUTTGL registers might get missed when a DMA transfer is ongoing

Description

The GPIO DMAMASK register information is mistakenly applied to a CPU write to the DOUTTGL register when a concurrent DMA transfer is in progress.

GPIO_ERR_05

(continued)

GPIO Module

Workaround

In the application code, ensure that the GPIO DMAMASK bit is set to 1 for the corresponding bit in the DOUTTGL register, before a CPU write access to the DOUTTGL register is issued. If no DMA transfer to any of the GPIO registers is required, the GPIO DMAMASK can be configured as 0xFFFFFFFF during the IO initialization step. This will solve the conflict of this errata. If the application also requires DMA write transfers to the GPIO registers, it is recommended that the application not use both DMA and CPU to write to the DOUTTGL register of the same GPIO module in the device. If the device has multiple GPIO modules, the DMA and the CPU can simultaneously write to the DOUTTGL register of different GPIO modules (while still requiring that the GPIO DMAMASK be configured for the GPIO module the CPU is writing to).

GPIO_ERR_06

GPIO Module

Category

Functional

Function

Writing to GPIO DOUT, DOUTSET and DOUTCLR registers might get missed when a DMA transfer is ongoing

Description

The GPIO DOUT, DOUTSET and DOUTCLR registers cannot be accessed by the DMA. Due to mistake in the implementation, the CPU access to the GPIO DOUT, DOUTSET and DOUTCLR will be also be blocked when a concurrent DMA transfer is in progress.

Workaround

In the application code, instead of writing to the DOUT, DOUTSET, and DOUTCLR registers, software should perform equivalent writes to the DOUTTGL register (see workaround GPIO_ERR_05 for restrictions on CPU writes to the DOUTTGL register).

In the pseudo code below, "pins" denotes the bit vector of pins in the GPIO module to be configured.

```
DL_GPIO_setPins(GPIO_Regs* gpio, uint32_t pins)
{
    gpio->DOUTTGL31_0 = ~(gpio->DOUT31_0) & pins;
}
```

```
DL_GPIO_clearPins(GPIO_Regs* gpio, uint32_t pins)
{
    gpio->DOUTTGL31_0 = gpio->DOUT31_0 & pins;
}
```

```
DL_GPIO_writePins(GPIO_Regs* gpio, uint32_t pins)
{
    gpio->DOUTTGL31_0 = ~(gpio->DOUT31_0) & pins;
    gpio->DOUTTGL31_0 = gpio->DOUT31_0 & (~pins);
}
```

```
DL_GPIO_writePinsVal(GPIO_Regs* gpio, uint32_t pinsMask, uint32_t pinsVal)
{
```

GPIO_ERR_06

(continued)

GPIO Module

```
uint32_t doutVal = gpio->DOUT31_0;
doutVal &= ~pinsMask;
doutVal |= (pinsVal & pinsMask);
gpio->DOU TTGL31_0 = ~(gpio->DOUT31_0) & doutVal;
gpio->DOU TTGL31_0 = gpio->DOUT31_0 & (~doutVal);
}
```

KEYSTORE_ERR_01**KEYSTORE Module****Category**

Functional

Function

STATUS.STAT value can be 0 or 1 without key access

Description

STATUS.STAT has a reset value of 1 and turns to 0 under these conditions: 1. After reset, debugger access via the register window returns 0x00. 2. After reset, the first CPU read returns 0x01, while subsequent CPU reads return 0x00. 3) After reset, first reading any other KEYSTORE register and then reading STATUS.STAT return 0x00.

Workaround

STATUS.STAT=0x0 means "No Error" . For checking if a slot is valid or not (Whether key is present), check STATUS.VALID.

MATHACL_ERR_01**MATHACL Module****Category**

Functional

Function

MATHACL status error bit does not get cleared

Description

If there is a status error generated by the mathacl (ex. divide by 0), then the status register never gets cleared.

Workaround

Reset the peripheral to clear the status bit.

PMCU_ERR_13**PMCU Module****Category**

Functional

Function

MCU may get stuck while waking up from STOP2 & STANDBY0

Description

If prefetch access is pending when the device transitions to STOP2 or STANDBY, when the device wakes up, the pending prefetch can prevent the device from resuming normal

PMCU_ERR_13

(continued)

PMCU Module

execution. The errata occurs if the WFI instruction is not word aligned, and the flash wait state is 2. In such a case, neither a DMA transfer nor a pending interrupt will be serviced.

Workaround

User should disable prefetch and issue a shutdown store memory read, which prevents a new prefetch from issuing and allows a pending prefetch to complete.

RST_ERR_01

RST Module

Category

Functional

Function

Device Remains in Reset When NRST Release Edge Coincides with LFOSCGOOD Rising Edge after A Loss of LFXT or LFCLK_IN

Description

In a specific corner case, if the release edge (low-to-high transition) of the NRST signal coincides with the rising edge of LFOSCGOOD, the device fails to detect the deassertion of NRST and remains in a reset state indefinitely. Conditions that might trigger the issue: When the LFCLK source is switched from LFCLK_IN or LFXT to LFOSC, the LFOSC is re-enabled and LFOSCGOOD status asserts high within 250us (min) to 1.5ms (max) of the re-enable event. If an NRST low pulse is applied such that its release edge (low-to-high transition) aligns with the LFOSCGOOD rising edge within a 50 ns window, the NRST deassertion is not detected and the device remains in reset.

Workaround

Keep the NRST low pulse width higher than 2ms to avoid this issue

RST_ERR_03

RST Module

Category

Functional

Function

RESETCAUSE Register Not Updated Following NRST Pulse (<1s) After LFCLK Monitor Failure

Description

When an NRST pulse is applied after LFCLK monitor failure, the RESETCAUSE register is not updated and retains its previous value (0x0) despite the reset occurring correctly.

Workaround

No workaround

SPI_ERR_10

SPI Module

Category

Functional

Function

DMA is not sampling the latest SPI trigger count

SPI_ERR_10

(continued)

SPI Module

Description

When SPI is configured to trigger a DMA transfer on a receive timeout (RTOUT) event, if the DMA channel is busy servicing another transfer at the time of the trigger request, any additional bytes received by SPI before the DMA acknowledges the request will not be included in the transfer. The DMA transfers only the number of bytes received when the trigger request was issued, leaving the subsequently received bytes in the RX FIFO unread.

Workaround

Configure DMA_TRIG_RX to use the RX trigger condition in combination with RTOUT. The RX trigger fires when the RX FIFO reaches the configured threshold level, ensuring the full expected number of bytes is present in the FIFO before DMA servicing begins. This guarantees the DMA transfer count is accurate regardless of whether the DMA channel is delayed in acknowledging the request.

SYSCTL_ERR_01 **SYSCTL Module**

Category

Functional

Function

SW-POR functionality is combined with HW-POR

Description

When a user writes to the LFSSRST register with the correct key to generate a software-triggered POR, the RSTCAUSE register will display 0x2 (indicating an NRST-triggered POR) instead of the expected 0x3 (Software-Triggered POR). This occurs because the SW-POR functionality is combined with the HW-POR path.

Workaround

No

SYSCTL_ERR_02 **SYSCTL Module**

Category

Functional

Function

SYSSTATUS.FLASHSEC is non-zero after a BOOTRST

Description

After BOOTRST/ bootcode completion SYSSTATUS.FLASHSEC is non-zero. This the customer will see after bootcode completion.

Workaround

No

SYSCTL_ERR_03 **SYSCTL Module**

Category

Functional

Function

DEDERRADDR persists after a SYSRESET or a write to the SYSSTATUSCLR

SYSCTL_ERR_03

(continued)

SYSCTL Module

Details

DEDERRADDR persists after either a SYSRESET or a write to the SYSSTATUSCLR register. Its value is overwritten only when a new FLASHDED error occurs. This behavior contradicts the Technical Reference Manual (TRM), which specifies its initial reset value as zero.

Workaround

No workaround

SYSCTL_ERR_04 ***SYSCTL Module***

Category

Functional

Function

SYSSTATUS.FLASHSEC is not cleared after a SYSRESET

Description

SYSSTATUS.FLASHSEC is not cleared after a SYSRESET and is only cleared by writing to the SYSSTATUSCLR register.

Workaround

Upon returning from SYSRESET, manually clear the FLASHSEC status with SYSSTATUSCLR = 0xCE000001.

SYSCTL_ERR_05 ***LFCLK Module***

Category

Functional

Function

LFCLK not working on exit from shutdown

Description

If LFCLK_IN pin is configured as a general input (or) LFCLK_IN function with pull-up, in this configuration, exiting shutdown mode will cause the LFCLK to be stuck.

Workaround

Choose either: 1.Enable pull-down instead of pull-up on this LFCLK_IN I/O 2.Avoid configuring it as an input

SYSCTL_ERR_06 ***CLK_OUT Module***

Category

Functional

Function

Glitch can occur on External Clock Output (CLK_OUT) when user disables the CLK_OUT while an asynchronous clock was selected as CLK_OUT source

Description

If the clock source for CLK_OUT is asynchronous with the current bus clock (for example, the bus clock is SYSOSC, while the clock source for CLK_OUT is selected as LFCLK), then in this case, glitches may appear on the CLK_OUT pin when it is enabled for a period of time and then disabled

SYSCTL_ERR_06

(continued)

CLK_OUT Module

Workaround

To avoid the glitch output to external pin, follow below sequence: CLK_OUT enable case: IOMUX enable configuration should be after CLK_OUT enable configuration. CLK_OUT disable case: IOMUX disable configuration should be before CLK_OUT disable configuration.

SYSCTL_ERR_11 **SYSCTL Module**

Category

Functional

Function

Unexpected BOR reset cycle happens when device in RUN2/SLEEP2 mode with FCL enabled

Description

When FCL is enable and device is running in RUN2/SLEEP2 mode, there is unexpected BOR followed by NMI happens for BOR1/2/3

Workaround

When using BOR1/2/3 monitor, avoid enabling FCL and making device run in RUN2 or SLEEP2 mode.

SYSCTL_ERR_12 **SYSCTL Module**

Category

Functional

Function

BOR events remain masked post a BOR1/2/3 when device in RUN2/SLEEP2 mode with FCL enabled

Description

When FCL is enabled in RUN2 or SLEEP2 power modes, BOR mask logic does not work as expected, resulting in the failure to detect all subsequent BOR events following the first BOR occurrence. Until software re-enable SYSOSC, BOR MASK is cleared.

Workaround

Workaround 1: Avoid enabling FCL and make device in RUN2/SLEEP2 mode, when BOR1/2/3 detections are needed Workaround 2: Manually clear the BOR mask flag in the BOR/NMI handler after the initial masking period has elapsed Workaround3: Implement a software-based refresh mechanism by periodically toggling Sysosc enable/disable to clear the BOR mask flag

SYSOSC_ERR_01 **SYSOSC Module**

Category

Functional

Function

MFCLK drift when using SYSOSC FCL together with STOP1 mode

Description

When MFCLK is enabled AND SYSOSC is using the frequency correction loop (FCL) mode AND the STOP1 low power operating mode is used, THEN the MFCLK may drift by

SYSOSC_ERR_01

(continued)

SYSOSC Module

2 cycles when SYSOSC shifts from 4MHz back to 32MHz (either upon exit from STOP1 to RUN mode or upon an asynchronous fast clock request that forces SYSOSC to 32MHz).

Workaround

Workaround1: Use STOP0 mode instead of STOP1 mode. There is no MFCLK drift when STOP0 mode is used. Workaround2: Do not use SYSOSC in the FCL mode (leave FCL disabled) when using STOP1.

SYSOSC_ERR_02 SYSOSC Module

Category

Functional

Function

MFCLK does not work when Async clock request is received in an LPM where SYSOSC was disabled in FCL mode

Description

MFCLK will not start to toggle in below scenario:
 1. FCL mode is enabled and then MFCLK is enabled
 2. Enter a low power mode where SYSOSC is disabled (SLEEP2/STOP2/STANDBY0/STANDBY1).
 3. Async request is received from some peripherals which use MFCLK as functional clock. On receiving async request, SYSOSC gets enabled and ulpclk becomes 32MHz. But MFCLK is gated off and it does not toggle at all as the device is still set to the LPM.

Workaround

If SYSOSC is using the FCL mode - Do not enable the MFCLK for a peripheral when you're entering a LPM mode which would typically turn off the SYSOSC.

SYSOSC_ERR_04 SYSOSC Module

Category

Functional

Function

SYSOSC Accuracy Degradation in FCL ON Mode

Description

When using the FCL ON mode of the internal oscillator, SYSOSC, accuracy can degrade up to +/-3%. The accuracy degradation is due to a synchronization between the 4MHz FCL sampling clock and noise in the system caused by other peripherals operating at harmonics of 4MHz.

Workaround

There is no workaround to completely remove noise from the system. However, if using the FCL ON mode, the degradation can be minimized when peripherals are operating at frequencies that are not multiples of 4MHz.

SYSOSC_ERR_05 SYSOSC Module

Category

Functional

SYSOSC_ERR_05

(continued)

SYSOSC Module**Function**

SYSOSC May Operate Below Base Freq During Async Fast Wakeup from LPM When FCL Is Enabled

Description

When FCL=enabled, SYSOSC may operate up to 10% below its base frequency during low power modes and while there is an active asynchronous fast clock request. This occurs while the device is in low power modes because the FCL = Disabled trim is always applied instead of using FCL= enabled trim even though FCL = enabled. Once the device has woken up, exited LPM, and serviced the request, SYSOSC applies correct trim resumes normal FCL = Enabled operation at its intended target frequency.

Most use cases should have no meaningful impact and can continue to use both FCL and Async fast clock requests as needed. The main applications in which this erratum will have meaningful impact are those where UART is the source of the async fast clock request, as this temporary shift could have impact on the effective baud rate until the device fully wakes.

Workaround

1. Keep FCL = disabled
2. If FCL =Enabled needed, disable Async fast clock requests.

SYSOSC_ERR_06 SYSOSC Module**Category**

Functional

Function

MFCLK work with 32MHz instead of 4MHz when async clock request is received in an LPM where SYSOSC was disabled in FCL mode

Description

MFCLK is incorrectly configured to 32MHz in below scenario:

1. FCL mode is enabled and then MFCLK is enabled
2. Enter a low power mode where SYSOSC is disabled (SLEEP2/STOP2/STANDBY0/STANDBY1)
3. Async request is received from some peripherals which use MFCLK as functional clock
On receiving async request, SYSOSC gets enabled and ULPCLK becomes 32MHz. Then MFCLK is incorrectly configured to 32MHz.

Workaround

If SYSOSC is using the FCL mode, do NOT enable the MFCLK for a peripheral when you're entering a LPM mode which would typically turn off the SYSOSC.

SYSPLL_ERR_01 SYSPLL Module**Category**

Functional

Function

SYSPLL Frequency may not lock to correct frequency when enabled.

SYSPLL_ERR_01

(continued)

SYSPLL Module

Description

When setting the SYSPLLEN bit to 1 in SYSCTL HSCLKEN register, the SYSPLL will run the phase locked loop search. The search can potentially fail where the frequency will not be set to the correct value, instead the resultant frequency will be drastically different than the configured frequency.

Workaround

Frequency Verification Process

Monitor the SYSPLL frequency output using the Frequency Clock Counter (FCC) whenever the SYSPLLEN bit is set to 1. Once the correct frequency is established, it will remain stable until the SYSPLL is disabled and re-enabled (SYSPLLEN bit toggled from 0 to 1). If an incorrect frequency is detected, disable and re-enable the SYSPLL to perform another verification.

Workaround 1: FCC Count Check

Use LFCLK as the FCC trigger source to count the SYSPLL output clock frequency. Execute the FCC and verify the measured value against the configured SYSPLL frequency using LFCLK as reference.

Example calculation:

- SYSPLLCLK0 = 80MHz ; LFCLK = 32.768kHz
- Measured FCC Count = $80,000,000 / 32,768 = 2,441$

FCC Count Tolerance:

The real FCC count will vary depending on the combined clock accuracies (SYSPLLCLK0 and LFCLK). Recommend to add +/- 5~10% to allowed FCC check range.

- FCC count upper limit = $2,441 * 1.05 = 2,563$
- FCC count lower limit = $2,441 * 0.95 = 2,318$

Timing considerations:

- Clock synchronization time: 5-6 LFCLK cycles
- FCC trigger time: 1-32 LFCLK cycles (user-configurable)

Register Configuration:

- FCC Settings: SYSCTL.GENCLKCFG.FCCTRIGSRC = 1;
- SYSCTL.GENCLKCFG.FCCLVLTRIG = 0;
- SYSCTL.GENCLKCFG.FCCTRIGCNT = 0;
- SYSCTL.GENCLKCFG.FCCSELCLK = 4;
- Start FCC: SYSCTL.FCCCMD = 0x0E000001U
- Check FCC Done Status: SYSCTL.CLKSTATUS.FCCDONE
- Read FCC Count: SYSCTL.FCC

Timeout Protection:

Implement a software-based timeout during FCC Done status monitoring to prevent longer wait time under the condition the unlocked SYSPLL frequency is less than FCC trigger clock frequency (LFCLK).

```
fccTimeoutCounter = 0;
while (DL_SYSCTL_isFCCDone() == 0) {
    delay_cycles(977); /* 1x LFCLK cycle = 32MHz/32.768kHz */
    fccTimeoutCounter++;
    if(fccTimeoutCounter > 65) break;
    /* Timeout set to approximately 2ms (user-customizable) */
}
```

FCC Check Restart:

If the FCC measurement falls outside the expected range, disable and re-enable the SYSPLL (set SYSPLLEN to 0, then 1) and repeat the FCC verification.

Workaround 2: FCC Ratio Check

SYSPLL_ERR_01

(continued)

SYSPLL Module

Use LFCLK as the FCC trigger source to count both SYSPLL output and input clock frequency. Execute the FCC and verify the measured ratio of the FCC check value between output and input clock to the expected ratio.

Example calculation:

- SYSPLL = 80MHz ; HFCLK = 40MHz ; LFCLK = 32.768kHz
- Expect clock ratio = 80MHz/40MHz = 2.0000
- Measured FCC count (SYSPLL) = 80,000,000/32,768 = 2,441
- Measured FCC count (HFCLK) = 40,000,000/32,768 = 1,220
- Measured clock ratio = 2,441/1,220 = 2.0008

FCC Ratio Tolerance:

The FCC ratio method eliminates combined clock accuracy errors and depends only on FCC uncertainty (2 counted clock cycles) and calculation rounding error. This allows for much tighter tolerance ranges compared to FCC count check method, for example +/- 0.3%.

Timing considerations:

- Clock synchronization time: 5-6 LFCLK cycles
- FCC trigger time: 1-32 LFCLK cycles (user-configurable)
- Total time per complete FCC ratio check: 2*(sync time + trigger time)

FCC Ratio Check Flow:

1. Configure FCC for SYSPLL output clock (SYSPLL0 or SYSPLL2X)
2. Start FCC and wait for FCC done (Add Timeout Protection, refer to FCC Count Check)
3. Read FCC check count back
4. Configure FCC for SYSPLL input clock (SYSOSC or HFCLK)
5. Start FCC and wait for FCC done (Add Timeout Protection, refer to FCC Count Check)
6. Read FCC check count back
7. Calculate the FCC check ratio and compared to the expected ratio range
8. If the FCC ratio falls outside the expected range, disable and re-enable the SYSPLL (set SYSPLLEN to 0, then 1) and repeat the FCC ratio verification.

TIMER_ERR_04**TIMER Module****Category**

Functional

Function

TIMER re-enable may be missed if done close to zero event

Description

When using a TIMER in one shot mode, TIMER re-enable may be missed if done close to zero event. The HW update to the timer enable bit will take a single functional clock cycle, if you use 32.768kHz and divider of 3, then it will take ~100us to have the enable bit set to 0 properly.

Workaround

Wait 1 functional clock cycle before re-enabling the timer OR the timer can be disabled first before re-enabling. Disable the counter with CTRCTL.EN = 0, then reenable with CTRCTL.EN = 1

TIMER_ERR_06**TIMG Module****Category**

Functional

TIMER_ERR_06

(continued)

TIMG Module

Function

Writing 0 to CLKEN bit does not disable counter

Description

Writing 0 to the Counter Clock Control Register(CCLKCTL) Clock Enable bit(CLKEN) does not stop the timer.

Workaround

Stop the timer by writing 0 to the Counter Control(CTRCTL) Enable(EN) bit.

TIMER_ERR_07

Initial repeat counter has 1 less period than next repeats Module

Category

Functional

Function

TIMER

Description

When using the timer repeat counter mode, the first repeat will have 1 less count than the subsequent repeats because the following repeat counters will include the transition between 0 and the load value. For example if the TIMx.RCLD = 0x3 then 3 observable zero events would appear on the first repeat counter and 4 observable zero events would appear on the following repeat counter sequences.

Workaround

Set the initial RCLD value to 1 more than the expected RCLD, then in the ISR for the Repeat Counter Zero Event (REPC), set the RCLD to the intended RCLD value. For example, if intending to have 4 repeats, set the initial RCLD value to RCLD = 0x5, then in the timer ISR for the REPC interrupt, set RCLD = 0x4. Now all timer repeats will have the same number of zero/load events.

UNICOMMI2CC_ERR_01

UNICOMMI2CC Module

Category

Functional

Function

I2C Controller BUSY Status Polling Issue

Description

When initiating an I2C controller transfer by setting the BUSRTRUN/FRAME_START bit, the BUSY status flag requires approximately 2-3 I2C functional clock cycles to assert. If polling the BUSY bit immediately after setting BUSRTRUN/FRAME_START, the status might be checked by the application before it's properly set. This issue is more pronounced with higher CLKDIV values (resulting in slower I2C functional clock) or under higher compiler optimization levels.

Workaround

Add a software delay before polling BUSY status. The recommended delay should be:
Software delay = 3 x I2C functional clock = 3 x clock_divider x (CPU_CLK / selected clock source frequency) For example, with a clock_divider of 2, a clock source of 4 MHz(MFCLK), and CPU_CLK of 80 MHz: Software delay = 3 x 2 x (80 MHz / 4 MHz)= 120 CPU cycles

**UNICOMMI2CC_E
RR_02**
UNICOMMI2CC Module

Category

Functional

Function

RXDONE interrupt is triggered twice when I2C controller ACKOEN bit is enabled

Description

When I2C controller CTR.ACKONE bit is enable, software controls the sending of ACK/NACK by writing CTR.ACK bit 0 or 1. This can be done in the RXDONE interrupt before the stop condition. After writing the CTR.ACK bit, there will be valid ACK/NACK signal on I2C bus, and an additional RXDONE interrupt will be triggered in the subsequent I2C stop status.

Workaround

Disable the RXDONE interrupt before writing CTR.ACK bit, and manually enable it before next transmission.

**UNICOMMI2CC_E
RR_03**
UNICOMMI2CC Module

Category

Functional

Function

I2C controller SCL frequency is lower when clock stretching function is enable

Description

- Please do never recommend to use an ETW feature as a workaround. - Please check that the ERRATA proposal does not compromise other existing ERRATAs on this device revision or previous revisions. When clock stretching is enabled, CR.CLKSTRETCH = 1, SCL frequency will be lower than the settings. For example, I2C controller bus speed is set to 400kHz, if clock stretching function is enabled, I2C controller bus clock will be lower than 400kHz, for example, 360kHz or 390kHz. Same phenomenon will be seen on other bus speed, for example, 100kHz or 1Mhz.

Workaround

Not recommended to disable the clock stretching function, because there is no large impact to I2C communication function. After disable the clock stretching function, expected I2C bus clock speed will be seen.

**UNICOMMSPI_ER
R_02**
UNICOMM SPI Module

Category

Functional

Function

Controller Side Busy status read incorrectly

Description

SPI busy status doesn't depend on the TXFIFO's data elements, so essentially once we load the SPI TXFIFO with some data, the busy bit will not go high instantly in case the CLKSEL is a slower clock (LFCLK for instance) or CLKDIV > /1. This leads to incorrect status being read during the time between loading TXFIFO data and until the communication actually starts since the SPI source clock is slower.

UNICOMMSPI_ER

R_02 (continued) *UNICOMM SPI Module*

Workaround

Can introduce a check for TXFIFO empty STATUS bit when writing application code.

UNICOMMSPI_ER

R_03 *UNICOMM SPI Module*

Category

Functional

Function

RTOUT won't be set at second time when CPU is used to clear RXFIFO

Description

After SPI communication is completed with rxtimeout enabled and then RTOUT is set, if use CPU to read out the RXFIFO data and read IIDX to clear the interrupt but don't do any reset or disable the IP, the RTOUT interrupt won't be set when you do the same transmission at the second time even though rxtimeout counter is "0" and RXFIFO is not empty.

Workaround

Use DMA to read out the RXFIFO data instead of using CPU directly .Or reset the UNICOMM SPI after reading out the RXFIFO data using CPU directly.

UNICOMMSPI_ER

R_04 *UNICOMM SPI Module*

Category

Functional

Function

RX stored data is incorrect when do re-transfer after last one transfer with DSS mismatch

Description

When sending data with different data packet size setting (DSS) on controller & peripheral side and after that doing RXCLR to clear the FIFO, again sending data with proper data package size but still older data is coming into RXFIFO when first SCLK pulse comes.

Workaround

Please make sure the data package size setting (DSS) on controller side and peripheral side is same.

UNICOMMUART_E

RR_06 *UNICOMMUART Module*

Category

Functional

Function

RTOUT/LTOUT computation issues due to STOP bit handling

Description

On the receiver side, the function state machine transitions from STOP bit to IDLE at the middle of the STOP bit. This causes the receive timeout (RTOUT) & line timeout (LTOUT) counter to start counting at the middle of the STOP bit and not at its end. Leading to

UNICOMMUART_E**RR_06 (continued) UNICOMMUART Module**

RTOUT/LTOUT being triggered half a baud-period early. This is especially pronounced for low-baud rates with higher UART functional clock frequency.

Workaround

Add compensation with a half stop bit period to the RTOUT counter.

UNICOMMUART_E**RR_07 UNICOMMUART Module**

Category

Functional

Function

RTS line does not go HIGH if UART is disabled in RS-232 mode

Description

When UART is disabled, the RTS line fails to return to its idle state (HIGH), remaining stuck at LOW.

Workaround

Use software to enable the internal pull-up resistor and set the RTS line IO to Hiz mode.

UNICOMMUART_E**RR_09 UNICOMMUART Module**

Category

Functional

Function

ISO-7816 Smartcard Mode baud rate limitation

Description

To achieve a 9600 baud rate in ISO-7816 Smartcard Mode, a UARTCLK frequency exceeding 57 MHz is required due to the following constraints: 1. The ISO-7816 standard requires 372 clock cycles per bit 2. In MSPM0 ISO-7816 mode, the oversampling rate (OVS) is fixed at 16x in the UART peripheral Minimum UARTCLK calculation: Required UARTCLK = $9600 \times 372 \times 16 = 57.139 \text{ MHz}$

Workaround

UART with lower input frequency will not be able to support SMARTCARD mode.

UNICOMMUART_E**RR_10 UNICOMMUART Module**

Category

Functional

Function

LIN Registers CLKDIV Restriction

Description

When CLKDIV value is other than 0, writing into LINC0/1 will not have any effect.

UNICOMMUART_E

RR_10 (continued) *UNICOMMUART Module*

Workaround

To properly configure LIN registers: 1. First set CLKDIV to '0' 2. Update the LINC0/1 register configurations with desired values 3. Restore CLKDIV to its intended operating value

UNICOMMUART_E

RR_12 *UNICOMMUART Module*

Category

Functional

Function

CTL0.TXE bit setting to 0 in middle of frame makes current transmission error

Description

Start transmitting packet and then in middle of transfer, set the CTL0.TXE bit to 0, the current packet won't complete and the TX line will be pulled low, which shows the busy behavior as BUSY. The TX line will be released only when CTL0.TXE = 1, and till then BUSY remains 1.

Workaround

User must poll for BUSY STATUS to go low and CTL0.TXE = 0, so as to avoid this issue.

UNICOMMUART_E

RR_13 *UNICOMMUART Module*

Category

Functional

Function

RTOUT interrupt coming 1 fclk cycle earlier

Description

RTOUT (Receive timeout) interrupt coming 1 fclk cycle earlier from the exact time which it should come for.

Workaround

The way user can take care of this is to expect the RTOUT interrupt to come one fclk cycle earlier.

UNICOMMUART_E

RR_14 *UNICOMMUART Module*

Category

Functional

Function

CTS interrupt gets toggled on both falling and rising edges of CTS line

Description

CTS interrupt gets toggled on both RISE edge as well as FALL edge of CTS line. Ideally, it should just toggle on the FALL edge of CTS line.

UNICOMMUART_E**RR_14** (continued) **UNICOMMUART Module****Workaround**

Instead of RIS bit, user can monitor STAT.CTS bit's toggle to 1, since this toggling of STAT.CTS = 1 indicates that falling of CTS line has been encountered.

UNICOMMUART_E**RR_15** **UNICOMMUART Module****Category**

Functional

Function

LCRH.SENDIDLE access missed in between fclk and new IDLELINE frame not produced

Description

If toggle LCRH.SENDIDLE from 0 to 1 and then back from 1 to 0, or toggle from 1 to 0 and then back from 0 to 1 in between two fclk edges, the toggle will be missed and no new IDLELINE frame is sent.

Workaround

After toggling LCRH.SENDIDLE, don't put it back immediately, please keep more than one fclk period, because LCRH.SENDIDLE is sampled on fclk.

VREF_ERR_05**VREF Module****Category**

Functional

Function

VREF READY0 status goes low for VREF0 when COMP is configured in Sampled (ultra-low-power) Mode

Description

The VREF READY0 status remains low for VREF0 when the comparator is configured in sampled (ultra-low-power) mode. The comparator is configured in sampled mode when the REFMODE bit is set to 1.

Workaround

Do not poll for the VREF READY0 status after the comparator is configured in sampled mode. Configure the comparator in static (fast) mode before polling the READY0 status for VREF0 or poll the READY0 status before enabling the comparator.

8 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from May 31, 2026 to October 5, 2026 (from Revision A (May 2026) to Revision B (October 2026))

	Page
• BSL_ERR_01 Advisory was added.....	5
• COMP_ERR_06 Module was updated.....	5
• COMP_ERR_06 Function was updated.....	5
• COMP_ERR_06 Description was updated.....	5
• COMP_ERR_06 Workaround was updated.....	5
• FCC_ERR_01 Workaround was updated.....	7

• FCC_ERR_01 Function was updated.....	7
• FCC_ERR_01 Description was updated.....	7
• FCC_ERR_01 Added workaround	7
• Added FLASH_ERR_03.....	7
• RST_ERR_01 Function was updated.....	11
• RST_ERR_01 Description was updated.....	11
• RST_ERR_01 Workaround was updated.....	11
• RST_ERR_03 Module was updated.....	11
• RST_ERR_03 Function was updated.....	11
• RST_ERR_03 Description was updated.....	11
• RST_ERR_03 Workaround was updated.....	11
• SPI_ERR_10 Module was updated.....	11
• SPI_ERR_10 Function was updated.....	11
• SYSCTL_ERR_11 Module was updated.....	14
• SYSCTL_ERR_11 Function was updated.....	14
• SYSCTL_ERR_11 Workaround was updated.....	14
• SYSCTL_ERR_11 Description was updated.....	14
• SYSCTL_ERR_12 Module was updated.....	14
• SYSCTL_ERR_12 Function was updated.....	14
• SYSCTL_ERR_12 Description was updated.....	14
• SYSCTL_ERR_12 Workaround was updated.....	14
• SYSOSC_ERR_04 Workaround was updated.....	15
• SYSOSC_ERR_04 Function was updated.....	15
• SYSOSC_ERR_04 Description was updated.....	15
• SYSOSC_ERR_05 Function was updated.....	15
• SYSOSC_ERR_05 Workaround was updated.....	15
• SYSOSC_ERR_05 Description was updated.....	15
• SYSOSC_ERR_06 Module was updated.....	16
• SYSOSC_ERR_06 Function was updated.....	16
• SYSOSC_ERR_06 Description was updated.....	16
• SYSOSC_ERR_06 Workaround was updated.....	16
• UNICOMMI2CC_ERR_03 Module was updated.....	20
• UNICOMMI2CC_ERR_03 Function was updated.....	20
• UNICOMMI2CC_ERR_03 Description was updated.....	20
• UNICOMMI2CC_ERR_03 Workaround was updated.....	20
• UNICOMMSPI_ERR_02 Module was updated.....	20
• UNICOMMSPI_ERR_02 Function was updated.....	20
• UNICOMMSPI_ERR_02 Description was updated.....	20
• UNICOMMSPI_ERR_02 Workaround was updated.....	20
• UNICOMMSPI_ERR_03 Module was updated.....	21
• UNICOMMSPI_ERR_03 Function was updated.....	21
• UNICOMMSPI_ERR_03 Description was updated.....	21
• UNICOMMSPI_ERR_03 Workaround was updated.....	21
• UNICOMMSPI_ERR_04 Module was updated.....	21
• UNICOMMSPI_ERR_04 Function was updated.....	21
• UNICOMMSPI_ERR_04 Description was updated.....	21
• UNICOMMSPI_ERR_04 Workaround was updated.....	21
• UNICOMMUART_ERR_12 Advisory was added.....	23
• UNICOMMUART_ERR_13 Advisory was added.....	23
• UNICOMMUART_ERR_14 Advisory was added.....	23
• UNICOMMUART_ERR_15 Advisory was added.....	24

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#), [TI's General Quality Guidelines](#), or other applicable terms available either on [ti.com](#) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Copyright © 2026, Texas Instruments Incorporated

Last updated 10/2025